

Genetic and Evolutionary Computation

Wolfgang Banzhaf
Penousal Machado
Mengjie Zhang *Editors*

Handbook of Evolutionary Machine Learning



Springer

Genetic and Evolutionary Computation

Series Editors

Wolfgang Banzhaf , Department of Computer Science and Engineering,
Michigan State University, East Lansing, MI, USA

Kalyanmoy Deb , Department of Electrical and Computer Engineering, Michigan
State University, East Lansing, MI, USA

The area of Genetic and Evolutionary Computation has seen an explosion of interest in recent years. Methods based on the variation-selection loop of Darwinian natural evolution have been successfully applied to a whole range of research areas.

The Genetic and Evolutionary Computation Book Series publishes research monographs, edited collections, and graduate-level texts in one of the most exciting areas of Computer Science. As researchers and practitioners alike turn increasingly to search, optimization, and machine-learning methods based on mimicking natural evolution to solve problems across the spectrum of the human endeavor, this growing field will continue to surprise with novel applications and results. Recent award-winning PhD theses, special topics books, workshops and conference proceedings in the areas of EC and Artificial Life Studies are of interest.

Areas of coverage include applications, theoretical foundations, technique extensions and implementation issues of all areas of genetic and evolutionary computation. Topics may include, but are not limited to:

Optimization (multi-objective, multi-level) Design, control, classification, and system identification Data mining and data analytics Pattern recognition and deep learning Evolution in machine learning Evolvable systems of all types Automatic programming and genetic improvement

Proposals in related fields such as:

Artificial life, artificial chemistries Adaptive behavior and evolutionary robotics Artificial immune systems Agent-based systems Deep neural networks Quantum computing will be considered for publication in this series as long as GEVO techniques are part of or inspiration for the system being described. Manuscripts describing GEVO applications in all areas of engineering, commerce, the sciences, the arts and the humanities are encouraged.

Prospective Authors or Editors:

If you have an idea for a book, we would welcome the opportunity to review your proposal. Should you wish to discuss any potential project further or receive specific information regarding our book proposal requirements, please contact Wolfgang Banzhaf, Kalyan Deb or Mio Sugino:

Areas: Genetic Programming/other Evolutionary Computation Methods, Machine Learning, Artificial Life

Wolfgang Banzhaf Consulting Editor BEACON Center for Evolution in Action Michigan State University, East Lansing, MI 48824 USA banzhafw@msu.edu

Areas: Genetic Algorithms, Optimization, Meta-Heuristics, Engineering

Kalyanmoy Deb Consulting Editor BEACON Center for Evolution in Action Michigan State University, East Lansing, MI 48824 USA kdeb@msu.edu

Mio Sugino mio.sugino@springer.com

The GEVO book series is the result of a merger the two former book series: Genetic Algorithms and Evolutionary Computation <https://link.springer.com/bookseries/6008> and Genetic Programming <https://link.springer.com/bookseries/6016>.

Wolfgang Banzhaf · Penousal Machado ·
Mengjie Zhang
Editors

Handbook of Evolutionary Machine Learning



Springer

Editors

Wolfgang Banzhaf  Department of Computer Science and Engineering Michigan State University East Lansing, MI, USA

Penousal Machado Department of Informatics Engineering University of Coimbra Coimbra, Portugal

Mengjie Zhang School of Engineering and Computer Science and Centre for Data Science and Artificial Intelligence Victoria University of Wellington Wellington, New Zealand

ISSN 1932-0167 ISSN 1932-0175 (electronic)
Genetic and Evolutionary Computation
ISBN 978-981-99-3813-1 ISBN 978-981-99-3814-8 (eBook)
<https://doi.org/10.1007/978-981-99-3814-8>

© The Editor(s) (if applicable) and The Author(s), under exclusive license to Springer Nature Singapore Pte Ltd. 2024

This work is subject to copyright. All rights are solely and exclusively licensed by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Singapore Pte Ltd. The registered company address is: 152 Beach Road, #21-01/04 Gateway East, Singapore 189721, Singapore

Paper in this product is recyclable.

Preface

The *Handbook of Evolutionary Machine Learning* in front of you is the work of many authors. Even if the field is still small compared to either Machine Learning or Evolutionary Computation, no single author can any more hold all the methods, techniques, and applications present in their head.

A look at the references cited by the authors in each of these chapters reveals that many of the developments of the last decade, which really is the time of visible exponential growth of this new field, have been in the making for many more years. The history of Evolutionary Machine Learning is rich, and as a tree in nature, it has many roots.

The editors of this handbook have strived to give voice to a multitude of research directions in the field. Initially, it was not easy to structure all the material appropriately, but we believe we have arrived at a good balance of the presentation as the book grew.

We start with the basics of the field in Part I, then report on the interpretation of some Evolutionary Computation techniques as Machine Learning (Part II). We next delve deeper into the modern understanding of Machine Learning as embodied by Neural Networks, with its exciting methods of generative systems in Part III. Part IV reports on the applications of Evolutionary Computation in Machine Learning, which exemplifies the beneficial hybridization of both techniques. Finally, Part V discusses applications of Evolutionary Machine Learning in diverse fields, providing many fascinating perspectives.

This handbook is primarily intended for researchers, engineers, graduate and post-graduate students interested in evolutionary machine learning, evolutionary computation, machine learning, and broad artificial intelligence and use them for different applications. It is highly suitable as a reference book for postgraduate research students with directions in evolutionary machine learning, where they can use the techniques in this handbook as a baseline for further improvement and innovations. It is also suitable to use a selection of chapters for postgraduate and high-level undergraduate courses in artificial intelligence, data science and machine learning majors. Furthermore, this handbook can also contribute potentially to real-world applications

in primary industry, climate change, (bio)medical and health, cybersecurity, renewable energy, high-value manufacturing, economy and finance, and public policies and decision-making that generate impact on the economy, environment, health outcome and social aspects.

East Lansing, MI, USA
Coimbra, Portugal
Wellington, New Zealand
July 2023

Wolfgang Banzhaf
Penousal Machado
Mengjie Zhang

Acknowledgments

In 2020, the annual EvoStar series of conferences established the EvoApps special session on Evolutionary Machine Learning. Over time, the event grew organically, eventually becoming a joint track of EuroGP and EvoApps. The increasing interest in the field and the high quality of the submissions made us realize that the time was ripe to bring forth this book. Therefore, the origins of this book are closely connected with these EvoStar events, and we seize this opportunity to extend our heartfelt gratitude to all the EvoStar chairs, organizers, authors and participants with whom we have had the privilege of collaborating and engaging over the years.

We express our appreciation to our colleagues and friends in the field who have contributed to one or more chapters of this handbook. Thanks should also go to the reviewers who provided constructive comments, suggestions and feedback to the chapters' authors, improving the handbook through this process. We appreciate the help of MSU postdoctoral fellow Dr. Jory Schossau, who helped create this volume's index.

The sabbatical leaves of Penousal Machado, which was partially spent at Michigan State University with Wolfgang Banzhaf, and Wolfgang Banzhaf's sabbatical, partially spent at Victoria University of Wellington University with Mengjie Zhang, played a crucial role in the design and development of this book. They allowed us to collaborate closely, filling whiteboards with sketches, taxonomies, ideas and topics, many of which have ultimately made it into this book. We sincerely thank our institutions, the University of Coimbra, Portugal, Michigan State University, USA, and Victoria University of Wellington, New Zealand, for their invaluable support and the exceptional conditions they have provided us and our guests.

Finally, we thank the publisher, Springer-Nature, and particularly Computer Science book editor Mio Sugino, who supported the handbook from its inception to its final publication in the Genetic and Evolutionary Computation book series.

East Lansing, MI, USA
Coimbra, Portugal
Wellington, New Zealand
July 2023

Wolfgang Banzhaf
Penousal Machado
Mengjie Zhang

Contents

Part I Evolutionary Machine Learning Basics

1 Fundamentals of Evolutionary Machine Learning	3
Wolfgang Banzhaf and Penousal Machado	
2 Evolutionary Supervised Machine Learning	29
Risto Miikkulainen	
3 EML for Unsupervised Learning	59
Roberto Santana	
4 Evolutionary Computation and the Reinforcement Learning Problem	79
Stephen Kelly and Jory Schossau	

Part II Evolutionary Computation as Machine Learning

5 Evolutionary Regression and Modelling	121
Qi Chen, Bing Xue, Will Browne, and Mengjie Zhang	
6 Evolutionary Clustering and Community Detection	151
Julia Handl, Mario Garza-Fabre, and Adán José-García	
7 Evolutionary Classification	171
Bach Nguyen, Bing Xue, Will Browne, and Mengjie Zhang	
8 Evolutionary Ensemble Learning	205
Malcolm I. Heywood	

Part III Evolution and Neural Networks

9 Evolutionary Neural Network Architecture Search	247
Zeqiong Lv, Xiaotian Song, Yuqi Feng, Yuwei Ou, Yanan Sun, and Mengjie Zhang	

10	Evolutionary Generative Models	283
	João Correia, Francisco Baeta, and Tiago Martins	
11	Evolution Through Large Models	331
	Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O. Stanley	
12	Hardware-Aware Evolutionary Approaches to Deep Neural Networks	367
	Lukas Sekanina, Vojtech Mrazek, and Michal Pinos	
13	Adversarial Evolutionary Learning with Distributed Spatial Coevolution	397
	Jamal Toutouh, Erik Hemberg, and Una-May O'Reilly	

Part IV Evolutionary Computation for Machine Learning

14	Genetic Programming as an Innovation Engine for Automated Machine Learning: The Tree-Based Pipeline Optimization Tool (TPOT)	439
	Jason H. Moore, Pedro H. Ribeiro, Nicholas Matsumoto, and Anil K. Saini	
15	Evolutionary Model Validation—An Adversarial Robustness Perspective	457
	Inês Valentim, Nuno Lourenço, and Nuno Antunes	
16	Evolutionary Approaches to Explainable Machine Learning	487
	Ryan Zhou and Ting Hu	
17	Evolutionary Algorithms for Fair Machine Learning	507
	Alex Freitas and James Brookhouse	

Part V Applications of Evolutionary Machine Learning

18	Evolutionary Machine Learning in Science and Engineering	535
	Jianjun Hu, Yuqi Song, Sadman Sadeed Omee, Lai Wei, Rongzhi Dong, and Siddharth Gianey	
19	Evolutionary Machine Learning in Environmental Science	563
	João E. Batista and Sara Silva	
20	Evolutionary Machine Learning in Medicine	591
	Michael A. Lones and Stephen L. Smith	
21	Evolutionary Machine Learning for Space	611
	Moritz von Looz, Alexander Hadjiivanov, and Emmanuel Blazquez	
22	Evolutionary Machine Learning in Control	629
	Guy Y. Cornejo Maceda and Bernd R. Noack	

Contents	xi
23 Evolutionary Machine Learning in Robotics	657
Eric Medvet, Giorgia Nadizar, Federico Pigozzi, and Erica Salvato	
24 Evolutionary Machine Learning in Finance	695
Michael O'Neill and Anthony Brabazon	
25 Evolutionary Machine Learning and Games	715
Julian Togelius, Ahmed Khalifa, Sam Earle, Michael Cerny Green, and Lisa Soros	
26 Evolutionary Machine Learning in the Arts	739
Jon McCormack	
Index	761

Contributors

Nuno Antunes University of Coimbra, CISUC, DEI, Coimbra, Portugal

Francisco Baeta Department of Informatics Engineering, Centre for Informatics and Systems of the University of Coimbra, University of Coimbra, Coimbra, Portugal

Wolfgang Banzhaf Department of Computer Science and Engineering, Michigan State University, East Lansing, MI, USA

João E. Batista LASIGE, Department of Informatics, Faculty of Sciences, University of Lisbon, Lisboa, Portugal

Emmanuel Blazquez European Space Agency ESA, Noordwijk, Netherlands

Anthony Brabazon Natural Computing Research & Applications Group, School of Business, University College Dublin, Dublin, Ireland

James Brookhouse School of Computing, University of Kent, Canterbury, UK

Will Browne Faculty of Engineering, School of Electrical Engineering and Robotics, Queensland University of Technology, Brisbane, Australia

Qi Chen Evolutionary Computation Research Group at the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand

Guy Y. Cornejo Maceda School of Mechanical Engineering and Automation, Harbin Institute of Technology (Shenzhen), Shenzhen, PR China

João Correia Department of Informatics Engineering, Centre for Informatics and Systems of the University of Coimbra, University of Coimbra, Coimbra, Portugal

Rongzhi Dong Department of Computer Science and Engineering, University of South Carolina, Columbia, SC, USA

Sam Earle Computer Science and Engineering, New York University, Brooklyn, NY, USA

Yuqi Feng College of Computer Science, Sichuan University, Chengdu, China

Alex Freitas School of Computing, University of Kent, Canterbury, UK

Mario Garza-Fabre Cinvestav, Campus Tamaulipas, Victoria, Tamaulipas, Mexico

Siddharth Gianey Department of Computer Science and Engineering, University of South Carolina, Columbia, SC, USA

Jonathan Gordon OpenAI Inc., San Francisco, CA, USA

Michael Cerny Green Computer Science and Engineering, New York University, Brooklyn, NY, USA

Alexander Hadjiivanov European Space Agency ESA, Noordwijk, Netherlands

Julia Handl University of Manchester, Manchester, UK

Erik Hemberg MIT CSAIL, Cambridge, MA, USA

Malcolm I. Heywood Faculty of Computer Science, Dalhousie University, Halifax, NS, Canada

Jianjun Hu Department of Computer Science and Engineering, University of South Carolina, Columbia, SC, USA

Ting Hu School of Computing, Queen's University, Kingston, ON, Canada

Shawn Jain OpenAI Inc., San Francisco, CA, USA

Adán José-García Univ. Lille, CNRS, Centrale Lille, Lille, France

Stephen Kelly Department of Computing and Software, McMaster University, Hamilton, Ontario, Canada

Ahmed Khalifa Institute of Digital Games, University of Malta, Msida, Malta

Joel Lehman OpenAI Inc., San Francisco, CA, USA

Michael A. Lones Department of Computer Science, Heriot-Watt University, Edinburgh, UK

Nuno Lourenço University of Coimbra, CISUC, DEI, Coimbra, Portugal

Zeqiong Lv College of Computer Science, Sichuan University, Chengdu, China

Penousal Machado Department of Informatics Engineering, Centre for Informatics and Systems of the University of Coimbra, University of Coimbra, Coimbra, Portugal

Tiago Martins Department of Informatics Engineering, Centre for Informatics and Systems of the University of Coimbra, University of Coimbra, Coimbra, Portugal

Nicholas Matsumoto Cedars-Sinai Medical Center, Los Angeles, CA, USA

Jon McCormack Monash University, Melbourne, Australia

Eric Medvet University of Trieste, Trieste, Italy

Risto Miikkulainen Department of Computer Science, University of Texas at Austin, Austin, TX, USA;
Cognizant AI Labs, San Francisco, CA, USA

Jason H. Moore Cedars-Sinai Medical Center, Los Angeles, CA, USA

Vojtech Mrazek Faculty of Information Technology, Brno University of Technology, Brno, Czech Republic

Giorgia Nadizar University of Trieste, Trieste, Italy

Kamal Ndousse Anthropic Inc., San Francisco, CA, USA

Bach Nguyen Centre of Data Science and Artificial Intelligence & School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand

Bernd R. Noack School of Mechanical Engineering and Automation, Harbin Institute of Technology (Shenzhen), Shenzhen, PR China

Michael O'Neill Natural Computing Research & Applications Group, School of Business, University College Dublin, Dublin, Ireland

Una-May O'Reilly MIT CSAIL, Cambridge, MA, USA

Sadman Sadeed Omee Department of Computer Science and Engineering, University of South Carolina, Columbia, SC, USA

Yuwei Ou College of Computer Science, Sichuan University, Chengdu, China

Federico Pigozzi University of Trieste, Trieste, Italy

Michal Pinos Faculty of Information Technology, Brno University of Technology, Brno, Czech Republic

Pedro H. Ribeiro Cedars-Sinai Medical Center, Los Angeles, CA, USA

Anil K. Saini Cedars-Sinai Medical Center, Los Angeles, CA, USA

Erica Salvato University of Trieste, Trieste, Italy

Roberto Santana University of the Basque Country (UPV/EHU), San Sebastian, Gipuzkoa, Spain

Jory Schossau Department of Computer Science & Engineering, Michigan State University, East Lansing, MI, USA

Lukas Sekanina Faculty of Information Technology, Brno University of Technology, Brno, Czech Republic

Sara Silva LASIGE, Department of Informatics, Faculty of Sciences, University of Lisbon, Lisboa, Portugal

Stephen L. Smith Department of Electronic Engineering, University of York, York, UK

Xiaotian Song College of Computer Science, Sichuan University, Chengdu, China

Yuqi Song Department of Computer Science and Engineering, University of South Carolina, Columbia, SC, USA

Lisa Soros Computer Science, Barnard College, New York, USA

Kenneth O. Stanley OpenAI Inc., San Francisco, CA, USA

Yanan Sun College of Computer Science, Sichuan University, Chengdu, China

Julian Togelius Computer Science and Engineering, New York University, Brooklyn, NY, USA;

Odl.ai, Copenhagen, Denmark

Jamal Toutouh ITIS Software, University of Malaga, Malaga, Spain

Inês Valentim University of Coimbra, CISUC, DEI, Coimbra, Portugal

Moritz von Looz European Space Agency ESA, Noordwijk, Netherlands

Lai Wei Department of Computer Science and Engineering, University of South Carolina, Columbia, SC, USA

Bing Xue Evolutionary Computation Research Group at the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand; Centre of Data Science and Artificial Intelligence & School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand

Cathy Yeh OpenAI Inc., San Francisco, CA, USA

Mengjie Zhang Evolutionary Computation Research Group at the School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand;

Centre of Data Science and Artificial Intelligence & School of Engineering and Computer Science, Victoria University of Wellington, Wellington, New Zealand

Ryan Zhou School of Computing, Queen's University, Kingston, ON, Canada

Part I

Evolutionary Machine Learning Basics

In which we introduce some of the basics of Evolutionary Machine Learning: Its foundation, and the three principal learning methods—supervised learning, unsupervised learning, and reinforcement learning.

Chapter 1

Fundamentals of Evolutionary Machine Learning



Wolfgang Banzhaf and Penousal Machado

Abstract In this opening chapter, we overview the quickly developing field of evolutionary machine learning. We first motivate the field and define how we understand evolutionary machine learning. Then we take a look at its roots, finding that it has quite a long history, going back to the 1950s. We introduce a taxonomy of the field, discuss the major branches of evolutionary machine learning, and conclude by outlining open problems.

1.1 Introduction

Machine Learning is the branch of Artificial Intelligence that concerns itself with building models of systems and making predictions based on extracted data from data sources or system observations. The term Machine Learning was coined in 1959 by Arthur Samuel of IBM [87], the first person to make a computer perform a serious learning task, a checkers playing program [7]. Since the late 1940s Samuel had worked on games as study objects for computer programming and intelligence [106]. In his seminal 1959 contribution that started the field of Machine Learning with a bang, Samuel writes:

Two machine-learning procedures have been investigated in some detail using the game of checkers. Enough work has been done to verify the fact that a computer can be programmed so that it will learn to play a better game of checkers than can be played by the person who wrote the program. Furthermore, it can learn to do this in a remarkably short period of time (8 or 10 hours of machine-playing time) when given only the rules of the game, a sense

W. Banzhaf
Department of Computer Science and Engineering, Michigan State University,
East Lansing,
MI 48864, USA
e-mail: banzhafw@msu.edu

P. Machado (✉)
Department of Informatics Engineering, Centre for Informatics and Systems of the University
of Coimbra, University of Coimbra, 3004-531 Coimbra, Portugal
e-mail: machado@dei.uc.pt

of direction, and a redundant and incomplete list of parameters which are thought to have something to do with the game, but whose correct signs and relative weights are unknown and unspecified. The principles of machine learning verified by these experiments are, of course, applicable to many other situations [87].

Many decades later, Machine Learning (ML) has become a mainstream topic in Computer Science research, taking up more and more space also in other Scientific and Engineering disciplines, with even the Arts and Humanities becoming involved, as will be amply demonstrated in this book. The success of ML relative to the lingering progress of other areas of Artificial Intelligence can be attributed to mainly four factors:

1. The explosive development of hardware
2. The availability of data
3. The grounding in Mathematical/Statistical thought
4. The ability to find solvable ML problems

First, hardware has continued to progress on an exponential curve for many decades. While Moore's law might cease very soon to be valid in its current form [5], the explosive development of hardware capabilities—processing speed, data storage capacity—has made ML possible at scale. Second, data is everywhere around us, to the point where in some applications it is no longer possible to even store the data, and one has to resort to stream processing. So the raw material for machine learning—large amounts of data—is copiously available thanks to physical sensory devices that transform the dynamics of our world into electric signals. Third, ML has a strong grounding in existing formal methods, Mathematics and Statistics chief among them. The availability of many statistical techniques thus can turn large amounts of data into patterns and other types of useful information. This is not without caveats and dangers, as data bias or assumptions about statistical distributions exemplify, but it has allowed decent progress on learning language, image classification, text processing etc. Finally, it was critical to restrict targets for ML to solvable problems. As technology progressed, the front of solvable problems was continuously expanding, from early game learning, to the most difficult games like Chess and Go today, just to name the class of board games. In many other categories of problems, a similar trajectory was followed by ML researchers.

Evolutionary Computation (EC), much like machine learning, has an extensive history. In 1957, Fraser [34] simulated a genetic system using a computer, which is likely to be the first evolutionary algorithm. Friedberg's work in 1958 [35] is acknowledged as the first application of an evolutionary approach to program solving, specifically automated programming. By the 1960s, most of the foundations of what we now know as EC were already laid. Fogel et al. introduced Evolutionary Programming as an alternative to classical AI between 1962 and 1966 [30, 31]. Concurrently, Holland presented Genetic Algorithms in 1962 [45], while Rechenberg et al. developed Evolution Strategies in 1965 [83]. The fourth cornerstone for EC was only laid in 1989, when Koza invented Genetic Programming in 1989 [54].

These fields of research remained independent until the early 1990s, when the efforts to promote collaboration between these different tribes paid off, resulting

in the establishment of scientific events that welcomed contributions from all these streams. Created in 1991, ‘Parallel Problem Solving from Nature’ (PPSN) is the most notable example of this trend. By 1993 the term ‘Evolutionary Computation’ had gained recognition, and a journal with the same name was created. The merge of conferences dedicated to independent tribes gave rise to ‘Genetic and Evolutionary Computation Conference’ (GECCO) and ‘IEEE Congress on Evolutionary Computation’ (IEEE CEC) that encompass any flavor of EC and many other EC conferences have arisen.

The main contributions that Evolutionary Computation can bring to ML, thus leading to the field of *Evolutionary Machine Learning* are, first, its ability to create novel and, therefore, unforeseen solutions to ML problems. The element of surprise is enabled by the stochastic nature of these algorithms, which allows solutions to emerge from noisy beginnings, under the guidance of selection. Second, the inherent stochasticity in producing solutions can generate more robust solutions, solutions that can persist in the presence of noise. While again stochasticity is a key contributor to this feature, the fact that normally populations of solutions are considered in EC contributes substantially to this strength. Third, the unforeseen character of many solutions can be used to discover logical flaws in systems and tools designed by humans or other algorithms. It is well known that evolution exploits all kinds of shortcuts if it is able to discover them. That frequently leads to surprising results, uncovering untested or unforeseen loopholes in the human designs [58]. Finally, because evolution can be used very well to optimize processes, it brings to the table an ability to streamline systems. By not only targeting the accuracy of a solution, but also simplicity, speed of achievement, or other efficiency measures in a multi-objective setting, EML can serve multiple purposes at the same time [17, 61].

The promise of Evolutionary Machine Learning is thus that it can lead to better solutions faster, while exhibiting a surprising ability to come up with creative and robust features of these solutions. These are characteristics that every machine learning algorithm should strive to possess.

We now want to step back for a moment and consider how to define EML.

1.2 A Definition of Evolutionary Machine Learning

A definition of Evolutionary Machine Learning (EML) has to look at all three ingredients to the term—evolution, machine and learning. From a distant point of view, EML is a method of adaptation, where we understand ‘adaptation’ as *the continuous improvement of a system’s performance over time and with experience*. By improved performance we mean that the system is able to react to its environment in a more appropriate way over time. There are other types of adaptation, like the adaptation of organisms to environmental trends (‘epigenetic adaptation’), or the adaptation of defense mechanisms of an organism in its immune system to outside attacks. But here we want to focus on evolution and learning as the main components of adaptation under consideration.

What, then, do we understand under evolutionary adaptation? Evolutionary adaptation is adaptation governed by the evolutionary principles of inheritance and cumulative selection of beneficial variations in individuals. Thus, it is fundamentally a process that takes place over generations of individuals, i.e., on a timescale whose smallest unit is the expected lifetime of an individual. This should be seen in contrast to the adaptive processes in learning [71], where the timescale of the adaptive process is much shorter, as learning has to take place *within* the lifetime of an individual. Learning is an extension of adaptive processes effective in organisms that are provided with the necessary pre-requisite ‘hardware’ which comes about by evolution and development.

What is a machine? There are various definitions and some scholars count even organisms as machines [96]. Here we want to define a machine as *an artificial ('man-made') system that can perceive its environment in some way and can act on it*. The emphasis is on the fact that a machine can act, which requires the ability to adjust its complexity (both for behavior and for the processing of input information). It is a system, i.e., it can only work in its entirety. While it has parts that might or might not be necessary, the assembly of the parts in itself does not constitute the machine. It requires, in addition, some dynamics that we can call its behavior and which comes about only if the parts interact appropriately. In some sense, it is a weak emergent process [9] that leads to the function/behavior of the machine.

And what do we want to understand under the term ‘learning’? Learning is *the adaptation that an organism exhibits over its lifetime*. Transferred into the realm of artifacts, it is the adaptation of the behavior of an agent over its existence. Learning requires a number of functionalities that can be listed as follows:

1. Input (data from sensors or other devices)
2. Output (actions/behaviors exerted on the environment or itself)
3. Feedback from the environment (external)
4. Error/value measurements (internal)
5. Flexibility to change the interaction of its parts
6. An improvement (or even optimization) algorithm that enables adaptation

Learning is one of the most fascinating abilities of living organisms and has for a long time been an inspiration for engineers and tinkerers. It is also the fastest and most efficient way organisms can adapt to their environment.

If we now put the definition together, we can say: *The field of Evolutionary Machine Learning concerns itself with the application of evolution to Machine Learning problems and methods and the application of machine learning to evolutionary computation (problems and) methods.*

1.3 A History of EML

In this section we lay out a historical timeline of some of the contributions that played an important role in the emergence and development of EML. These events are presented chronologically in Figs. 1.1, 1.2 and 1.3.

1.3.1 EC Applied to ML Methods

We shall start with the development of evolutionary computation principles being applied to ML methods. This history really begins with Turing's essay on computers and intelligence [101] (see Fig. 1.1). Turing argues that a key part of intelligence is the ability to make errors. In fact, in a presentation at a radio program Turing [102] had clear words about this:

My contention is that machines can be constructed which will simulate the behavior of the human mind very closely. They will make mistakes at times, and at times they may make new and very interesting statements, and on the whole, the output of them will be worth attention to the same sort of extent as the output of a human mind.

He also introduces the idea of random elements into consideration when discussing intelligent machines, saying:

There is, however, one feature that I would like to suggest should be incorporated in the machines, and that is a 'random element'. Each machine should be supplied with a tape bearing a random series of figures, e.g., 0 and 1 in equal quantities, and this series of figures should be used in the choices made by the machine. This would result in the behavior of the machine not being by any means completely determined by the experiences to which it was subjected, and would have some valuable uses when one was experimenting with it.

This latter point, that a stochastic element is necessary to provide some type of creative solutions has therefore been identified at the beginning of AI by one of the pioneers of computing as being an important element.

Friedberg [35] in 1958 then uses random variations in his algorithms to produce a variety of responses. Tragically, Friedberg missed the second important ingredient when using randomness selection. So when Samuel worked on his Checkers program which is the first demonstration of the power of ML [87], he was emphasizing this aspect of selection more than the aspect of randomness (he provided highly structured systems to make progress toward playing Checkers).

But it was the combination of randomness and selection, variation and cumulative selection, that made evolutionary principles available for ML. Fogel and co-workers proposed as material for evolution finite state machines, automata that at the time were the predominant way to formalize computing methods. In their seminal contribution [30, 31] they introduce a system that can be used to discern and react to time series of states.

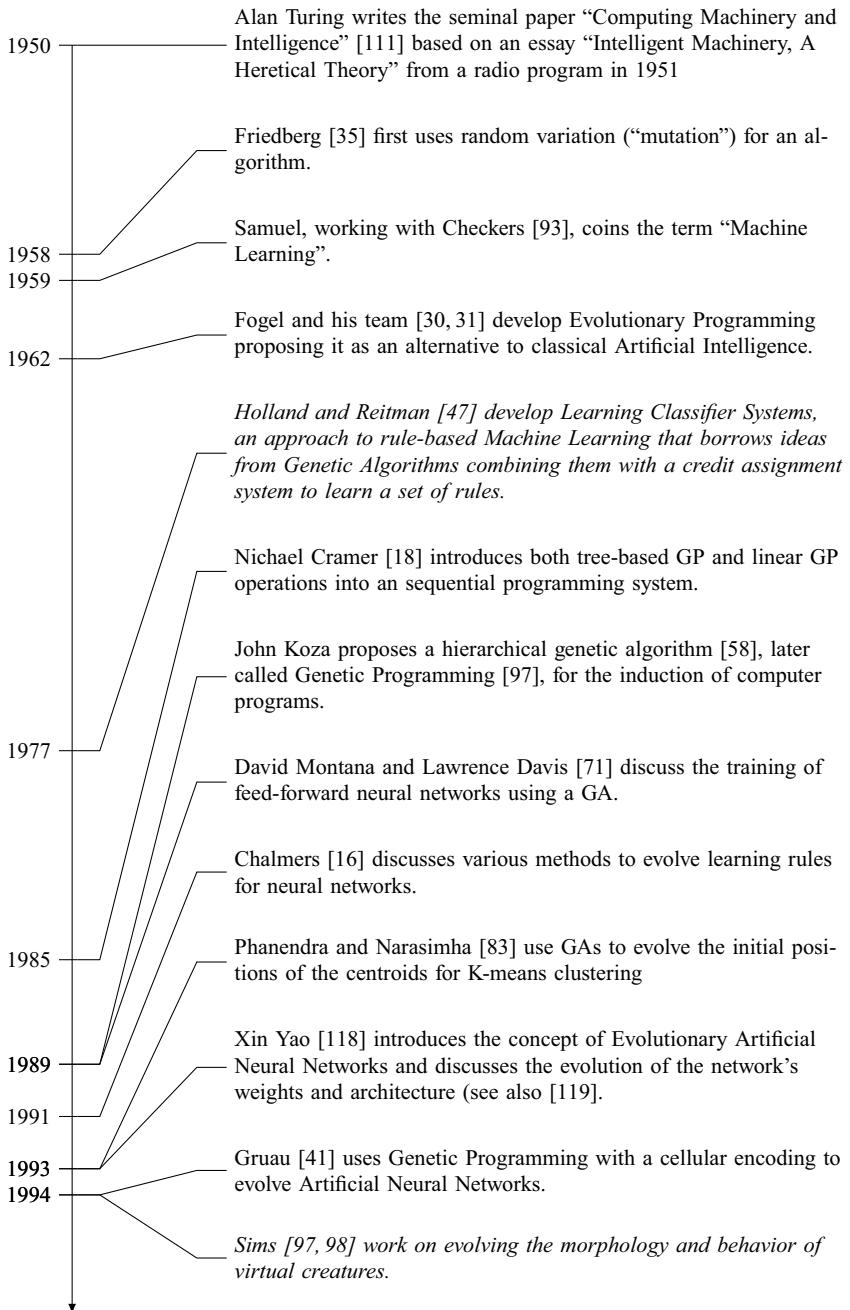


Fig. 1.1 Timeline of the application of EC techniques to ML methods—Part 1

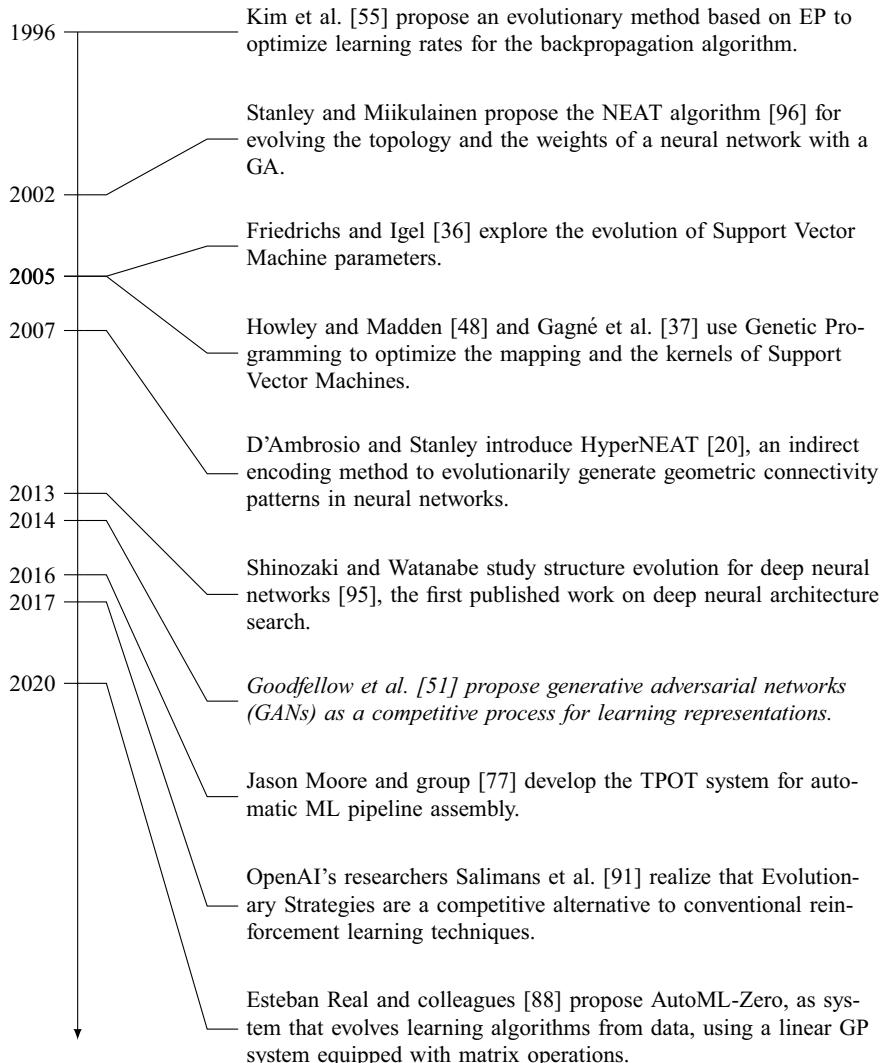


Fig. 1.2 Timeline of the application of EC techniques to ML methods—Part 2

While Holland more or less at the same time conceptualized the genetic algorithm [46], a key contribution of Holland to machine learning was rule-based systems under evolution [47], what later would become ‘learning classifier systems’. Rules here were composed of condition/action pairs, with the condition having to be fulfilled for the action to be taken.

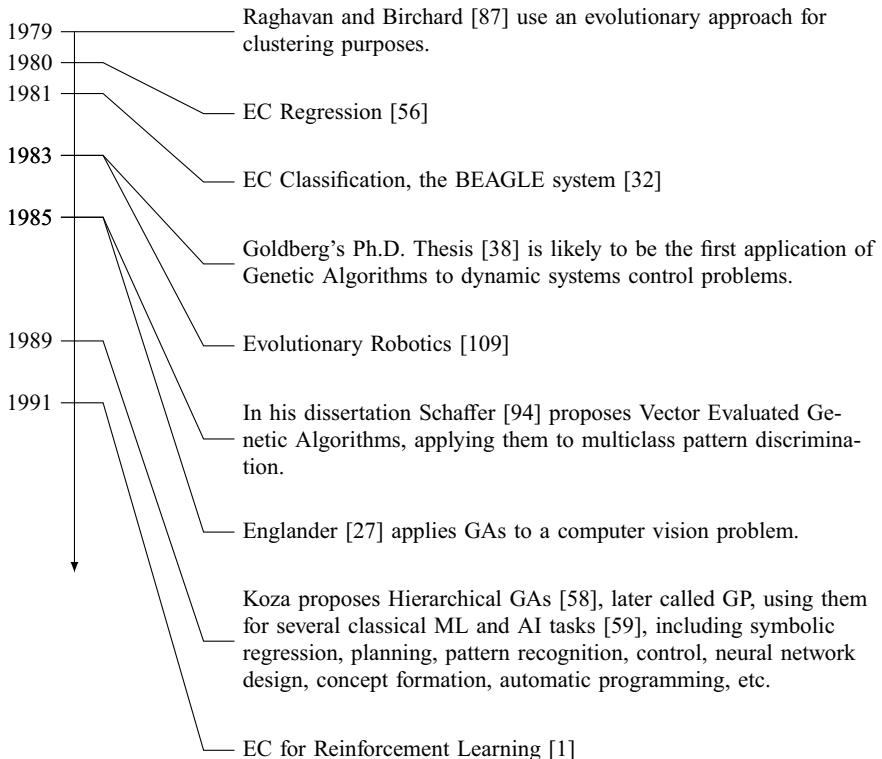


Fig. 1.3 Timeline of the early applications of EC techniques to classical ML problems

Then, from the mid-1980s to the early 1990s, Genetic Programming was invented and established. First, it appeared rather inconspicuously at the ICGA conference [18], already in two different representations. But the importance of these proposals was first not recognized, and only became established with John Koza's work later in the 1980s [54] and early 1990s [55, 56], in particular with Koza's seminal book on Genetic Programming.

During the same time, neural networks were examined more closely in connection with genetic approaches. Montana and Davis [69] and Chalmers [16] are concerned with the training of neural networks, either the error reduction process or the invention of learning rules. Phanendra et al. [77] use GAs to evolve centroid positions in a K-means clustering algorithm. At around the same time, the concept of evolutionary neural networks, or neuroevolution is introduced by Yao [108].

Further ideas were developed around the same time, like Gruau's method to grow neural network by a kind of developmental process [41] or Sims' seminal work on hybridizing evolution with neural networks for behavioral control of virtual creatures [91, 92].

Further developments are listed in Fig. 1.2.

1.3.2 EC Applied to ML Problems

Besides the merging of methods from EC and ML, EC has also been applied to typical ML problems, which could be termed another branch of EML (see Fig. 1.3).

Clustering was first as a task where an evolutionary approach was considered [81]. This happened in 1979, at a time when genetic algorithms were not yet widely known, though Holland’s book was already published a few years earlier [46]. Then, in a quick succession of developments, evolutionary computation methods were tried out on classical tasks like classification [32] and regression [52].

Goldberg’s thesis is likely the first to apply GAs to dynamic systems control problems [38]. Evolutionary robotics in the form of first EC algorithms applied to adaptation tasks for robots was introduced in 1985 [99]. Another pioneer in GAs, Schaffer, proposes more complex data structures to address multiclass pattern discrimination [88].

Broader areas of application of EC were examined by Englander [27] who examined computer vision tasks, and Koza [55, 56] who showed that several classical tasks can be solved by Genetic Programming. It was thus emphasized and explained that GP is a technique [7, 56].

Then in the early 1990s, reinforcement learning was hybridized with evolution [1]. From the 1990s onward, one after the other of typical ML tasks was subjected to EC examination, often showing surprising performance.

1.4 A Taxonomy of EML

In this section, we introduce a taxonomy for EML based on our analysis and understanding of the field. More than aiming at classifying algorithms and approaches in rigid categories, we aim at providing a structured perspective that guides newcomers and experienced practitioners, and should contribute to the discovery of related work and the identification of new research opportunities. Furthermore, we recognize that alternative classification schemes and gray areas may exist (see, e.g., [2, 98]). We divide our classification into three main categories:

1. The use of EC to enhance, support, expand, or amplify ML methods;
2. The reciprocal, employing ML techniques to enhance, support, expand, or amplify EC methods;
3. The application of EC to problems typically considered ML problems and traditionally solved by other ML approaches;

These categories are addressed, respectively, in the following subsections.

1.4.1 EC for ML Methods

In this section we focus on the contributions of EC in the context of ML approaches. These are summarized in Fig. 1.4.

At the initial level of analysis, we examine the stage of the ML pipeline where EC plays a role: (i) Data or Input for the ML algorithm, (ii) the ML algorithm itself, (iii) and the resulting outcome.

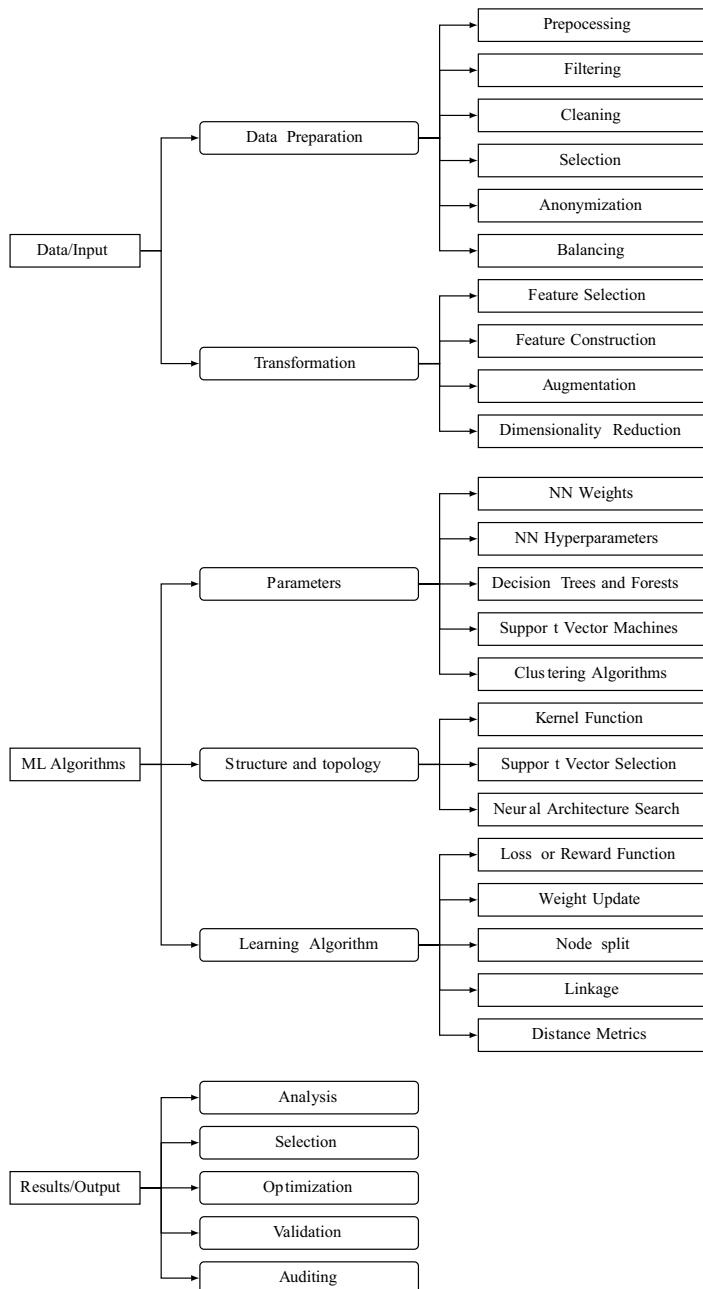
Specifically focusing on the ‘Data/Input’ stage, we further categorize it into ‘Data Preparation’ and ‘Data Transformation’, acknowledging that the boundary between these subcategories may sometimes be blurry. Data preparation tasks typically involve filtering, cleaning, selection, anonymization, balancing, and other forms of preprocessing such as missing data imputation. On the other hand, data transformation involves processes like feature selection, feature construction (including feature extraction), data augmentation, and dimensionality reduction.

The application of EC to ML algorithms offers numerous and diverse opportunities. Here we identify three key levels of application: ‘parameter,’ ‘topology,’ and ‘learning algorithm.’ At the ‘parameter’ level, the goal is to optimize the parameters or coefficients of a given model with a predetermined structure/topology. At the ‘topological’ level, no predefined model structure exists. The objective is to automatically learn an adequate model structure/topology for a given problem. Lastly, at the ‘learning algorithm’ level, the task involves either selecting or developing an algorithm capable of learning and optimizing the model’s topology, parameters, or a combination of both.

Examples of hybridization at the ‘parameter’ level encompass the evolutionary optimization of neural network weights and hyperparameters. The ‘topological’ level includes the area now known as Neural Architecture Search (NAS). Lastly, at the ‘learning algorithm’ level, we can consider applications such as the evolution of the weight update function, loss function, and reward function. When focusing solely on neural networks, the techniques discussed at the ‘parameter’ and ‘topological’ levels are collectively known as neuroevolution.

While our examples lean toward connectionist approaches due to their current popularity and relevance of neuroevolution, it is important to note that EC has also been applied successfully to other ML algorithms. For instance, at the ‘parameter’ level, parameterization can be applied to tree classifiers (e.g., number of levels, node split criteria), clustering algorithms (e.g., centroid initialization, cluster number selection), and the optimization of hyperparameters in Support Vector Machines (e.g., choice of kernel type, kernel parameters, and regularization parameter).

At the ‘topological’ level we can mention the evolution of kernel functions and support vector selection, as they impact the structure of the space. However, we acknowledge that these can also be seen as interventions at the ‘algorithm’ level. Examples of EC applications at the ‘learning algorithm’ level include the evolution of linkage criteria, distance metrics, and node split functions.

**Fig. 1.4** EC for ML methods

The application of EC to the ‘Results/Output’ of ML algorithms is less common than its application to the ‘data/input’ or ‘ML algorithm’. Nonetheless, we identified several opportunities and examples of application, including:

- Using EC to analyze the results of ML methods, such as evaluating their robustness by generating adversarial examples that aim to exploit vulnerabilities in ML models, helping auditors to identify potential weaknesses, understand the limitations, and assess its susceptibility to adversarial attacks.
- Use EC to select among several results of ML algorithms, e.g., optimize the creation of ensembles by evolving diverse combinations of base ML models;
- Optimizing or fine-tuning the results of ML by EC methods;
- Validating ML solutions by generating novel test instances;
- Creating EC gray-box or white-box surrogate models for the behavior of black-box ML models;
- Contributing to ‘auditing’ ML applications for trustworthy AI, e.g., evaluating the fairness of ML models by evolving instances that represent different demographic groups or protected attributes, contributing to the identification of potential biases or disparities in how the model treats different groups.

1.4.2 ML for EC Methods

While in the previous subsection, we focused on the application of EC in non-evolutionary ML pipelines, our focus now shifts to the use of ML algorithms within EC approaches.

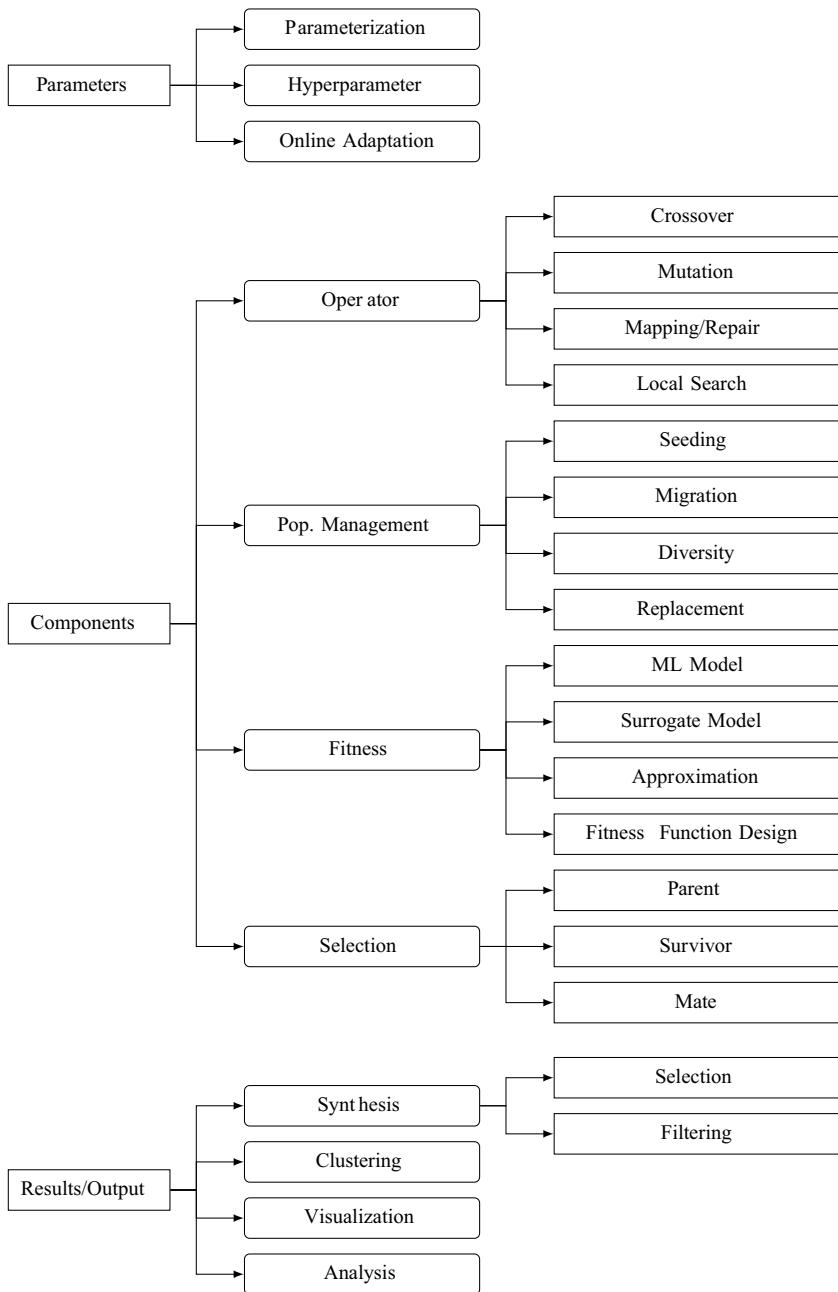
We identify three primary levels of ML application in the context of EC: parameterization of the EC method, selection or replacement of specific EC algorithm components, and processing of the results. Fig. 1.5 provides an overview of this classification.

When it comes to the parameterization of EC, ML can be employed to determine the values for various parameters and hyperparameters. This may include selecting appropriate crossover and mutation rates, determining the population and tournament sizes, making choices regarding operator selection, population structure, or even EC algorithm selection (e.g., deciding based on the characteristics of the problem which EC variant to use). In most cases, the parameterization is performed offline before the beginning of the runs. However, the online adaptation of EC parameters and hyperparameters by ML means is a promising opportunity for hybridization.

The use of ML techniques to replace or adapt specific components of an EC approach is promising. We identify four key types of components: ‘Operators,’ ‘Population Management,’ ‘Fitness’ and ‘Selection.’

For the ‘Operators’ component, there are several opportunities where ML can be applied, including:

- Learning to perform crossover and mutation using techniques such as reinforcement learning;

**Fig. 1.5** ML for EC methods

- Developing repair operators that can correct invalid phenotypes generated during the evolutionary process;
- Learning genotype to phenotype mapping schemes, which can promote evolvability;
- Employing ML algorithms as local search operators, for instance using Adam as a local search operator when exploring a latent space through evolutionary means;

Population management also offers several opportunities for ML use, including:

- Using ML solutions as seeds or selecting previously evolved solutions to initialize new runs, improving the starting point of the evolutionary run.
- Controlling migration of individuals in multi-population EC models, determining when and how individuals are transferred between populations.
- Regulating population diversity, promoting a balance between exploration and exploitation.
- Determining the replacement strategy, optimizing the selection and replacement of individuals.

The most common application of ML in EC is fitness assignment. A typical scenario is using ML to learn an objective function that is difficult to formally express, such as subjective or esthetic criteria. The ML model is then used to assign fitness. Another scenario arises when the exact objective function exists but is too computationally expensive, prompting the use of ML methods to approximate its value for fitness assignment. When the ML models are also too costly, surrogate models can be employed as approximations.

The automated design of fitness functions is an area that holds great promise but remains relatively unexplored. Recent advancements in Large Language Models offer exciting possibilities. For instance, one notable application is using CLIP [80] to calculate the cosine similarity between a natural language prompt and an evolved image, and using this similarity as a fitness measure.

It is important to acknowledge that using an ML model for fitness assignment has its limitations. Most notably, EC algorithms tend to exploit shortcomings of an ML model, maximizing fitness without necessarily improving the desired objective function. This issue has been consistently reported since the early days of EML and persists in the Deep Learning era [6, 70, 94].

Furthermore, ML can be applied to parent, mate, or survivor selection, considering factors such as population quality, diversity, compatibility among mates, and proximity to the optimum.

Lastly, let's consider the application of ML methods to the outcomes of EC methods. One obvious application is using ML to synthesize the outcomes of evolutionary runs by filtering or selecting results. Additionally, clustering the outputs of one or multiple evolutionary runs can be valuable, especially in scenarios with multiple alternative solutions, such as multi-objective and many-objective optimization. ML-powered visualization techniques, such as t-SNE [103], can also be powerful tools for presenting results to end-users and analyzing the evolutionary process. Furthermore, ML techniques, such as Large Language Models, can significantly expedite the

analysis of EC processes, offering an important tool for researchers and practitioners in this field.

While there are several noteworthy works, we believe that the use of ML for EC is just beginning to blossom. Many of the areas we have identified remain largely uncharted, and hold potential for exploration. Moreover, even the territories that were explored in the early days of EML deserve to be revisited with the tools and knowledge of modern ML. By delving into these untapped areas and building upon existing knowledge, we can uncover new insights and advancements in the field.

1.4.3 EC for ML Problems

In our perspective EC can be seen as a form of ML and, as such, several of the problems that are typically addressed by non-evolutionary ML can also be tackled by EC. Fig. 1.6 summarizes the main categories and problem types that we identify in this context.

We consider four main categories, and corresponding typical problems: ‘supervised,’ ‘unsupervised,’ ‘reinforcement,’ and ‘meta-learning.’

In the ‘supervised’ learning category, EC has been successfully applied to address classic problems such as Classification, Regression, and Generation. Clustering is a prototypical ‘unsupervised’ learning problem solved by evolutionary means.

The domain of reinforcement learning is particularly well-suited for EC, since it is natural to see fitness as a consequence of the interaction between the individuals and their environment, resulting in a reward or penalization. While reinforcement learning scenarios can be tackled using canonical EC methods, they lend naturally to the application of co-evolutionary approaches, whether they are competitive, cooperative, or a combination of both.

It is worth noting that competitive co-evolution predates what is now known as adversarial AI, including Generative Adversarial Neural Networks (GANs). Furthermore, the competition between evolutionary and non-evolutionary ML approaches, including the co-adaptation of both types of models as a result of this competition, has a long history (see, e.g., [66] and [65] for early and recent examples, respectively).

Meta-learning is a significant challenge for ML and hence for EML. In this area we identify five subcategories: ‘ensemble’ learning, ‘federated’ learning, ‘life-long’ learning, ‘online’ learning, and ‘transfer’ learning. When it comes to ensemble and federated learning, collective intelligence models are an intuitive metaphor.

Life-long and online learning implies the continuous adaptation of individuals. While this process is not achieved through evolutionary means in nature, computationally we have the freedom to explore EC approaches for these purposes. Nevertheless, if one wishes to follow the metaphor closely, evolving ML models and algorithms capable of life-long and online learning would be a natural approach. Likewise, in the case of transfer learning, we can promote the evolution of ML models that exhibit knowledge transfer abilities. This can be accomplished by directly assessing this capability in the fitness function or by valuing characteristics that may

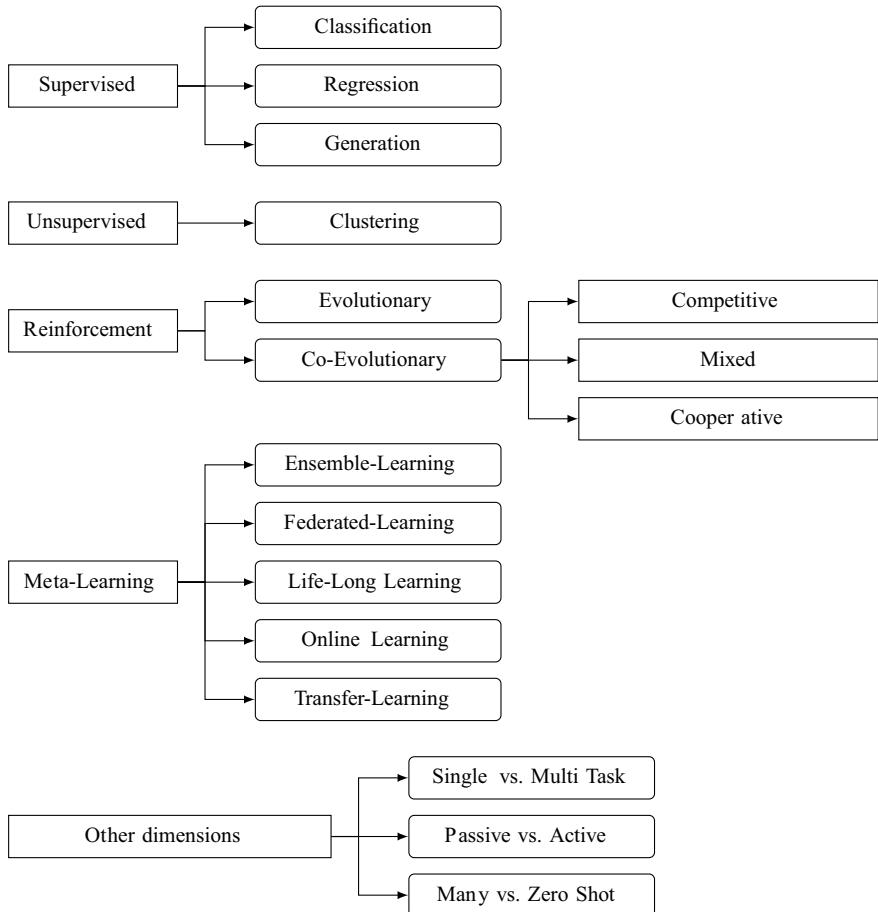


Fig. 1.6 EC for ML Problems

facilitate transfer learning, such as modularity, re-utilization of knowledge or pathways, incremental learning abilities, compactness or even physical plausibility of the model.

Finally, we identify additional dimensions of analysis that are, to some extent, orthogonal to the previously identified categories. These dimensions include:

- **Single versus Many task**—while in single task learning we focus on a specific task on many task learning the models should be able to perform multiple tasks, typically by using shared knowledge and patterns across tasks.
- **Passive versus Active learning**—Passive learning assumes that all data is pre-labelled, whereas in active learning the model prompts the user interactively for labeling data. This type of interaction bears resemblance to certain forms of inter-

- active EC, particularly semi-interactive ones, and can be a valuable opportunity for EC research.
- Many versus Zero Shot learning—while in many shot learning we have several labeled training examples by class in zero-shot learning the challenge is to classify objects without any labeled training instances. This is typically attained by associating observed and non-observed classes through auxiliary information, e.g., a model algorithm trained to recognize horses may recognize zebras by using the auxiliary information that ‘zebras look like striped horses.’

1.5 Discussion of Main Branches of the Field

We shall start our discussion with a brief overview of the main branches before delving into more details of each one of them. Many evolutionary and, more generally, selectionist approaches have been studied in the context of brain function [29].

Due to the recent success of deep neural networks, a natural topic for EML is the question of how to improve neural networks with evolution. This field of EML is called *neuroevolution*. It addresses the question on all levels of the neural network from the synaptic level, through the neuronal level, to the connectivity patterns and higher level structures like neural architectures and even developmental aspects of the nervous system.

A second branch concerns itself with the algorithms for learning. There are various aspects of this, including the evolution of learning algorithms (meta-learning), efficiency aspects of training and learning, considerations of ensembles and competition and cooperation in learning as well as the evolutionary integration of tools into the ML pipeline.

A third branch of EML studies the results of ML algorithms. This refers to whether the results are fair, trustworthy, explainable, interpretable, usable, accessible etc. Many of these aspects require more analysis for measurement and optimization, with some still being under basic research for quantification.

The fourth main branch of EML considers ethical aspects, green EML, rules for using the systems etc. This is a less well-defined branch, that will, however, only gain more visibility over the next couple of years.

1.5.1 Neuroevolution

It is well known from Biology that synapses are complex systems that must have undergone substantial evolution to reach the current status [25]. Therefore, it might be the case that the connections in artificial neural networks should also be subjected to evolution to yield more complex and better organized connectivity in the networks [110].

Moving to the level of the neuron, it has been found that neurons themselves are highly complex and individual entities that require for their simulation entire deep artificial neural networks [11]. It should, therefore, not astonish when multiple aspects of neurons are subjected to various optimization approaches from the domain of evolutionary algorithms. For an example, see [105].

Moving up a further level, to network/architecture level, an entire new field called Neural Architecture Search (NAS) has emerged which benefits from evolutionary approaches [59]. This entails the design of networks with sparse layers, connections that skip over layers or recurrent connections between layers. Multi-objective algorithms have been recruited to pressure evolution toward simple solutions [62].

Finally, developmental aspects of neural networks, like growing neural network structures from small structures, have become an important issue. In Biology, the evo-devo movement has taken firm roots in research examining the brain and its architecture [42]. Taking inspiration from developmental neuroscience, EML addresses the building of artificial neural structures [68].

1.5.2 Learning Algorithms

The evolution of learning algorithms from scratch has recently shown major success for EML. Real et al. have been able to demonstrate the evolution of the backpropagation learning algorithm from data and simple mathematical building blocks like matrices and vectors [82]. Clustering algorithms have for a long time been considered material for evolution [2]. The newest in meta-learning is the evolution of transformers [93].

While much of this work features under the term ‘meta-learning,’ other aspects of learning algorithms, like hyperparameters or the amount and sequential organization of training can also be subject to evolution [28].

A whole different area is constituted by ensemble learning methods [22, 84], where not just one, but a number of models are used to make progress on a learning task. Evolutionary ensemble learning is discussed in one of the chapters here.

Apart from ensembles, which constitute part of cooperative algorithms, competitive or adversarial algorithms are also used to generate different solutions [49]. This whole area in EML is known under the topic co-evolutionary algorithms [63], and contains, in addition to cooperative also competitive systems, like adversarial algorithms that make use of two different populations [39]. Game-like algorithms where players are modeled [112] are also relevant for evolutionary studies.

Finally, we need to mention the evolutionary design of pipelines which might consist of many different learning methods at different stages of the ML process. TPOT [72] is a famous example of this line of work which claims its rightly place in the field of automatic machine learning.

1.5.3 Learning Results

One key aspect of learning systems is whether the results of such learning can be interpreted by humans or explained and communicated to others. Interpretable [23] and explainable [10] models will be required in many fields of application, like business and government, and neural network solutions have some problems with those required characteristics. Evolution can pressure solutions toward such criteria, if they are quantifiable. Also, genetic programming, as one of the major EML methods can be based on symbolic structures and is naturally geared toward explainability [67].

Further requirements for learning results include trustworthiness, user friendliness for AI novices, usability and accessibility. All of these, as long as they can reasonably be defined, can be part of a multi-objective EML approach.

1.5.4 Ethical Aspects

We end this discussion by mentioning a number of aspects of EML that are somewhat less clearly defined, yet merit continued research and investigation. Much of the work currently going on in the field of ML is based on case studies [60, 73]. However, fairness and bias have been refined gradually to the point where they can be considered measurable quantities [76].

But other aspects of ethical behavior, not subsumed under the previous topics, are ideas about green ML [43, 100]. In this line of inquiry, how to learn as quickly as possible, with as few CPU/GPU cycles as possible, is a quantifiable goal open to EML (see, e.g., [78]).

This also includes the systematic measurement and reporting on energy requirements of ML generated models [44, 57] which can help to design efficient hardware and software approaches [3, 97]. While not yet systematically deployed, multi-objective EML approaches hold much promise for this field.

1.6 Open Problems

Evolutionary Machine Learning is a young field, despite its roots in the 1980s. This is mostly due to the technical progress that allowed powerful neural networks to solve difficult problems. The research questions in EML can be roughly categorized into two classes: Methodological open problems and practical open problems. Below we provide a list of where we see the major open problems in each of these categories. This is not a complete list, but it is our attempt to take stock of the current status and outline the problems we currently see in this field.

We see the following methodological problems:

- The need to create generalized and robust solutions. Some research has started to model generalization ability [8], but this area is far from explored.
- The need to process multi-modal information, i.e., information from different types of sensing devices [53, 79].
- The need to fuse models. This is related to multi-modal information processing. Assuming that each mode develops its distinct models, how can they be integrated into a cohesive picture of the situation [13, 111]?
- The need to make use of transfer and ‘lifelong’ learning. The former refers to the need to transfer models from one task to a related task [107, 113]. The latter refers to the problem of avoiding the unlearning of skills when tasks change, with the potential that earlier tasks are necessary to perform later again [74].
- Meta-learning—evolution in the natural world has shown its ability to create learning methods used by organisms. How can we recruit evolution to evolve learning methods for entire classes of problems encountered in the world [75, 82, 104]?
- Connected to the above questions, but more on the methodological level—how can we evolve modular and hierarchical models of the world? This is a very important question when scaling up to real-world problem dimensions [15, 50, 90].
- Models are only approximations of the systems they are supposed to reflect. How to close the reality gap [86]?
- There are ethical questions around the use of artificial intelligence: Can we secure its beneficial use while avoiding its misuse? What do we do to control the potential for AI weapons [21, 64]?
- Given the growth of applications in AI, how to avoid a development akin to a nuclear explosion, where AI/ML creates a strong positive feedback loop without leaving humans able to control it? Is that a realistic perspective [12]? There are also questions raised around sentience.
- The need to set up rules and methods for the interaction of AI/ML systems. How can AI/ML systems be allowed to negotiate outcomes among themselves, to the benefit of human users [19]?

On the more practical side, we see the following questions:

- How can we minimize the resource requirements of (E)ML tools? In particular, the need for energy, data storage, data and processing power.
- How can we ensure the acceptability of AI/ML suggested solutions? The usefulness of EML must be demonstrated in the context of human needs.
- How can we make sure that AI/ML/EML solutions are available to all users, not just the privileged few with large HPC centres?
- How can the quality of data be improved, or possibly curated automatically?
- How can we use EC to create models that generalize better?
- How can a solution for problems in principle be scaled up to the real-world domains it needs to prove to work in?
- How can the interaction between humans and machines be organized in a way that prevents us from becoming machines?

There are a number of even more fundamental questions, however, that will possibly remain unresolved for the foreseeable future, but still might be useful to guide inquiry.

The first we see is how we can handle novelty, and the open-endedness that comes with learning systems in interaction with humans and their societies. While open-endedness is a feature of our natural world under evolution, human-made (artificial) systems do not really possess open-endedness yet. They are closed systems that need repair, rather than open systems autonomously satisfying their needs. Can we manage to keep it that way, or do we need to take precautions to make open-ended systems still fit into a world we can call ours?

The second question sounds very practical or methodological, but it seems to us indeed a fundamental question: How can we understand the interaction between evolution and learning, how can we make use of it in the context of artificial learning and evolving systems? Is there another level of yet to be found phenomena that accelerates adaptation beyond learning?

Finally, we are used to discuss intelligence as a key concept of human existence, with some authors floating the idea of super-intelligence [14]. It is likely that entities with higher intelligence than humans can exist, e.g., humans with augmented intelligence. Early work on the intelligence amplifier by W. Ross Ashby [4] and intellect augmentation by Douglas Engelbart [26] comes to mind. But is there a limit to intelligence [24, 33], and at which level will it enforce itself? We believe that evolution teaches us that there is such a limit, but whether we are close is subject to exploration.

Acknowledgements This work is funded by the FCT–Foundation for Science and Technology, I.P./MCTES through national funds (PIDDAC), within the scope of CISUC R&D Unit–UIDB/00326/2020 or project code UIDP/00326/2020.

References

1. Ackley, D., Littman, M.: Interactions between learning and evolution. In: Langton, C., Taylor, C., Farmer, J., Rasmussen, S. (eds.) *Artificial Life II*, pp. 487–509. Addison-Wesley (1991)
2. Al-Sahaf, H., Bi, Y., Chen, Q., Lensen, A., Mei, Y., Sun, Y., Tran, B., Xue, B., Zhang, M.: A survey on evolutionary machine learning. *J. Royal Soc. New Zealand* **49**(2), 205–228 (2019)
3. Amershi, S., Begel, A., Bird, C., DeLine, R., Gall, H., Kamar, E., Nagappan, N., Nushi, B., Zimmermann, T.: Software engineering for machine learning: A case study. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), pp. 291–300. IEEE (2019)
4. Ashby, W.R.: *An Introduction to Cybernetics*. Chapman and Hall (1956)
5. Bailey, B.: The Impact Of Moore’s Law Ending (2018). <https://cacm.acm.org/news/232532-the-impact-of-moores-law-ending/fulltext> (Last accessed Oct 12 2022)
6. Baluja, S., Pomerleau, D., Jochem, T.: Towards automated artificial evolution for computer-generated images. *Connect. Sci.* **6**, 325–354 (1994)
7. Banzhaf, W., Nordin, P., Keller, R., Francone, F.: *Genetic Programming—An Introduction*. Morgan Kaufmann (1998)

8. Barbiero, P., Squillero, G., Tonda, A.: Modeling generalization in machine learning: A methodological and computational study. ArXiv preprint [arXiv:2006.15680](https://arxiv.org/abs/2006.15680) (2020)
9. Bedau, M.A.: Weak emergence. *Philos. Perspect.* **11**, 375–399 (1997)
10. Belle, V., Papantonis, I.: Principles and practice of explainable machine learning. *Front. Big Data* **39** (2021)
11. Beniaguev, D., Segev, I., London, M.: Single cortical neurons as deep artificial neural networks. *Neuron* **109**(17), 2727–2739 (2021)
12. Bentham, S.: Don't fear the reaper: Refuting Bostrom's superintelligence argument. ArXiv preprint [arXiv:1702.08495](https://arxiv.org/abs/1702.08495) (2017)
13. Blasch, E., Pham, T., Chong, C.-Y., Koch, W., Leung, H., Braines, D., Abdelzaher, T.: Machine learning/artificial intelligence for sensor data fusion-opportunities and challenges. *IEEE Aerosp. Electron. Syst. Mag.* **36**(7), 80–93 (2021)
14. Bostrom, N.: Superintelligence. Oxford University Press (2016)
15. Callebaut, W., Rasskin-Gutman, D., Simon, H.A.: Modularity: Understanding the Development and Evolution of Natural Complex Systems. MIT Press (2005)
16. Chalmers, D.J.: The evolution of learning: An experiment in genetic connectionism. In: Connectionist Models, pp. 81–90. Elsevier (1991)
17. Coello, C.C.: Evolutionary multi-objective optimization: A historical view of the field. *IEEE Comput. Intell. Mag.* **1**(1), 28–36 (2006)
18. Cramer, N.L.: A representation for the adaptive generation of simple sequential programs. In: Proceedings of the First International Conference on Genetic Algorithms (ICGA-1985), pp. 183–187 (1985)
19. Dai, T., Sycara, K., Zheng, R.: Agent reasoning in AI-powered negotiation. In: Handbook of Group Decision and Negotiation, pp. 1187–1211 (2021)
20. D'Ambrosio, D.B., Stanley, K.O.: A novel generative encoding for exploiting neural network sensor and output geometry. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, pp. 974–981 (2007)
21. de Ágreda, Á.G.: Ethics of autonomous weapons systems and its applicability to any ai systems. *Telecommun. Policy* **44**(6), 101953 (2020)
22. Dong, X., Yu, Z., Cao, W., Shi, Y., Ma, Q.: A survey on ensemble learning. *Front. Comput. Sci.* **14**, 241–258 (2020)
23. Du, M., Liu, N., Hu, X.: Techniques for interpretable machine learning. *Commun. ACM* **63**(1), 68–77 (2019)
24. Eden, A.H., Moor, J.H., Søraker, J.H., Steinhart, E.: Singularity Hypotheses. The Frontiers Collection. Springer, Berlin (2012)
25. Emes, R.D., Grant, S.G.: Evolution of synapse complexity and diversity. *Ann. Rev. Neurosci.* **35**, 111–131 (2012)
26. Engelbart, D.: Augmenting Human Intellect: A Conceptual Framework, Summary Report. Technical Report AFOSR-3233, Stanford Research Institute, Menlo Park, CA (1962)
27. Englander, A.C.: Machine learning of visual recognition using genetic algorithms. In: Proceedings of the 1st International Conference on Genetic Algorithms, pp. 197–201. Erlbaum Associates Inc, USA (1985)
28. Fernando, C., Banarse, D., Blundell, C., Zwols, Y., Ha, D., Rusu, A.A., Pritzel, A., Wierstra, D.: Pathnet: Evolution channels gradient descent in super neural networks. ArXiv preprint [arXiv:1701.08734](https://arxiv.org/abs/1701.08734) (2017)
29. Fernando, C., Szathmáry, E., Husbands, P.: Selectionist and evolutionary approaches to brain function: a critical appraisal. *Front. Comput. Neurosci.* **6**, 24 (2012)
30. Fogel, L., Owens, A., Walsh, M.: Artificial Intelligence through Simulated Evolution. Wiley, Chichester, WS, UK (1966)
31. Fogel, L.J.: Autonomous automata. *Ind. Res.* **4**, 14–19 (1962)
32. Forsyth, R.: BEAGLE-A Darwinian approach to pattern recognition. *Kybernetes* **10**, 159–166 (1981)
33. Fox, D.: The limits of intelligence. *Sci. Am.* **305**(1), 36–43 (2011)

34. Fraser, A.S.: Simulation of genetic systems by automatic digital computers i. introduction. *Australian J. Biol. Sci.* **10**(4), 484–491 (1957)
35. Friedberg, R.M.: A learning machine: Part i. *IBM J. Res. Develop.* **2**(1), 2–13 (1958)
36. Friedrichs, F., Igel, C.: Evolutionary tuning of multiple SVM parameters. *Neurocomputing* **64**, 107–117 (2005)
37. Gagné, C., Schoenauer, M., Sebag, M., Tomassini, M.: Genetic programming for kernel-based learning with co-evolving subsets selection. In: Runarsson, T.P., Beyer, H.-G., Burke, E., Merelo-Guervós, J.J., Whitley, L.D., Yao, X. (eds.) *Parallel Problem Solving from Nature-PPSN IX*, pp. 1008–1017. Springer, Berlin (2006)
38. Goldberg, D.E.: Computer-aided Gas Pipeline Operation using Genetic Algorithms and Rule Learning. University of Michigan (1983)
39. Gong, Y.-J., Chen, W.-N., Zhan, Z.-H., Zhang, J., Li, Y., Zhang, Q., Li, J.-J.: Distributed evolutionary algorithms and their models: A survey of the state-of-the-art. *Appl. Soft Comput.* **34**, 286–300 (2015)
40. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: *27th Advances in Neural Information Processing Systems Conference* (2014)
41. Gruau, F.: Genetic micro programming of neural networks. In: Kinnear, K.E., Jr. (ed.) *Advances in Genetic Programming*, pp. 495–518. MIT Press (1994)
42. Harris, W.: *Zero to Brain*. Princeton University Press (2022)
43. Hassan, M.B., Saeed, R.A., Khalifa, O., Ali, E.S., Mokhtar, R.A., Hashim, A.A.: Green machine learning for green cloud energy efficiency. In: *2022 IEEE 2nd International Maghreb Meeting of the Conference on Sciences and Techniques of Automatic Control and Computer Engineering (MI-STA)*, pp. 288–294. IEEE (2022)
44. Henderson, P., Hu, J., Romoff, J., Brunskill, E., Jurafsky, D., Pineau, J.: Towards the systematic reporting of the energy and carbon footprints of machine learning. *J. Mach. Learn. Res.* **21**(1), 10039–10081 (2020)
45. Holland, J.H.: Outline for a logical theory of adaptive systems. *J. ACM* **9**(3), 297–314 (1962)
46. Holland, J.H.: *Adaptation in Natural and Artificial Systems*. MIT Press, Cambridge, MA, USA (1975)
47. Holland, J.H., Reitman, J.S.: Cognitive systems based on adaptive algorithms. *SIGART Bull.* **63**, 49 (1977)
48. Howley, T., Madden, M.G.: The genetic kernel support vector machine: Description and evaluation. *Artif. Intell. Rev.* **24**(3–4), 379–395 (2005)
49. Jabbar, A., Li, X., Omar, B.: A survey on generative adversarial networks: Variants, applications, and training. *ACM Comput. Surv. (CSUR)* **54**(8), 1–49 (2021)
50. Kelly, S., Smith, R.J., Heywood, M.I., Banzhaf, W.: Emergent tangled program graphs in partially observable recursive forecasting and vizdoom navigation tasks. *ACM Trans. Evol. Learn. Optim.* **1**(3), 1–41 (2021)
51. Kim, H.B., Jung, S.H., Kim, T.G., Park, K.H.: Fast learning method for back-propagation neural network by evolutionary adaptation of learning rates. *Neurocomputing* **11**(1), 101–106 (1996)
52. Koeppe, P., Hamann, C.: A program for non-linear regression analysis to be used on desk-top computers. *Comput. Program. Biomed.* **12**(2–3), 121–128 (1980)
53. Korot, E., Guan, Z., Ferraz, D., Wagner, S.K., Zhang, G., Liu, X., Faes, L., Pontikos, N., Finlayson, S.G., Khalid, H., et al.: Code-free deep learning for multi-modality medical image classification. *Nat. Mach. Intell.* **3**(4), 288–298 (2021)
54. Koza, J.R.: Hierarchical genetic algorithms operating on populations of computer programs. In: *International Joint Conference on Artificial Intelligence (IJCAI-89)*, vol. 89, pp. 768–774 (1989)
55. Koza, J.R.: Genetic programming: A paradigm for genetically breeding populations of computer programs to solve problems. Technical report, Department of Computer Science, Stanford University, Stanford, CA, USA (1990)

56. Koza, J.R.: Genetic Programming—On the Programming of Computers by Means of Natural Selection. MIT Press (1992)
57. Lacoste, A., Luccioni, A., Schmidt, V., Dandres, T.: Quantifying the carbon emissions of machine learning. ArXiv preprint [arXiv:1910.09700](https://arxiv.org/abs/1910.09700) (2019)
58. Lehman, J., Clune, J., Misevic, D., et al.: The surprising creativity of digital evolution: A collection of anecdotes from the evolutionary computation and artificial life research communities. *Artif. Life* **26**(2), 274–306 (2020)
59. Liu, Y., Sun, Y., Xue, B., Zhang, M., Yen, G.G., Tan, K.C.: A survey on evolutionary neural architecture search. *IEEE Trans. Neural Networks Learn. Syst.* **34**, 550–570 (2021)
60. Lo Piano, S.: Ethical principles in machine learning and artificial intelligence: Cases from the field and possible ways forward. *Humanit. Soc. Sci. Commun.* **7**(1), 1–7 (2020)
61. Lu, Z., Cheng, R., Jin, Y., Tan, K.C., Deb, K.: Neural architecture search as multiobjective optimization benchmarks: Problem formulation and performance assessment. ArXiv preprint [arXiv:2208.04321](https://arxiv.org/abs/2208.04321) (2022)
62. Lu, Z., Whalen, I., Boddeti, V., Dhebar, Y., Deb, K., Goodman, E., Banzhaf, W.: NSGA-Net: Neural Architecture Search using Multi-objective Genetic Algorithm. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 419–427 (2019)
63. Ma, X., Li, X., Zhang, Q., Tang, K., Liang, Z., Xie, W., Zhu, Z.: A survey on cooperative co-evolutionary algorithms. *IEEE Trans. Evol. Comput.* **23**(3), 421–441 (2018)
64. Maas, M.M.: How viable is international arms control for military artificial intelligence? three lessons from nuclear weapons. *Contemp. Secur. Policy* **40**(3), 285–311 (2019)
65. Machado, P., Baeta, F., Martins, T., Correia, J.: GP-based generative adversarial models. In: Trujillo, L., Winkler, S.M., Silva, S., Banzhaf, W. (eds.) *Genetic Programming Theory and Practice XIX, Genetic and Evolutionary Computation*, pp. 117–140. Springer, Ann Arbor, USA (2022)
66. Machado, P., Romero, J., Manaris, B.Z.: Experiments in computational aesthetics. In: Romero, J., Machado, P. (eds.) *The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music*, Natural Computing Series, pp. 381–415. Springer (2008)
67. Mei, Y., Chen, Q., Lensen, A., Xue, B., Zhang, M.: Explainable artificial intelligence by genetic programming: A survey. *IEEE Trans. Evol. Comput.* **27**(3), 621–641 (2023)
68. Miller, J.F.: Designing Multiple ANNs with Evolutionary Development: Activity Dependence. In: Banzhaf, W., Trujillo, L., Winkler, S., Worzel, B. (eds.) *Genetic Programming Theory and Practice XVIII*, pp. 165–180. Springer Nature Singapore, Singapore (2022)
69. Montana, D.J., Davis, L., et al.: Training feedforward neural networks using genetic algorithms. In: International Joint Conference on Artificial Intelligence (IJCAI-89), vol. 89, pp. 762–767 (1989)
70. Nguyen, A.M., Yosinski, J., Clune, J.: Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. CoRR, [arXiv:1412.1897](https://arxiv.org/abs/1412.1897) (2014)
71. Nolfi, S., Floreano, D.: Learning and evolution. *Autonom. Robot.* **7**, 89–113 (1999)
72. Olson, R.S., Moore, J.H.: Tpot: A tree-based pipeline optimization tool for automating machine learning. In: Workshop on Automatic Machine Learning, pp. 66–74. PMLR (2016)
73. Paleyes, A., Urma, R.-G., Lawrence, N.D.: Challenges in deploying machine learning: a survey of case studies. *ACM Comput. Surv.* **55**(6), 1–29 (2022)
74. Parisi, G.I., Kemker, R., Part, J.L., Kanan, C., Wermter, S.: Continual lifelong learning with neural networks: A review. *Neural Networks* **113**, 54–71 (2019)
75. Peng, H.: A comprehensive overview and survey of recent advances in meta-learning. ArXiv preprint [arXiv:2004.11149](https://arxiv.org/abs/2004.11149) (2020)
76. Pessach, D., Shmueli, E.: A review on fairness in machine learning. *ACM Comput. Surv. (CSUR)* **55**(3), 1–44 (2022)
77. Phanendra Babu, G., Narasimha Murty, M.: A near-optimal initial seed value selection in k-means means algorithm using a genetic algorithm. *Patt. Recogn. Lett.* **14**(10), 763–769 (1993)
78. Pujari, K.N., Miriyala, S.S., Mittal, P., Mitra, K.: Better wind forecasting using evolutionary neural architecture search driven green deep learning. *Expert Syst. Appl.* **214**, 119063 (2023)

79. Qiao, Y., Zhao, L., Luo, C., Luo, Y., Wu, Y., Li, S., Bu, D., Zhao, Y.: Multi-modality artificial intelligence in digital pathology. *Brief. Bioinform.* **23**(6), bbac367 (2022)
80. Radford, A., Kim, J.W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., Sutskever, I.: Learning transferable visual models from natural language supervision. *CoRR arXiv:2103.00020* (2021)
81. Raghavan, V.V., Birchard, K.: A clustering strategy based on a formalism of the reproductive process in natural systems. In: *Proceedings of the 2nd Annual International ACM SIGIR Conference on Information Storage and Retrieval: Information Implications into the Eighties, SIGIR '79*, pp. 10–22. Association for Computing Machinery , New York, NY, USA (1979)
82. Real, E., Liang, C., So, D., Le, Q.: Automl-zero: Evolving machine learning algorithms from scratch. In: *International Conference on Machine Learning*, pp. 8007–8019. PMLR (2020)
83. Rechenberg, I.: Cybernetic solution path of an experimental problem. Technical Report Library Translation No. 1122, Royal Aircraft Establishment, Farnborough (1965)
84. Sagi, O., Rokach, L.: Ensemble learning: A survey. *Wiley Interdisc. Rev. Data Min. Knowl. Discov.* **8**(4), e1249 (2018)
85. Salimans, T., Ho, J., Chen, X., Sidor, S., Sutskever, I.: Evolution strategies as a scalable alternative to reinforcement learning (2017)
86. Salvato, E., Fenu, G., Medvet, E., Pellegrino, F.A.: Crossing the reality gap: A survey on sim-to-real transferability of robot controllers in reinforcement learning. *IEEE Access* **9**, 153171–153187 (2021)
87. Samuel, A.L.: Some studies in machine learning using the game of checkers. *IBM J. Res. Develop.* **3**(3), 210–229 (1959)
88. Schaffer, J.D.: Some experiments in machine learning using vector evaluated genetic algorithms. Technical report, Vanderbilt Univ., Nashville, TN (USA) (1985)
89. Shinozaki, T., Watanabe, S.: Structure discovery of deep neural network based on evolutionary algorithms. In: *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4979–4983. IEEE (2015)
90. Simon, H.A.: *The Sciences of the Artificial*, Reissue of the third edition with a New Introduction by John Laird. MIT Press (2019)
91. Sims, K.: Evolving 3d morphology and behavior by competition. *Artif. Life* **1**(4), 353–372 (1994)
92. Sims, K.: Evolving virtual creatures. In: Schweitzer, D., Glassner, A.S., Keeler, M. (eds.) *Proceedings of the 21th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1994*, Orlando, FL, USA, July 24–29, 1994, pp. 15–22. ACM (1994)
93. So, D., Le, Q., Liang, C.: The evolved transformer. In: Chaudhuri, K., Salakhutdinov, R. (eds.) *Proceedings of the 36th International Conference on Machine Learning*, vol. 97, pp. 5877–5886 (2019)
94. Spector, L., Alpern, A.: Induction and recapitulation of deep musical structure. In: *Proceedings of the IFCAI-95 Workshop on Artificial Intelligence and Music*, pp. 41–48 (1995)
95. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. *Evol. Comput.* **10**(2), 99–127 (2002)
96. Stepney, S.: Life as a cyber-bio-physical system. In: Trujillo, L., Winkler, S., Silva, S., Banzhaf, W. (eds.) *Genetic Programming—Theory and Practice. GPTP XIX*, pp. 167–200. Springer Nature, Singapore (2023)
97. Sze, V.: Designing hardware for machine learning: The important role played by circuit designers. *IEEE Solid-State Circ. Mag.* **9**(4), 46–54 (2017)
98. Tefikani, A., Tahmassebi, A., Banzhaf, W., Gandomi, A.H.: Evolutionary machine learning: A survey. *ACM Comput. Surv.* **54**(8) (2021)
99. Tkachenko, A., Brovinskaya, N., Kondratenko, Y.: Evolutionary adaptation of control processes in robots operating in nonstationary environments. *Mech. Mach. Theory* **18**(4), 275–278 (1983)
100. Tornede, T., Tornede, A., Hanselle, J., Wever, M., Mohr, F., Hüllermeier, E.: Towards green automated machine learning: Status quo and future directions. *ArXiv preprint arXiv:2111.05850* (2021)

101. Turing, A.M.: Computing machinery and intelligence. *Mind* **59**(236), 433–460 (1950)
102. Turing, A.M.: Intelligent machinery, a heretical theory. In: The Essential Turing. Oxford University Press (2004)
103. van der Maaten, L., Hinton, G.: Visualizing data using t-sne. *J. Mach. Learn. Res.* **9**(86), 2579–2605 (2008)
104. Wang, J.X.: Meta-learning in natural and artificial intelligence. *Current Opin. Behav. Sci.* **38**, 90–95 (2021)
105. Wang, Z., Gao, S., Wang, J., Yang, H., Todo, Y.: A dendritic neuron model with adaptive synapses trained by differential evolution algorithm. In: Computational Intelligence and Neuroscience (2020)
106. Weiss, E.: Arthur Lee Samuel (1901–90). *IEEE Ann. History Comput.* **14**(3), 55–69 (1992)
107. Weiss, K., Khoshgoftaar, T.M., Wang, D.: A survey of transfer learning. *J. Big Data* **3**(1), 1–40 (2016)
108. Yao, X.: Evolutionary artificial neural networks. *Int. J. Neural Syst.* **4**(3), 203–222 (1993)
109. Yao, X.: Evolving artificial neural networks. *Proceed. IEEE* **87**(9), 1423–1447 (1999)
110. Yerushalmi, U., Teicher, M.: Evolving synaptic plasticity with an evolutionary cellular development model. *PLoS ONE* **3**(11), e3697 (2008)
111. Zhang, Q., Liu, Y., Blum, R.S., Han, J., Tao, D.: Sparse representation based multi-sensor image fusion for multi-focus and multi-modality images: A review. *Inf. Fusion* **40**, 57–75 (2018)
112. Zhou, Y., Kantarcioglu, M., Xi, B.: A survey of game theoretic approach for adversarial machine learning. *Wiley Interdisc. Rev. Data Mining Knowl. Discov.* **9**(3), e1259 (2019)
113. Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., He, Q.: A comprehensive survey on transfer learning. *Proceed. IEEE* **109**(1), 43–76 (2020)

Chapter 2

Evolutionary Supervised Machine Learning



Risto Miikkulainen

Abstract This chapter provides an overview of evolutionary approaches to supervised learning. It starts with the definition and scope of the opportunity, and then reviews three main areas: evolving general neural network designs, evolving solutions that are explainable, and forming a synergy of evolutionary and gradient-based methods.

2.1 Introduction

In supervised learning problems, in principle, the correct answers are known for each input. In the most general sense, the learner has access to a supervisor who tells what needs to be done for each input; in practice, such information is collected into a dataset ahead of time and sampled during the learning, and the dataset may be noisy and incomplete. The challenge in supervised learning is to construct a model that accurately predicts the outputs for new samples that are not in the dataset. Thus, the model imitates known behavior in a way that generalizes to new situations as well as possible (Fig. 2.1).

Examples of supervised learning include classifying objects in visual images; diagnosing pathologies in X-ray images; predicting the next word in a stream of text; predicting future values of stocks; predicting the weather. Given that data is now collected routinely across various human activities in business, healthcare, government, education, and so on, many opportunities exist for using supervised learning in the real world.

The standard approach is gradient descent on neural networks, i.e., making small changes to the network weights (or parameters) in order to reduce the error (or loss) on known examples as quickly as possible. This approach has been commonplace

R. Miikkulainen (✉)

Department of Computer Science, University of Texas at Austin, Austin, TX 78712, USA
e-mail: risto@cs.utexas.edu

Cognizant AI Labs, San Francisco, CA 94501, USA

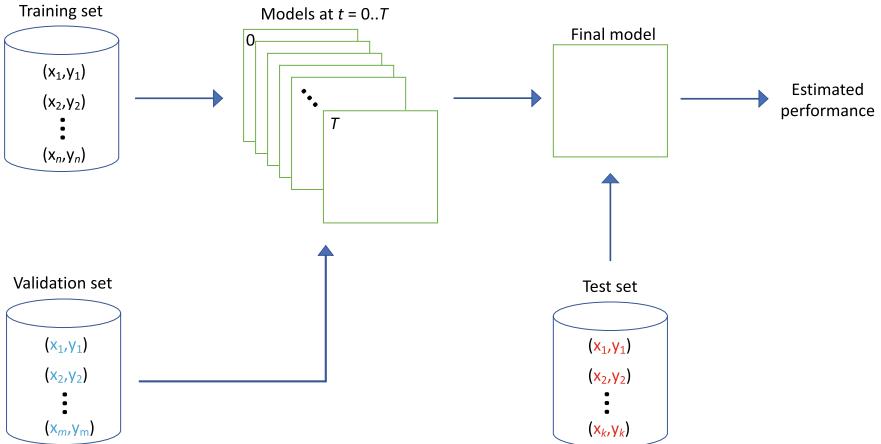


Fig. 2.1 A general setup for supervised learning. Each input example x_i in the dataset is paired with the desired output y_i . The dataset is split into a training set that is used to modify the model over time, a validation set that is used to decide which model to choose as the final result of learning, and a test set that is used to estimate performance of the model on future examples. The system thus learns a model that imitates the known behavior and generalizes to future situations as well as possible

since the 1980s [69]. However, given a million-fold increase in computing power from just a couple of decades ago [67], it has recently become possible to scale supervised learning techniques to much larger datasets and architectures. Much of the power of modern supervised learning lies in this scale-up [11]. While the basis for this success still needs to be understood better, it has already revolutionized the field of artificial intelligence.

Evolutionary machine learning constitutes a distinctly different approach to supervised learning. It is not an incremental improvement method based on gradients, but instead a search method guided by fitness feedback, or reinforcement. It is therefore more general, but also cannot as naturally take advantage of massive datasets as gradient-based methods can. On the other hand, many domains in the real world do exist where such a scale-up is not possible. Datasets are sometimes very specific to the application (e.g., in a business domain, or medicine, or engineering design) and consist of thousands, instead of millions, of samples. In such domains, evolutionary supervised learning can provide two advantages, leading to two opportunities for advancing supervised machine learning.

The first opportunity is to take into account other goals than simply accuracy, i.e., to evolve general neural network designs. Note that a supervised signal (i.e., a gradient vector), can be converted to a fitness signal (i.e., a scalar, such as a norm of the gradient). Specific information will be lost; on the other hand, it is then possible to combine the accuracy goal with other goals. For instance, the networks evolved can be small enough to be transparent. It may be possible to improve regularization in

such networks at the same time. Such networks can potentially take better advantage of minimal computing resources, and even fit better to hardware constraints.

The second opportunity is explainability. That is, gradients are inherently opaque, and machine learning systems based on them are essentially black boxes. This opaqueness makes it difficult to deploy them in many applications in the real world. For instance, in domains like health care and finance, it is important to be able to verify that the information used to draw the conclusion was relevant, unbiased, and trustworthy. In contrast, evolutionary machine learning can be used to discover solutions that are transparent. For instance, a solution can be represented in terms of decision trees, classifiers, programs, and rule sets.

A third opportunity for evolutionary supervised learning has emerged with the scale-up itself. Deep learning systems have become extremely large and complex, with many elements that interact nonlinearly. Like in many areas of system design, it is no longer possible for humans to design them optimally. However, evolutionary metalearning can be used to configure the architectures, hyperparameters, loss functions, activation functions, data selection and augmentation, and even learning methods so that the resulting deep learning models perform as well as possible. In this manner, it is possible to use evolution synergistically with gradient descent. Such automatic machine learning makes deep learning systems accessible to more people, and the designs can be optimized for size and data, making it possible to apply it more broadly.

These three opportunities will each be reviewed in the sections that follow.

2.2 Evolving General Neural Network Designs

In evolving general neural network designs, gradient descent is replaced entirely with evolutionary optimization. Fitness is still at least partially based on the loss or accuracy on the supervised dataset, but only as a scalar value. Most importantly, evolution can be used to optimize other aspects of the design at the same time, such as explainability (e.g., through complexification), size (e.g., to improve regularization, or to fit within computational constraints), of hardware fit more generally (e.g., using threshold units or other non-differentiable components). These benefits currently apply to compact neural networks but could extend to deep learning as well.

2.2.1 Compact Neural Networks

Neural networks have evolved for a long time, starting from the direct approaches of [54]. Most of the techniques were developed for sequential decision tasks, especially those that are POMDP (Partially Observable Markov Decision Processes), i.e., tasks where the state is not fully observable and where recurrency thus matters. However, they can be used in supervised learning as well.

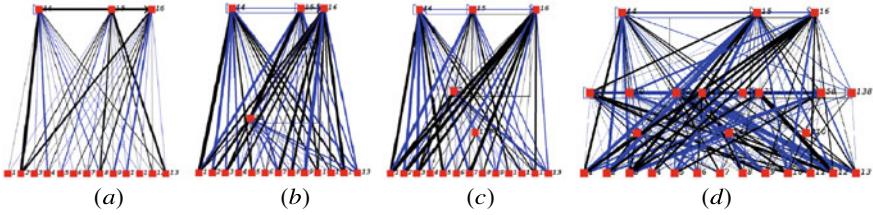


Fig. 2.2 Evolutionary Discovery through Complexification As the NEAT neuroevolution method [80] complexifies networks, the behaviors that these networks generate become more complex as well. It is thus possible to identify how the network generates the behavior it does. In the simultaneous pursuit–evasion–foraging task for simulated Khepera robots, the approach discovered **a** opponent following, **b** behavior selection, **c** opponent modeling, and **d** opponent control through several such complexification steps [81]. Nodes are depicted as red squares and numbered in the order they were created. Positive connections are black and negative are blue, recurrent connections are indicated by triangles, and the width of the connection is proportional to its strength. Complexifying evolution thus provides a way of understanding network performance. Figures from [78]

One example is the NEAT method [58, 80], which optimizes both the architecture and the weights of a neural network. The main idea is to start small and complex, which results in compact networks that are possible to understand. NEAT starts with a population of networks where the inputs are connected directly to the outputs. As evolution progresses, hidden nodes and recurrent as well as feedforward connections are added. Such architectural innovations are protected through specialization, i.e., they do not have to compete with other architectures until their parameters have been sufficiently optimized. Each innovation receives a unique identifier that allows taking advantage of crossover even among population of diverse architectures. As a result, NEAT evolves compact architectures where every complexification is evaluated and found useful or not.

Such compact networks are very different from deep learning networks in that they often employ principled and interpretable designs. For instance, in a combined foraging–pursuit–evasion task of simulated Khepera robots [81], evolution first discovered a simple follow-the-opponent behavior that was often successful by chance: The agent occasionally crashed into the opponent when it had more energy than the opponent (Fig. 2.2a). It then evolved a hidden node that allowed it to make an informed switch between behaviors: Attack when it had high energy, and rest when it did not (Fig. 2.2b). Another added node made it possible to predict the agent’s own and its opponent’s energy usage from afar and attack only when victory was likely (Fig. 2.2c). The most complex strategy, with several more nodes and complex recurrent connections between them, allowed the agent to predict also the opponent’s behavior, encourage it to make mistakes, and take advantage of the mistakes to win (Fig. 2.2d).

Although this example illustrates a sequential decision task, the same principles apply to supervised learning with NEAT. Note that these principles are very different from those in deep learning. Performance with very large networks is based on over-

parameterization where individual components perform only minimal operations: for instance, the residual module in ResNet architectures combines bypassing the module with the transformation that the module itself computes [32]. In contrast, in NEAT every complexification is there for a purpose that can in principle be identified in the evolutionary history. It thus offers an alternative solution, one that is based on principled neural network design.

This kind of compact evolved neural network can be useful in four ways in supervised learning:

1. They can provide an explainable neural network solution. That is, the neural network still performs based on recurrency and embeddings, but its elements are constructed to provide a particular functionality, and therefore its behavior is transparent [1, 38, 40, 81].
2. They can provide regularized neural network solutions, instead of overfitting to the dataset. The networks are compact, which generally regularization [19, 57, 66], and they are chosen based on their overall performance instead of fine-tuned to fit individual examples. This property should be particularly useful when the datasets are relatively small, which is the case in many practical applications. Thus, they can extend the scope of machine learning applications.
3. They can utilize minimal hardware resources well. The advantages of deep learning networks do not emerge until a very large number of parameters. If the hardware does not allow that scale (e.g., in edge devices), evolved networks provide an alternative principle that can be optimized to the given resources.
4. They can be constructed to fit hardware constraints. Gradient descent in principle requires high-precision weights and differentiable activation functions that are expensive to implement in hardware. In contrast, evolution can be used to optimize the performance of networks with, e.g., quantized weights, linear threshold units, or FPGA-compatible components that are easier to implement [18, 48, 73].

While examples of each of these advantages exist already, their large-scale applications are still future work and an interesting research opportunity. Another interesting opportunity is to scale them up to large networks, discussed next.

2.2.2 Deep Networks

As discussed above, evolution of entire networks usually focuses on small, recurrent architectures with up to hundreds of thousands of parameters. Evolutionary operations can still search such a space effectively. At the outset, it seems difficult to scale this approach to deep learning networks, which routinely contain hundreds of millions of parameters, and sometimes up to hundreds of billions of them.

There are special techniques that may make it possible to optimize solutions in such very high-dimensional spaces. For instance, [15] showed that in the specific case of metal casting scheduling, it was possible to establish a constrained search, and solve problems with up to a billion parameters. It may be possible to develop

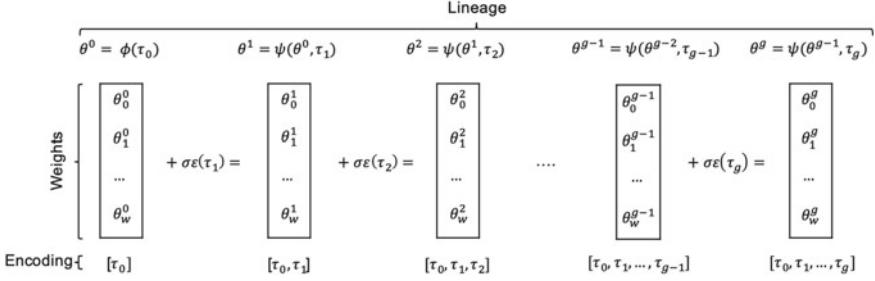


Fig. 2.3 Compact Indirect Encoding of Deep Neural Networks. In the approach of [82], a population of initial networks with parameters θ^0 are first created through an initialization function ϕ . Subsequent generations are created through mutations ψ seeded τ_i . Only the seeds τ_i need to be stored and evolved; the current population can be created by applying the same sequence of mutations to θ^0 . In this way, indirect encoding makes it possible to represent deep neural networks compactly enough so that they can be evolved in their entirety. Figure from [82]

similar techniques for neuroevolution as well and thereby scale up to deep learning networks.

Another approach is to utilize indirect encodings. Instead of optimizing each parameter independently, a coding is evolved that is then mapped to the final design through a developmental or a generative process. For instance, in the HyperNEAT approach [79], such a decoding neural network is evolved to output weight values for another neural network that actually performs the task. The performance network is embedded into a substrate where each of its neurons is located at particular coordinates; given the coordinates of the source and target neuron as input, the decoding neural network generates the weight of the connection between them as its output. Importantly, it is possible to sample the substrate at different resolutions, i.e., embed a very large number of neurons in it, and thus use the same decoding network to generate very large performance networks. The decoding networks are termed compositional pattern-producing networks because the weights they generate often follow regular patterns over the substrate. This observation suggests that the HyperNEAT approach could be used to generate structures over space, such as 2D visual or 1D time series domains. Whether it can be extended to layered structures and general architectures remains an open question.

Evolving such structured deep learning networks may be achieved by other means. For instance, [82] developed a method where each individual network is represented indirectly as a sequence of mutation operators performed on an initial set of weight parameters (Fig. 2.3). The initial set may have millions of parameters, but they only need to be stored once for each individual lineage. The mutations can be encoded efficiently as a sequence of seeds that generate the changes through a precomputed table. Thus, the compression rate depends on the number of generations but is in the order of $10^3 - 10^4$ -fold. It thus makes it possible to evolve deep learning networks with millions of parameters, as was demonstrated in several tasks in the Atari game-playing domain.

While evolution of entire very large networks is still a new area, these initial results suggest that it may indeed be possible. The benefits of general design, reviewed in the previous section, may then apply to them as well.

2.3 Evolving Explainable Solutions

Most AI today is based on neural networks. Given the vast amount of available data and compute, impressive performance is indeed possible in many real-world classification and prediction tasks. However, neural networks are inherently opaque. The knowledge they have learned is embedded in very high-dimensional vector spaces, with a lot of redundancy and overlap, and with nonlinear interactions between elements. Even though they may perform well on average, it is difficult to deploy such a system when we do not understand how it arrives at a particular answer in any individual case. The system may be overfitting, fooled by adversarial input, missing an important factor, or utilizing unfair bias to make its decision [13, 37, 72]. Explainability has thus emerged as one of the main challenges in taking AI to the real world.

Evolving compact neural networks through complexification may make their function transparent. However, explainability in supervised learning requires more: For each new example, the system should be able to identify the information and principles used to arrive at the decision. Structures different from neural networks are thus needed, such as decision trees, classifiers, programs, and rules. While they cannot be easily modified by gradients, evolutionary machine learning can be used to construct them, as reviewed in this section.

2.3.1 Decision Trees

Decision trees are a classical supervised learning method with low computational overhead and transparent, explainable performance [9, 60, 61]. In its most basic form, samples are represented as vectors of features, and the tree is used to classify them into discrete categories. Each node of the tree looks at one feature, and based on its value, passes the sample down to one of the nodes below it; the leaf nodes each assign a category label.

The traditional approach to constructing a decision tree is to do it top down in a greedy fashion: At each step, a feature that is deemed most useful for branching is chosen, and the process continues recursively in each branch until all training samples in a branch have the same category label. The choice can be based, e.g., on the information gain in the branching, which is a heuristic aimed at constructing simple and shallow trees, which in turn are likely to generalize better.

However, the greedy construction is often suboptimal. It is difficult to take into account interactions between features, and construct trees that are balanced, and often

trees end up complex and prone to overfitting [4]. Evolutionary search provides an alternative: it can search for optimal trees globally, taking into account both accuracy and size objectives [4].

A natural approach to evolving decision trees is to represent them, indeed, as trees where the nodes specify a comparison of a particular feature with a constant value, or a label if the node is a leaf [2]. Fitness can then consist of accuracy in the training set but also include complexity of the tree, such as depth and balance. Such approaches achieve competitive performance, e.g., in the UCI benchmarks, especially in tasks that are prone to overfitting [39].

One way to improve generalization with decision trees is to form ensembles of them, including methods such as random forests, where different trees are trained with different parts of the dataset [8]. Evolutionary optimization can be applied to such ensembles as well: The nodes of the tree (represented, e.g., as feature+value pairs) can be concatenated into a vector, and multiple such vectors representing the entire ensemble can then be used as an individual in the population. The resulting evolutionary ensembles can outperform, e.g., the standard random forests and AdaBoost methods of forming decision tree ensembles [16].

Thus, evolutionary optimization provides a potentially powerful way to overcome some of the shortcomings of decision trees, thus having a large impact in domains with few examples, meaningful features, and the need for transparent decision-making.

2.3.2 Learning Classifier Systems

The two classic evolutionary machine learning approaches, learning classifier systems (LCS) and genetic programming (GP), can be applied to supervised learning tasks as well. They may perform well on specific domains, especially when there is too little data to take advantage of deep learning. However, like decision trees, their solution structure is transparent and they can thus be used to construct explainable solutions.

LCS has a long history of development, and includes many variations [10, 14, 34, 88]. The approach was originally developed for reinforcement learning problems, and a more thorough overview of it will be given in that context in Chap. 4. Often a distinction is made between the Michigan-style LCS, where the population consists of individual classifiers and the solution of the entire population, and Pittsburgh-style LCS, where the population consists of sets of classifiers and the solution is the best such set. Pittsburgh-style LCS is often referred to as ruleset evolution and will be discussed in Sect. 2.3.4.

In Michigan-style LCS, the population consists of individual IF–THEN rules whose antecedents specify feature values (0, 1, don't_care) in the environment, and the consequence is a binary classification (0, 1). In a supervised setting, the rules that match an input example are assigned fitness based on whether their classification is correct or not. The fitness is accumulated over the entire training set, and fitness is then used as the basis for parent selection. Offspring rules are generated through crossover

and mutation as usual in a genetic algorithm. The population can be initialized to match training examples, and it is grown incrementally, which makes learning and performance easier to understand.

For instance, LCS methods of supervised learning have been developed for data mining in noisy, complex problems such as those in bioinformatics [89]. Expert knowledge can be encoded as probability weights and attribute tracking, guiding search toward more likely solutions. Applied to a large number of single-nucleotide polymorphism datasets, the approach proved to be effective across a range of epistasis and heterogeneity.

The final set of LCS rules is transparent, although it can grow very large and difficult to interpret. Methods for compacting and filtering it have been developed to improve interpretability. However, Pittsburgh-style rule evolution often results in smaller and more interpretable rule sets, as will be discussed in Sect. 2.3.4.

2.3.3 *Genetic Programming*

Along the same lines, GP is primarily a method for reinforcement learning domains and will be reviewed in detail in that context in Chap. 4. The main idea in GP is to evolve programs, usually represented by trees of elementary operations [41]. This idea is very general, and also applies to the supervised learning context. The program can represent a function that transfers inputs to outputs. Those inputs and outputs can originate from a supervised dataset, and fitness for a program is measured based on how often the output labels are correct. GP can thus evolve programs that perform well in the supervised task.

The idea is that domain knowledge about the features and operations can be encoded in the search space of the program. While in deep learning approaches, such knowledge is usually extracted from the examples, GP only needs to find how to use it. Therefore, it is possible to learn from much fewer examples. Also, the programs are transparent, and therefore it is possible to explain how the system arrives at an answer.

For instance, in the evolutionary deep learning GP (EDLGP) system [5], GP was used to classify images in standard object datasets CIFAR-10, SVHN, FashionMNIST, and two face image datasets. Starting from images as input, the programs consisted of several layers of processing, including image filtering, feature extraction, concatenation, and classification. At each level, a number of operators were provided as elements, and the goal was to find a tree that utilizes these operators at the appropriate levels most effectively. Thus, much knowledge about image processing was provided to the search, making it possible to construct effective functions with only a few training examples. Indeed, the best-evolved trees performed much better than standard deep learning approaches such as CNN and ResNet when only one to eight training examples were available for each class; when 10–128 were available, the two approaches were roughly comparable.

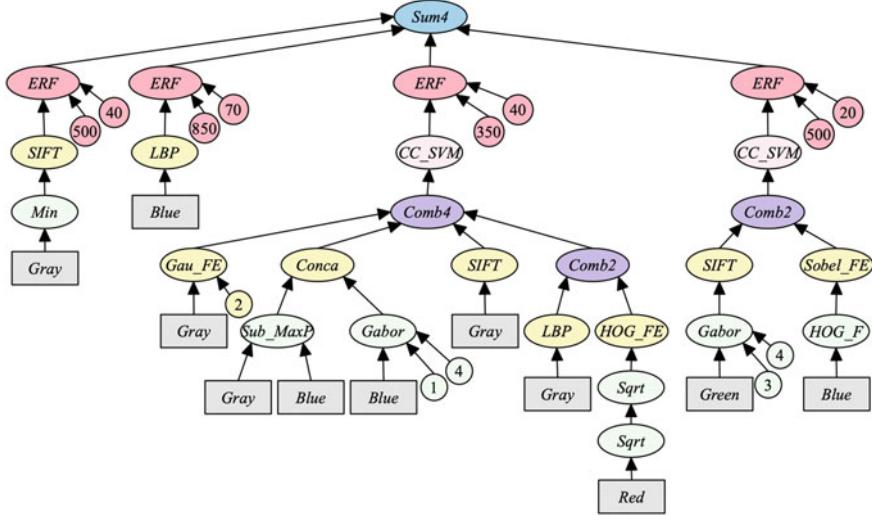


Fig. 2.4 An image classification tree discovered by GP. The tree was evolved with a training set of 80 images per class in CIFAR-10, reaching accuracy of 52.26%, which is better than deep learning approaches [5]. The gray nodes at the bottom indicate input channels; green nodes perform filtering operations; yellow nodes extract features; purple nodes form combinations of features; red modes make classification decisions. Taking advantage of image processing knowledge given in terms of layers and elements, the operations are chosen and the tree structure is formed through GP. It results in effective classification even with very few examples, and the classification decisions are transparent. Figure from [5]

Moreover, the resulting trees are transparent, taking advantage of operations that make sense in the domain (Fig. 2.4). Thus, the example demonstrates once again that evolutionary supervised learning makes sense in low-data environments in general, and also provides a level of explainability that is missing from the standard deep learning approaches.

2.3.4 Rulesets

Compact neural networks, decision trees, classifier populations, and programs may be transparent, but the most natural and clear form of explainability is in terms of rules. Indeed, AI has a rich tradition of rule-based systems [31]. Within their scope, they provide a rigorous inference mechanism based on logic. Importantly, they also provide explainability in that the rules specify exactly what knowledge is used to arrive at a decision, and how the decision is made. In other words, they provide precisely what is missing from current neural network-based AI.

On the other hand, rule-based systems are usually constructed by hand, to encode knowledge of human experts, instead of learning such knowledge from supervised

datasets. Rule-based systems are thus an important opportunity for evolutionary machine learning. Whereas rules cannot be learned through gradient descent, they can be learned through evolution.

First, rules are represented in a structure that can be evolved with, e.g., crossover and mutation operators. For instance, an individual in a population can consist of a set of rules, each with a left-hand side that consists of logical clauses and/or arithmetic expressions based on domain features, and a right-hand side that specifies a classification or prediction [71, 76]. Then, the number of rules, their order, the number of features, and the features themselves, the logical and arithmetic operations between them, and coefficients are evolved, as are the actions and their coefficients on the right-hand side. Furthermore, there can be probabilities or confidence values associated with each rule, and ways of combining outputs of multiple rules can be evolved as well.

In other words, rule-set evolution can take advantage of existing representations in Turing-complete rule-based systems. As usual in supervised learning, the fitness comes from loss, or accuracy, across the dataset. Instead of coding the rule-based system by hand, the entire system can then be evolved. The result, in principle, is explainable machine learning.

This general approach has already been demonstrated in several decision-making tasks, including stock trading, game playing, robotic control, and diabetes treatment recommendation [70, 71]. However, it has also been applied to the supervised task of blood-pressure prediction in intensive care (Fig. 2.5; [33, 71]). Based on a time series of blood-pressure measurements, that task was to predict whether the patient will go into a septic shock in the next 30 minutes, while there is still time to prevent it. Indeed, the evolved rulesets achieved an accuracy of 0.895 risk-weighted error on the unseen set, with a true positive rate of 0.96 and a false positive rate of 0.39 (Fig. 2.5). Most importantly, the rules were evaluated by emergency-room physicians who found them meaningful. Without such explicit rules, it would be very hard to deploy the prediction system. With rules, human experts can check the AI’s reasoning and decide whether to trust it, thus making the AI helpful as a first indicator of potential problems. Evolution of rulesets in this supervised task makes it possible.

2.4 Evolutionary Metalearning

The idea of metalearning is to enhance the general setup of the supervised learning system through a separate learning process. The two systems thus work synergistically; for instance, the fitness of an evolved neural network design is based on the performance of supervised training of that network. Metalearning is useful, and even necessary, because modern learning systems have become too large and complex for humans to optimize [17, 35, 45, 74]. There are many dimensions of design, such as network topology, components, modularity, size, activation function, loss function,

1. ($Mean[4] < 72.75\text{mmHg}$) & ($Kurtosis[3] < 4.09$)	→ Low
2. ($Skew[10] > 2.01$) & ($Mean[8] < 88.92\text{mmHg}$) & ($Skew[4] < 0.15$)	→ Normal
3. ($Mean[0] < 72.75\text{mmHg}$)	→ Low
4. ($Mean[10] < 73.10\text{mmHg}$)	→ Low
5. ($Mean[1] < 121.96\text{mmHg}$) & ($Mean[4] > 88.92\text{mmHg}$) & ($Mean[1] > 73.10\text{mmHg}$)	→ High
6. ($Mean[0] < 97.53\text{mmHg}$)	→ Normal
7. ($Mean[0] < 97.53\text{mmHg}$) & ($Kurtosis[0] > 12.71$)	→ Normal
8. ($Mean[4] < 72.75\text{mmHg}$) & ($Kurtosis[7] > 4.03$)	→ Low
9. ($Mean[4] > 121.96\text{mmHg}$) & ($Kurtosis[5] > 12.71$) & ($Kurtosis[3] > 1.00$)	→ Normal
10. ($Std[0] < 10.76$)	→ High
11. ($Kurtosis[0] > 1.00$)	→ High
12. ($Mean[0] < 72.75\text{mmHg}$) & ($Std[4] > 0.01$)	→ Low
13. ($Kurtosis[0] < 4.09$) & ($Skew[3] > 2.01$)	→ Normal
14. ($Skew[9] > 0.06$)	→ High
15. ($Skew[0] < 1.95$)	→ High
16. ($Mean[0] < 72.75\text{mmHg}$) & ($Mean[5] < 52.12\text{mmHg}$)	→ Low
17. Default	→ Normal

Fig. 2.5 A transparent and explainable rule set evolved to predict blood pressure. EVOTER discovered sets of features at specific time points to provide a useful signal for prediction. For instance, $Std[4]$ specifies the standard deviation of the aggregated mean arterial pressure (MAP) over 4 minutes earlier. The evolved rules predict sepsis within 30 mins with a 0.895 risk-weighted error on the unseen set, with a true positive rate of 0.96 and a false positive rate of 0.394. Most importantly, the rules are interpretable and meaningful to experts, which makes it much easier to deploy in practice compared to, e.g., black-box neural network models. Figure from [71]

learning rate and other learning parameters, data augmentation, and data selection. These dimensions often interact nonlinearly, making it very hard to find the best configurations.

There are many approaches to metalearning, including gradient-based, reinforcement learning, and Bayesian optimization methods [17, 68]. They can be fast and powerful, especially within limited search spaces. However, the evolutionary approach is most creative and versatile [44, 47, 53]. It can be used to optimize many of the design aspects mentioned above, including those that focus on optimizing discrete configurations that are less natural for the other methods. It can also be used to achieve multiple goals: It can be harnessed not only to improve state-of-the-art performance but also to achieve good performance with little expertise, fit the architecture to hardware constraints, take advantage of small datasets, improve regularization, find general design principles as well as customizations to specific problems. There is also an incipient and future opportunity to find synergies between multiple aspects of optimization and learning.

This section reviews the various aspects of evolutionary metalearning. Interestingly, many of the techniques originally developed for evolving entire networks (e.g., those in Sect. 2.2.1) have a new instantiation as metalearning techniques, evolving network architectures synergetically with learning. In addition, new techniques have been developed as well, especially for other aspects of learning system design.

2.4.1 Neural Architecture Search

The most well-studied aspect of metalearning is Neural Architecture Search (NAS), where the network wiring, including potentially overall topology, types of components, modular structure and modules themselves, and general hyperparameters are optimized to maximize performance and other metrics [17, 47]. NAS in general is a productive area in that many approaches have been developed and they work well. However, even random search works well, suggesting that the success has less to do with the methods so far but rather with the fact that architectures matter and many good designs are possible. It also suggests that further improvements are still likely in this area—there is no one dominant approach or solution.

One of the early and most versatile approaches is CoDeepNEAT [44, 52]. This approach builds on several aspects of techniques developed earlier for evolving complete networks. In SANE, ESP, and CoSYNE, partial solutions such as neurons and connections were evolved in separate subpopulations that were then combined into full solutions, i.e., complete neural networks, with the global structure specified, e.g., in terms of a network blueprint that was also evolved [21, 22, 55]. Similarly, CoDeepNEAT co-evolves multiple populations of modules and a population of blueprints that specifies which modules are used and how they are connected to a full network (Fig. 2.6a). Modules are randomly selected from the specified module population to fill in locations in the blueprint. Each blueprint is instantiated in this way many times, evaluating how well the design performs with the current set of blueprints. Each module participates in instantiations of many blueprints (and inherits the fitness of the entire instantiation each time), thus evaluating how well the module works in general with other modules. The main idea of CoDeepNEAT is thus to take advantage of (and scale up with) modular structure, similarly to many current deep learning designs such as the inception network and the residual network [32, 86].

The modules and the blueprints are evolved using NEAT (Sect. 2.2.1), again originally designed to evolve complete networks and adapted in CoDeepNEAT to evolving network structure. NEAT starts with a population of simple structures connecting inputs straight to outputs, and gradually adds more modules in the middle, as well as parallel and recurrent pathways between them. It thus prefers simple solutions but complexifies the module and blueprint structures over time as necessary. It can in principle design rather complex and general network topologies. However, while NEAT can be used to create entire architectures directly, in CoDeepNEAT, it is embedded into the general framework of module and blueprint evolution; it is thus possible to scale up through repetition that would not arise from NEAT naturally.

The power of CoDeepNEAT was originally demonstrated in the task of image captioning, a domain where a competition had been run for several years on a known dataset [52]. The best human design at that point, the Show&Tell network [91], was used to define the search space; that is, CoDeepNEAT was set to find good architectures using the same elements as in the Show&Tell network. Remarkably, CoDeepNEAT was able to improve the performance further by 15%, thus demonstrating the power of metalearning over the best human solutions [52]. Interestingly, the best

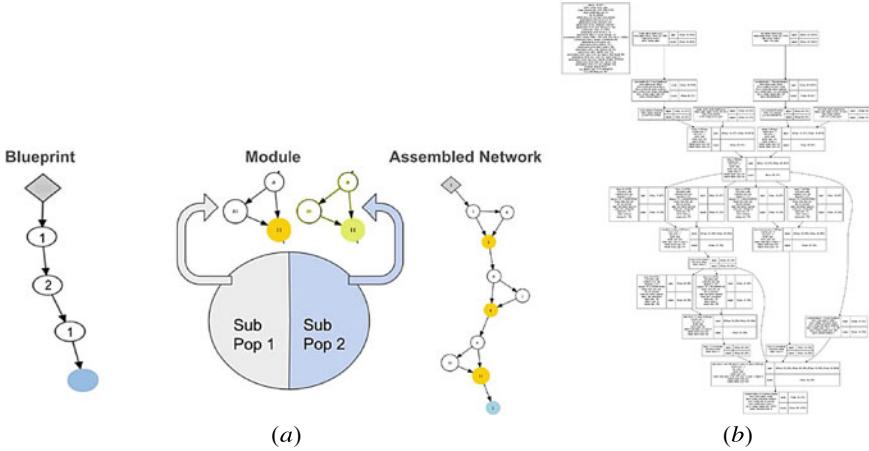


Fig. 2.6 Discovering Complex Neural Architectures through Coevolution of Modules and Blueprints. **a** In CoDeepNEAT [52], the blueprints represent the high-level organization of the network and modules fill in its details. The blueprint and module subpopulations are evolved simultaneously based on how well the entire assembled network performs in the task. This principle was originally developed for evolving entire networks [21, 55], but it applies in neural architecture search for deep learning as well. **b** The overall structure of a network evolved for the image captioning task; the rectangles represent layers, with hyperparameters specified inside each rectangle. One module consisting of two LSTM layers merged by a sum is repeated three times in the middle of the network. The approach allows the discovery of a wide range of network structures. They may take advantage of principles different from those engineered by humans, such as multiple parallel paths brought together in the end of this network. Figures from [52]

networks utilized a principle different from human-designed networks: They included multiple parallel paths, possibly encoding different hypotheses brought together in the end (Fig. 2.6b). In this manner, the large search space utilized by CoDeepNEAT may make it possible to discover new principles of good performance.

Similar CoDeepNEAT evolution from a generic starting point has since then been used to achieve a state of the art in text classification [44] and image classification [44]. The experiments also demonstrated that, with very little computational cost, it is possible to achieve performance that exceeds standard architectures, making it possible to quickly and effectively deploy deep learning to new domains. CoDeepNEAT has since then been extended with mechanisms for multiobjective optimization, demonstrating that the size of the network can be minimized at the same time as its performance. Indeed, size can sometimes be minimized to 1/12 with only a small (0.38%) cost in performance [44]. Another useful extension is to multitask learning by incorporating task routing, i.e., coevolution of task-specific topologies instead of a single blueprint, all constructed from the same module subpopulations [42]. In multitask learning, it is possible to learn accurate performance even with small datasets. Simultaneous learning of multiple datasets utilizes synergies between them, resulting in performance that exceeds learning in each task alone. Multitask evolution finds

architectures that best support such synergies. The approach achieved state of the art in, e.g., multialphabet character recognition [42], as well as multiattribute face classification [51]. These extensions again make it possible to apply deep learning to domains where hardware or data is limited, as it often is in real-world applications.

An important question about evolutionary NAS, and about metalearning in general, is whether it really makes a difference, i.e., results in architectures that advance the state of the art. Such an argument is indeed made below in Sect. 2.4.2.5 wrt synergies of different aspects of metalearning. However, perhaps a most convincing case wrt NAS is the AmoebaNet [64]. At its time, it improved the state of the art in the ImageNet domain, which had been the focus of deep learning research for several years.

There were three innovations that made this result possible. First, search was limited to a NASNet search space, i.e., networks with a fixed outer structure consisting of a stack of inception-like modules (Fig. 2.7a). There were two different module architectures, normal and reduction; they alternate in the stack, and are connected directly and through skip connections. The architecture of the modules is evolved, and consists of five levels of convolution and pooling operations. The idea is that NASNet represents a space of powerful image classifiers that can be searched efficiently. Second, a mechanism was devised that allowed scaling the architectures to much larger numbers of parameters, by scaling the size of the stack and the number of filters in the convolution operators. The idea is to discover good modules first and then increase performance by scaling up. Third, the evolutionary process was modified to favor younger genotypes, by removing those individuals that were evaluated the earliest from the population at each tournament selection. The idea is to allow evolution to explore more instead of focusing on a small number of genotypes early on. Each of these ideas is useful in general in evolutionary ML, not just as part of the AmoebaNet system.

Indeed, AmoebaNet’s accuracy was the state of the art in the ImageNet benchmark at the time, which is remarkable given how much effort the entire AI community had spent on this benchmark. Experiments also demonstrated that evolutionary search in NASNet was more powerful than reinforcement learning and random search in CIFAR-10, resulting in faster learning, more accurate final architectures, and ones with lower computational cost (Fig. 2.7b). It also demonstrates the value of focusing the search space intelligently so that good solutions are in that space, yet it is not too large to find them.

Evolutionary NAS is rapidly becoming a field of its own, with several new approaches proposed recently. They include multiobjective and surrogate-based approaches [49], Cartesian genetic programming [83, 92], variable-length encodings and smart initialization [84, 85], and LSTM design [63], to name a few; see Chap. X for a more detailed review. The work is expanding from convolutional networks to transformers and diffusion networks, and from visual and language domains to tabular data. An important direction is also to optimize design aspects other than the architecture, as will be reviewed next.

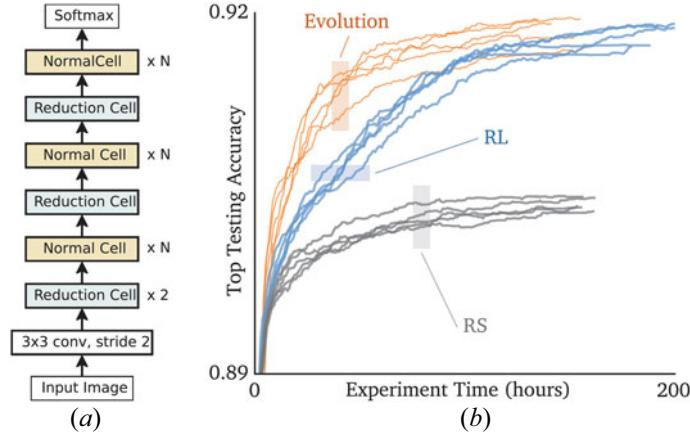


Fig. 2.7 Evolutionary Discovery in the NASNet Search Space Compared RL and Random Search. **a** The AmoebaNet method [64] focuses evolution to a particular stacked architecture of inception-like normal and reduction modules (cells); these networks are then scaled to larger sizes algorithmically. AmoebaNet also promotes regularization by removing the oldest individuals in the population. **b** As a result, it discovers architectures that are more accurate than those discovered through random search and RL, reaching state-of-the-art accuracy in standard benchmarks like ImageNet. Figures from [64]

2.4.2 Beyond Architecture Search

Besides the architecture, there are several other aspects of machine learning system design that need to be configured properly for the system to perform well. Those include learning hyperparameters (such as the learning rate), activation functions, loss functions, data sampling and augmentation, and the methods themselves. Approaches similar to those used in NAS can be applied to them; however, the evolutionary approach has an advantage in that it is the most versatile: It can be applied to graphs, vectors of continuous and discrete parameters, and configuration choices. This ability is particularly useful as new architectures are developed. For instance, at this writing, work has barely begun on optimizing designs of transformer [90] or diffusion [75] architectures. They have elements such as attention modules, spatial embeddings, and noise transformations that are different from prior architectures, yet may be parameterized and evolution applied to optimize their implementation. Most importantly, evolution can be used to optimize many different aspects of the design at the same time, discovering and taking advantage of synergies between them. Several such approaches are reviewed in this section.

2.4.2.1 Loss Functions

Perhaps the most fundamental is the design of a good loss function. The mean-squared-error (MSE) loss has been used for a long time, and more recently, the cross-entropy (CE) loss has become popular, especially in classification tasks. Both of those assign minimal loss to outputs that are close to correct, and superlinearly larger losses to outputs further away from correct values. They make sense intuitively and work reliably, so much so that alternatives are not usually even considered.

However, it turns out that it is possible to improve upon them, in a surprising way that would have been difficult to discover if evolution had not done it for us [25, 27]. If outputs that are extremely close to correct are penalized with a larger loss, the system learns to avoid such extreme outputs—which minimizes overfitting (Fig. 2.8a). Such loss functions, called Baikal loss for their shape, lead to automatic regularization. Regularization in turn leads to more accurate performance on unseen examples, especially in domains where the amount of available data is limited, as is the case in many real-world applications.

Baikal loss was originally discovered with a classic genetic programming approach where the function was represented as a tree of mathematical operations [25]. The structure of the tree was evolved with genetic algorithms and the coefficients in the nodes with CMA-ES [29]. This approach is general and creative in that it can be

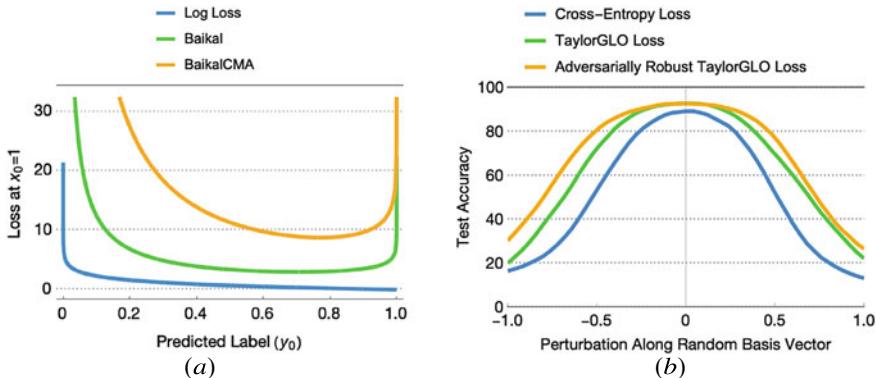


Fig. 2.8 Regularization and Robustness with Evolved Loss Functions. **a** The standard loss function, such as log loss (or cross-entropy) has a high loss for outputs that are far from correct (1.0 in this case) and a low loss otherwise. In contrast, evolutionary optimization of loss functions through GLO/TaylorGLO [25, 27] discovered a new principle: When the output is very close to the correct one, a high loss is incurred. This principle, termed Baikal loss for its shape, discourages overfitting, thus regularizing the network automatically, leading to better generalization. Such a loss is effective but counterintuitive, and thus unlikely to be discovered by human designers. **b** The Baikal loss also makes the network performance more robust. This effect can be quantified by perturbing the network weights. With Baikal loss, the network's performance is less affected than with cross-entropy loss. This effect can be further magnified by making robustness against adversarial inputs an explicit second objective in evolution. Thus, loss-function optimization can be used to improve not only regularization but also robustness. Figures from [25, 26]

used to explore a large search space of diverse functions. However, many of those functions do not work well and often are not even stable. In the follow-up TaylorGLO method [27], the functions were represented instead as third-order Taylor polynomials. Such functions are continuous and can be directly optimized with CMA-ES, making the search more effective.

Regularization in general is an important aspect of neural network design, there are many techniques available, such as dropout, weight decay, and label smoothing [30, 77, 87], but how they work is not well understood. Loss-function optimization, however, can be understood theoretically, and thus provides a starting point to understanding regularization in general [26]. It can be described as a balance of two processes, one a pull toward the training targets, and another a push away from overfitting. The theory leads to a practical condition for guiding the search toward trainable functions.

Note that Baikal loss is a general principle; evolutionary optimization was crucial in discovering it but it can now be used on its own in deep learning. It is still possible to customize it for each task and architecture, and even small modifications to the standard Baikal shape may make a difference. Optimization may also have a significant effect on various learning challenges, for instance, when there is not much training data [24], or when the labels are particularly noisy [20]. It may also be possible to modify the loss function during the course of learning, for instance by emphasizing regularization in the beginning and precision toward the end (similarly to activation functions; Sect. 2.4.2.2).

It turns out that loss functions that regularize also make networks more robust, and this effect can be further enhanced by including an explicit robustness goal in evolution (Fig. 2.8b). One way to create such a goal is to evaluate performance separately wrt adversarial examples. This result in turn suggests that loss-function optimization could be an effective approach to creating machine learning systems that are robust against adversarial attacks.

Loss-function optimization can also play a major role in systems where multiple loss functions interact, such as Generative Adversarial Networks (GANs; [23]). GANs include three different losses: discriminative loss for real examples and for fake examples, and the generative loss (for fake examples). It is difficult to get them right, and many proposals exist, including those in minimax, nonsaturating, Wasserstein, and least-squares GANs [3, 28, 50]. Training often fails, resulting, e.g., in mode collapse. However, the three losses can be evolved simultaneously, using performance and reliability as fitness. In one such experiment on generating building facade images given the overall design as a condition, the TaylorGLO approach was found to result in better structural similarity and perceptual distance than the Wasserstein loss [23]. Although this result is preliminary, it suggests that evolutionary loss-function optimization may make more complex learning systems possible in the future.

2.4.2.2 Activation Functions

Early on in the 1980s and 1990s, sigmoids (and tanh) were used almost exclusively as activation functions for neural networks. They had the intuitively the right behavior as neural models, limiting activation between the minimum and maximum values, a simple derivative that made backpropagation convenient, and a theorem suggesting that universal computing could be based on such networks [12, 36]. There were indications, however, that other activation functions might work better in many cases. Gaussians achieved universal computing with one less layer and were found powerful in radial basis function networks [59]. Ridge activations were also found to provide similar capabilities [46].

However, with the advent of deep learning, an important discovery was made: Activation function actually made a big difference wrt vanishing gradients. In particular, rectified linear units (ReLUs), turned out important in scaling up deep learning networks [56]. The linearly increasing region does not saturate activation or gradients, resulting in less signal loss. Moreover, it turned out that in many cases ReLU could be improved by adding a small differentiable dip at the boundary between the two regions, in a function called Swish [62]. This result suggested that there may be an opportunity to optimize activation functions, in general, and for specific architectures and tasks.

Like with loss functions, there is a straightforward opportunity in evolving functions through genetic programming [6]. Similarly to loss functions, such an approach can be creative, but also results in many functions that make the network unstable. A more practical approach is to limit the search space to, e.g., computation graphs of two levels, with a focused set of operators, that are more likely to result in useful functions. This approach was taken, e.g., in the Pangaea system [7]. Given a list of 27 unary and 7 binary operators, 2 basic two-level computation graph structures, and 4 mutation operators, evolution can search a space of over 10 trillion activation functions.

However, finding an effective function is only part of the challenge. The function also needs to be parameterized so that it performs as well as possible. While coefficients multiplying each operator can be evolved together with the structure, it turns out that such fine tuning can be done more efficiently through gradient descent. In other words, in Pangaea evolution and gradient descent work synergetically: evolution discovers the general structure of the function, and gradient descent finds its optimal instantiation.

The method is powerful in two ways: it finds general functions that perform better than previous functions (such as ReLU, SeLU, Swish, etc.) across architectures (such as All-CNN, Wide ResNet, Resnet, and Preactivation Resnet) and tasks (such as CIFAR-10, CIFAR-100). However, it is most powerful in discovering activation functions that are specialized in architecture and task, apparently taking advantage of the special requirements in each such context.

Furthermore, performance can be further improved by allowing different functions at different parts of the network, and at different times throughout training (Fig. 2.9). The optimal designs change continuously over time and space. Differ-

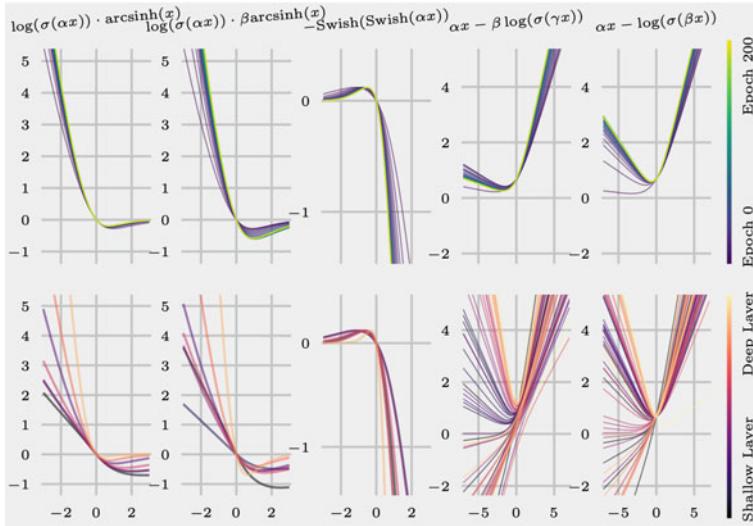


Fig. 2.9 Activation Functions Discovered over Space and Time. Pangaea [7] combines evolution of function structure synergetically with gradient descent of its parameters. It is possible to discover general functions, but the approach is most powerful in customizing them to a particular architecture and task. Moreover, the functions change systematically over learning time as well as through different depths of layers, presumably starting with coarse learning and regularization and transforming into fine-tuning and classification. These results suggest a possible duality with weight learning and a possible synergy for the future. Figure from [7]

ent activation functions are useful early in training when the network learns rapidly and late in training when fine-tuning is needed; similarly, more nonlinear functions are discovered for later layers, possibly reflecting the need to form a regularized embedding early, and make classification decisions later.

The Pangaea results suggest an intriguing duality: While neural network learning is mostly based on adapting a large number of parameters (i.e., weights), perhaps a similar effect might be achieved by adapting the activation functions over space and time? Perhaps the two mechanisms could be used synergetically? Evolution of the activation function structure provides the foundation for this approach, which still needs to be developed fully.

2.4.2.3 Data Use and Augmentation

Another important opportunity for evolutionary optimization of supervised learning systems is to optimize the training data. For instance, it may be possible to form embeddings of the training samples through an autoencoder, and then form a strategy for utilizing different kinds of samples optimally through time [24]. In this manner, evolution could discover ways for balancing an imbalanced dataset or designing

curricular learning from simple to more complex examples. Especially in domains where not a lot of labeled samples are available, such techniques could result in significant improvements. It may also be possible to extend the methods to utilize multiple datasets optimally over time in a multitask setting.

Another possibility is to evolve methods for augmenting the available data automatically through various transformations. Different datasets may benefit from different transformations, and it is not always obvious ahead of time how they should be designed. For instance, in an application to develop models for estimating the age of a person from an image of their face, evolution was used to decide vertical and horizontal shift and cutout, as well as a direction of flip operations, angle of rotation, degree of zoom, and extent of shear [53]. Unexpectedly, it chose to do vertical flips only—which made little sense for faces, until it was found that the input images had been rotated 90 degrees. It also discovered a combination of shift operations that allowed it to obfuscate the forehead and chin, which would otherwise be easy areas for the model to overfit.

A particularly interesting use for evolved data augmentation is to optimize not only the accuracy of the resulting models but also to mitigate bias and fairness issues with the data. As long as these dimensions can be measured [72], they can be made part of the fitness, or separate objectives in a multiobjective setting. Operations then need to be designed to increase variance across variables that might otherwise lead to bias through overfitting—for instance gender, ethnicity, and socioeconomic status, depending on the application. While evolutionary data augmentation is still new, this area seems like a differentiated and compelling opportunity for it.

2.4.2.4 Learning Methods

An interesting extension of NAS is to evolve the learning system not from high-level elements, but from the basic algorithmic building blocks (mathematical operations, data management, and ways to combine them)—in other words, by evolving code for supervised machine learning. In this manner, evolution can be more creative in discovering good methods, with fewer biases from the human experimenters.

The AutoML-Zero system [65] is a step toward this goal. Given an address space for scalars, vectors, and matrices of floats, it evolves setup, predicts, and learns methods composed of over 50 basic mathematical operations. Evolution is implemented as a linear GP and consists of inserting and removing instructions and randomizing instructions and addresses. Evaluation consists of computing predictions over unseen examples.

Starting from empty programs, AutoML-Zero first discovered linear models, followed by gradient descent, and eventually several extensions known in the literature, such as noisy inputs, gradient normalization, and multiplicative interactions (Fig. 2.10). When given small datasets, it discovers regularization methods similar to dropout; when given a few training steps, it discovers learning-rate decay.

Thus, the preliminary experiments with AutoML-Zero suggest that evolutionary search can be a powerful tool in discovering entire learning algorithms. As in many

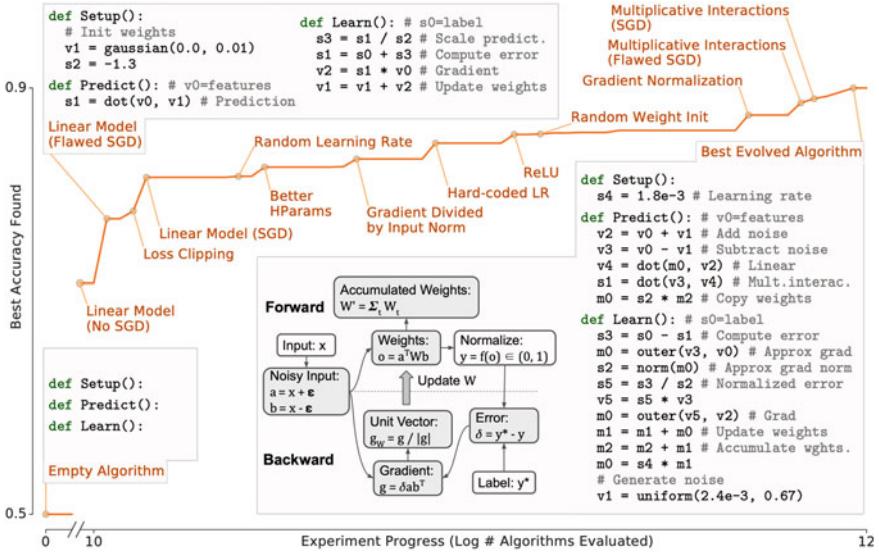


Fig. 2.10 Evolutionary Discovery of Learning Methods. In AutoML-Zero [65], sequences of instructions for setup, prediction, and learning are evolved through mutation-based regularized search. AutoML-Zero first discovered simple methods such as linear models, then several known extensions such as ReLU and gradient normalization, and eventually more sophisticated techniques such as multiplicative interactions. The approach could potentially be useful in particular in customizing learning methods to different domains and constraints. Figure from [65]

metalearning approaches, the main power may be in customizing these methods to particular domains and constraints. A crucial aspect will be to guide the evolution within the enormous search space toward meaningful solutions, without hampering its ability to create, again a challenge shared with most of metalearning.

2.4.2.5 Synergies

Perhaps the most important future direction in evolutionary metalearning is to discover and utilize synergies between the different aspects of the learning system design. For instance, the best performance was reached by optimization activation functions for the specific architecture; it might be possible to optimize the architecture simultaneously to emphasize this effect.

Simply running evolution on all these design aspects simultaneously is unlikely to work; the search space would be prohibitively large. Similarly, adding more outer loops to the existing process (where supervised learning is the inner loop and metalearning is the outer loop) is likely prohibitive as well. However, it might be possible to alternate evolution of different aspects. Better yet, techniques from bilevel (or

multilevel) optimization could be useful—the idea is to avoid full inner–outer loop structure, but instead use, e.g., surrogate models to evaluate outer loop innovations [45, 74].

A practical approach is to simply add constraints, and search in a smaller space. A first such step was already taken in the EPBT system [43], which combines hyperparameter tuning, loss-function optimization, and population-based training (PBT) into a single loop. That is, hyperparameters and loss functions are evolved at the same time as the networks are being trained. Hyperparameter tuning is limited to those that do not change the structure of the networks (e.g., learning rate schedules) so that they can be continuously trained, even when the hyperparameters change. Similarly, loss-function optimization is limited to TaylorGLO coefficients [43] that can be changed while training is going on. Even so, the simultaneous evolution and learning was deceptive, and needed to be augmented with two mechanisms: quality–diversity heuristic for managing the population and knowledge distillation to prevent overfitting. The resulting method not only worked well on optimizing ResNet and WideResnet architectures in CIFAR-10 and SVHN but also illustrated the challenges in taking advantage of synergies of metalearning methods.

Similarly, promising results were obtained in an experiment that compared human design with evolutionary metalearning [53]. Using the same datasets and initial model architectures, similar computational resources, and similar development time, a team of data scientists and an evolutionary metalearning approach developed models for age estimation in facial images (Fig. 2.11). The evolutionary metalearning approach, LEAF-ENN, included optimization of loss functions (limited to linear combinations of MSE and CE), learning hyperparameters, architecture hyperparameters, and data augmentation methods. Evolution discovered several useful principles that the data scientists were not aware of focusing data augmentation to regions that mattered most, and utilizing flips only horizontally across the face; utilizing different loss functions at different times during learning; relying mostly on the output level blocks of the base models. With both datasets, the eventual accuracy of the metalearned models was significantly better than that of the models developed by the data scientists. This result demonstrates the main value of evolutionary metalearning: it can result in models that are optimized beyond human ability to do so.

2.5 Conclusion

Although much of evolutionary machine learning has focused on discovering optimal designs and behavior, it is a powerful approach to supervised learning as well. While gradient-based supervised learning (i.e., deep learning) has benefited from massive scale-up, three opportunities for evolutionary supervised learning have emerged as well. In particular, in domains where such a scale-up is not possible, it can be useful in two ways. First, it can expand the scope of supervised learning to a more general set of design goals other than simply accuracy. Second, it can be applied to solution structures that are explainable. Third, in domains where deep learning is applicable,

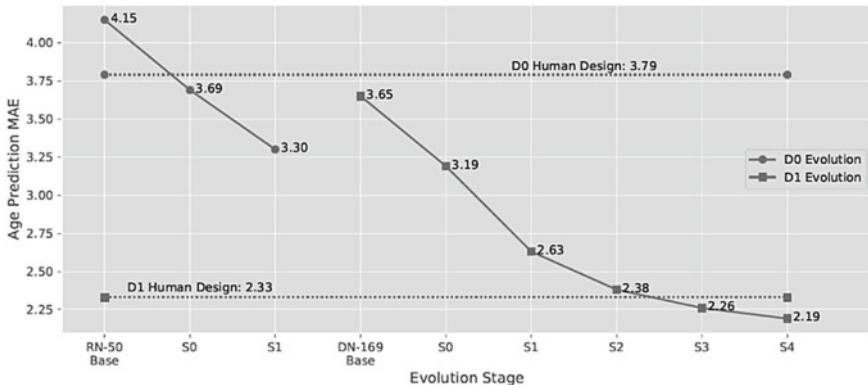


Fig. 2.11 Utilizing Metalearning Synergies to Beat Human Designers. In two datasets (D0 and D1) for age estimation from facial images, LEAF-ENN evolutionary metalearning [53] was able to discover models that performed better than those simultaneously designed by human experts. The humans optimized the ResNet-50 architecture for D0 and EfficientNet-B8 for D1. The evolutionary runs progressed in stages: In D0, ResNet-50 (S0) was expanded to DenseNet 169 (S1); in D1, DenseNet-169 (S0) was expanded to DenseNet-201 (S1) and trained longer (S2), then expanded to EfficientNet-B6 (S3), and ensembling (S4). At the same time, evolution-optimized learning and architecture hyperparameters, data-augmentation methods, and combinations of loss functions. The approach discovers and utilizes synergies between design aspects that are difficult for humans to utilize. The final accuracy, MSE of 2.19 years, is better than typical human accuracy in age estimation (3–4 years). Figure from [53]

it can be used to optimize the design of the most complex such architectures, thus improving upon the state of the art. Two most interesting research directions emerge: how to extend the generality and explainability to larger domains, and how to take advantage of synergies between multiple aspects of machine learning system design. Such work most likely requires developing a better understanding of how the search can be guided to desired directions without limiting the creativity of the approach.

References

1. Aharonov-Barki, R., Beker, T., Ruppin, E.: Emergence of memory-driven command neurons in evolved artificial agents. *Neural Comput.* **13**, 691–716 (2001)
2. Aitkenhead, M.J.: A co-evolving decision tree classification method. *Expert Syst. Appl.* **34**, 19–25 (2008)
3. Arjovsky, M., Chintala, S., Bottou, L.: Wasserstein generative adversarial networks. In: Precup, D., Teh, Y.W. (eds.), *Proceedings of the 34th International Conference on Machine Learning*, vol. 70, pp. 214–223 (2017)
4. Barros, R.C., Basgalupp, M.P., De Carvalho, A.C., Freitas, A.A.: A survey of evolutionary algorithms for decision-tree induction. *IEEE Trans. Syst., Man, Cybern., Part C (Appl. Rev.)* **42**, 291–312 (2012)

5. Bi, Y., Xue, B., Zhang, M.: Genetic programming-based evolutionary deep learning for data-efficient image classification. *IEEE Trans. Evolut. Comput.* (2022). <https://doi.org/10.1109/TEVC.2022.3214503>
6. Bingham, G., Macke, W., Miikkulainen, R.: Evolutionary optimization of deep learning activation functions. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 289–296 (2020)
7. Bingham, G., Miikkulainen, R.: Discovering parametric activation functions. *Neural Netw.* **148**, 48–65 (2022)
8. Breiman, L.: Random forests. *Mach. Learn.* **45**, 5–32 (2001)
9. Breiman, L., Friedman, J., Stone, C.J., Olshen, R.A.: Classification and Regression Trees. Chapman and Hall/CRC (1984)
10. Butz, M.V., Lanzi, P.L., Wilson, S.W.: Function approximation with xcs: Hyperellipsoidal conditions, recursive least squares, and compaction. *IEEE Trans. Evolut. Comput.* **12**, 355–376 (2008)
11. Canatar, A., Bordelon, B., Pehlevan, C.: Spectral bias and task-model alignment explain generalization in kernel regression and infinitely wide neural networks. *Nat. Commun.* **12**, 1914 (2021)
12. Cybenko, G.: Approximation by superpositions of a sigmoidal function. *Math. Control Signals Syst.* **2**, 303–314 (1989)
13. Dai, E., Zhao, T., Zhu, H., Xu, J., Guo, Z., Liu, H., Tang, J., Wang, S.: A comprehensive survey on trustworthy graph neural networks: privacy, robustness, fairness, and explainability. [arXiv:2104.05605](https://arxiv.org/abs/2104.05605), 2020
14. De Jong, K.: Learning with genetic algorithms: an overview. *Mach. Learn.* **3**, 121–138 10 (1988)
15. Deb, K., Myburgh, C.: A population-based fast algorithm for a billion-dimensional resource allocation problem with integer variables. *Eur. J. Oper. Res.* **261**, 460–474 (2017)
16. Dolotov, E., Zolotykh, N.Y.: Evolutionary algorithms for constructing an ensemble of decision trees (2020). [arXiv:2002.00721](https://arxiv.org/abs/2002.00721)
17. Elskens, T., Metzen, J.H., Hutter, F.: Neural architecture search: a survey. *J. Mach. Learn. Res.* **20**, 1–21 (2019)
18. Gaier, A., Ha, D.: Weight agnostic neural networks. In: Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., Garnett, R. (eds.), *Advances in Neural Information Processing Systems* 5364–5378 (2019)
19. Ganon, Z., Keinan, A., Ruppin, E.: Evolutionary network minimization: adaptive implicit pruning of successful agents. In: Banzhaf, W., Ziegler, J., Christaller, T., Dittrich, P., Kim, J.T. (eds.) *Advances in Artificial Life*, pp. 319–327. Springer, Berlin (2003)
20. Gao, B., Gouk, H., Hospedales, T.M.: Searching for robustness: loss learning for noisy classification tasks. *IEEE/CVF International Conference on Computer Vision*, pp. 6650–6659 (2021)
21. Gomez, F., Miikkulainen, R.: Incremental evolution of complex general behavior. *Adapt. Behav.* **5**, 317–342 (1997)
22. Gomez, F., Schmidhuber, J., Miikkulainen, R. and Mitchell, M.: Accelerated neural evolution through cooperatively coevolved synapses. *J. Mach. Learn. Res.* 937–965 (2008)
23. Gonzalez, S., Kant, M., Miikkulainen, R.: Evolving GAN formulations for higher quality image synthesis. In: Kozma, R., Alippi, C., Choe, Y., Morabito, F.C. (eds.) *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, 2nd edn. Elsevier, New York (2023)
24. Gonzalez, S., Landgraf, J., Miikkulainen, R.: Faster training by selecting samples using embeddings. In: Proceedings of the 2019 International Joint Conference on Neural Networks, pp. 1–7 (2019)
25. Gonzalez, S., Miikkulainen, R.: Improved training speed, accuracy, and data utilization through loss function optimization. In: Proceedings of the 2020 IEEE Congress on Evolutionary Computation (CEC), pp. 1–8 (2020)
26. Gonzalez, S., Miikkulainen, R.: Effective regularization through loss-function metalearning (2021). [arXiv:2010.00788](https://arxiv.org/abs/2010.00788)

27. Gonzalez, S., Miikkulainen, R.: Optimizing loss functions through multivariate Taylor polynomial parameterization. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 305–313 (2021)
28. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y.: Generative adversarial nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., Weinberger, K.Q., (eds.), Advances in Neural Information Processing Systems 27, pp. 2672–2680 (2014)
29. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. *Evol. Comput.* **9**, 159–195 (2001)
30. Hanson, S.J., Pratt, L.Y.: Comparing biases for minimal network construction with back-propagation. In: Proceedings of the 1st International Conference on Neural Information Processing Systems, pp. 177–185. MIT Press, Cambridge (1988)
31. Hayes-Roth, F.: Rule-based systems. *Commun. ACM* **28**, 921–932 (1985)
32. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
33. Hemberg, E., Veeramachaneni, K., Wanigarekara, P., Shahrzad, H., Hodjat, B., O'Reilly, U.-M.: Learning decision lists with lagged physiological time series. In: Workshop on Data Mining for Medicine and Healthcare, 14th SIAM International Conference on Data Mining, pp. 82–87 (2014)
34. Holland, J.H.: Escaping brittleness: the possibilities of general purpose learning algorithms applied to parallel rule-based systems. In: Michalski, R.S., Carbonell, J.G., Mitchell, T.M. (eds.) Machine Learning: An Artificial Intelligence Approach, vol. 2, pp. 593–623. Morgan Kaufmann, Los Altos (1986)
35. Hoos, H.: Programming by optimization. *Commun. ACM* **55**, 70–80 (2012)
36. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. *Neural Netw.* **2**, 359–366 (1989)
37. Huang, X., Kroening, D., Ruan, W., Sharp, J., Sun, Y., Thamo, E., Min, W., Yi, X.: A survey of safety and trustworthiness of deep neural networks: verification, testing, adversarial attack and defence, and interpretability. *Comput. Sci. Rev.* **37**, 100270 (2020)
38. Ijspeert, A.J.: Central pattern generators for locomotion control in animals and robots: A review. *Neural Netw.* **21**, 642–653 (2008)
39. Jankowski, D., Jackowski, K.: Evolutionary algorithm for decision tree induction. In: Saeed, K., Snášel, V. (eds.) Computer Information Systems and Industrial Management, pp. 23–32. Springer, Berlin (2014)
40. Kashtan, N., Alon, U.: Spontaneous evolution of modularity and network motifs. *Proc. Natl. Acad. Sci.* **102**, 13773–13778 (2005)
41. Langdon, W.B., Poli, R., McPhee, N.F., Koza, J.R.: Genetic programming: An introduction and tutorial, with a survey of techniques and applications. In: Fulcher, J., Jain, L.C. (eds.) Computational Intelligence: A Compendium, pp. 927–1028. Springer, Berlin (2008)
42. Liang, J., Meyerson, E. and Miikkulainen, R.: Evolutionary architecture search for deep multi-task networks. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 466–473 (2018)
43. Liang, J., Gonzalez, S., Shahrzad, H., Miikkulainen, R.: Regularized evolutionary population-based training. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 323–331 (2021)
44. Liang, J., Meyerson, E., Hodjat, B., Fink, D., Mutch, K. and Miikkulainen, R.: Evolutionary neural AutoML for deep learning. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2019), pp. 401–409 (2019)
45. Liang, J.Z., Miikkulainen, R.: Evolutionary bilevel optimization for complex control tasks. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2015), pp. 871–878 (2015)
46. Light, W.: Ridge functions, sigmoidal functions and neural networks. In: Approximation Theory VII, pp. 158–201. Academic, Boston (1992)

47. Liu, Y., Sun, Y., Xue, B., Zhang, M., Yen, G.G., Tan, K.C.: A survey on evolutionary neural architecture search. *IEEE Trans. Neural Netw. Learn. Syst.* 1–21 (2021)
48. Liu, Z., Zhang, X., Wang, S., Ma, S., Gao, W.: Evolutionary quantization of neural networks with mixed-precision. In: *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2785–2789 (2021)
49. Lu, Z., Deb, K., Goodman, E., Banzhaf, W., Boddeti, V.N.: Nsganetv2: evolutionary multi-objective surrogate-assisted neural architecture search. In: *European Conference on Computer Vision ECCV-2020*, LNCS, vol. 12346, pp. 35–51 (2020)
50. Mao, X., Li, Q., Xie, H., Lau, R.Y., Wang, Z., Paul Smolley, S.: Least squares generative adversarial networks. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 2794–2802 (2017)
51. Meyerson, E., Miikkulainen, R.: Pseudo-task augmentation: from deep multitask learning to intratask sharing—and back. In: *Proceedings of the 35th International Conference on Machine Learning*, pp. 739–748 (2018)
52. Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Duffy, N., Hodjat, B.: Evolving deep neural networks. In: Morabito, C.F., Alippi, C., Choe, Y., Kozma, R. (eds.) *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, 2nd edn., pp. 293–312. Elsevier, New York (2023)
53. Miikkulainen, R., Meyerson, E., Qiu, X., Sinha, U., Kumar, R., Hofmann, K., Yan, Y.M., Ye, M., Yang, J., Caiazza, D. and Brown, S.M.: Evaluating medical aesthetics treatments through evolved age-estimation models. In: *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1009–1017 (2021)
54. Montana, D.J., Davis, L.: Training feedforward neural networks using genetic algorithms. In: *International Joint Conference on Artificial Intelligence*, pp. 762–767 (1989)
55. Moriarty, D.E., Miikkulainen, R.: Forming neural networks through efficient and adaptive co-evolution. *Evol. Comput.* **5**, 373–399 (1997)
56. Nair, V., Hinton, G.E.: Rectified linear units improve restricted boltzmann machines. In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 807–814 (2010)
57. Oymak, S.: Learning compact neural networks with regularization. In: *International Conference on Machine Learning*, pp. 3963–3972 (2018)
58. Papavasileiou, E., Cornelis, J., Jansen, B.: A systematic literature review of the successors of “neuroevolution of augmenting topologies”. *Evol. Comput.* **29**, 1–73 (2021)
59. Park, J., Sandberg, I.W.: Universal approximation using radial-basis-function networks. *Neural Comput.* **3**, 246–257 (1991)
60. Quinlan, J.R.: Induction of decision trees. *Mach. Learn.* **1**, 81–106 (1986)
61. Quinlan, J.R.: *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco (1993)
62. Ramachandran, P., Zoph, B., Le, Q.V.: Searching for activation functions (2017). [arXiv:1710.05941](https://arxiv.org/abs/1710.05941)
63. Rawal, A., Miikkulainen, R.: Discovering gated recurrent neural network architectures. In: Iba, H., Noman, N., (eds.), *Deep Neural Evolution - Deep Learning with Evolutionary Computation*, pp. 233–251. Springer (2020)
64. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. In: *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 4780–4789 (2019)
65. Real, E., Liang, C., So, D., Le, Q.: AutoML-Zero: evolving machine learning algorithms from scratch. In: Daumé III, H., Singh, A. (eds.), *Proceedings of the 37th International Conference on Machine Learning*, pp. 8007–8019 (2020)
66. Reed, R.: Pruning algorithms-a survey. *IEEE Trans. Neural Netw.* **4**, 740–747 (1993)
67. Routley, N.: Visualizing the trillion-fold increase in computing power. Retrieved 11/17/2022 (2017)
68. Schaul, T., Schmidhuber, J.: Metalearning. *Scholarpedia* **5**, 4650 (2010)
69. Schmidhuber, J.: Annotated history of modern ai and deep learning (2022). [arXiv:2211.21127](https://arxiv.org/abs/2211.21127)

70. Shahrzad, H., Hodjat, B., Dolle, C., Denissov, A., Lau, S., Goodhew, D., Dyer, J., Miikkulainen, R.: Enhanced optimization with composite objectives and novelty pulsation. In: Banzhaf, W., Goodman, E., Sheneman, L., Trujillo, L., Worzel, B. (eds.), *Genetic Programming Theory and Practice XVII*, pp. 275–293. Springer, New York (2020)
71. Shahrzad, H., Hodjat, B., Miikkulainen, R.: EVOTER: evolution of transparent explainable rule-sets (2022). [arXiv:2204.10438](https://arxiv.org/abs/2204.10438)
72. Sharma, S., Henderson, J., Ghosh, J.: CERTIFAI: a common framework to provide explanations and analyse the fairness and robustness of black-box models. In: Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society, pp. 166–172, New York 2020. Association for Computing Machinery (2020)
73. Shayani, H., Bentley, P.J., Tyrrell, A.M.: An fpga-based model suitable for evolution and development of spiking neural networks. In: Proceedings of the European Symposium on Artificial Neural Networks, pp. 197–202 (2008)
74. Sinha, A., Malo, P., Xu, P. and Deb, K.: A bilevel optimization approach to automated parameter tuning. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2014), pp. 847–854, Vancouver, BC, Canada (2014)
75. Sohl-Dickstein, J., Weiss, E., Maheswaranathan, N., Ganguli, S.: Deep unsupervised learning using nonequilibrium thermodynamics. In: Proceedings of the 32nd International Conference on Machine Learning, vol. 37, pp. 2256–2265 (2015)
76. Srinivasan, S., Ramakrishnan, S.: Evolutionary multi objective optimization for rule mining: a review. *Artif. Intell. Rev.* **36**, 205–248, 10 (2011)
77. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **15**, 1929–1958 (2014)
78. Stanley, K.O.: Efficient Evolution of Neural Networks Through Complexification. PhD thesis, Department of Computer Sciences, The University of Texas at Austin (2004)
79. Stanley, K.O., D’Ambrosio, D.B., Gauci, J.: A hypercube-based encoding for evolving large-scale neural networks. *Artif. Life* **15**, 185–212 (2009)
80. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. *Evolut. Comput.* **10**, 99–127 (2002)
81. Stanley, K.O., Miikkulainen, R.: Competitive coevolution through evolutionary complexification. *J. Artif. Intell. Res.* **21**, 63–100 (2004)
82. Such, F.P., Madhavan, V., Conti, E., Lehman, J., Stanley, K.O. and Clune, J.: Deep neuroevolution: genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning (2017). [arXiv:1712.06567](https://arxiv.org/abs/1712.06567)
83. Suganuma, M., Kobayashi, M., Shirakawa, S., Nagao, T.: Evolution of deep convolutional neural networks using cartesian genetic programming. *Evol. Comput.* **28**, 141–163 (2020)
84. Sun, Y., Xue, B., Zhang, M., Yen, G.G.: Evolving deep convolutional neural networks for image classification. *IEEE Trans. Evol. Comput.* **24**, 394–407 (2020)
85. Sun, Y., Xue, B., Zhang, M., Yen, G.G., Lv, J.: Automatically designing cnn architectures using the genetic algorithm for image classification. *IEEE Trans. Cybern.* **50**, 3840–3854 (2020)
86. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1–9 (2015)
87. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2818–2826 (2016)
88. Urbanowicz, R., Moore, J.: Learning classifier systems: a complete introduction, review, and roadmap. *J. Artif. Evolut. Appl.* **2009**, 736398 (2009)
89. Urbanowicz, R.J., Bertasius, G. and Moore, J.H.: An extended michigan-style learning classifier system for flexible supervised learning, classification, and data mining. In: International Conference on Parallel Problem Solving from Nature, pp. 211–221. Springer (2014)
90. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Guyon, I., Von Luxburg, U., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R., (eds.), *Advances in Neural Information Processing Systems*, vol. 30, pp. 6000–6010 (2017)

91. Vinyals, O., Toshev, A., Bengio, S. and Erhan, D.: Show and tell: a neural image caption generator. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3156–3164 (2015)
92. Wu, X., Jia, L., Zhang, X., Chen, L., Liang, Y., Zhou, Y., Wu, C.: Neural architecture search based on cartesian genetic programming coding method (2021). [arXiv:2103.07173](https://arxiv.org/abs/2103.07173)

Chapter 3

EML for Unsupervised Learning



Roberto Santana

Abstract This chapter introduces the use of Evolutionary Machine Learning (EML) techniques for unsupervised machine learning tasks. First, a brief introduction to the main concepts related to unsupervised Machine Learning (ML) is presented. Then, an overview of the main EML approaches to these tasks is given together with a discussion of the main achievements and current challenges in addressing these tasks. We focus on four commonly found unsupervised learning tasks: Data preparation, Outlier detection, Dimensionality reduction, and Association rule mining. Finally, we present a number of findings from the review. These findings could guide the reader at the time of applying EML techniques to unsupervised ML tasks or when developing new EML approaches.

3.1 Introduction

While supervised Machine Learning (ML) algorithms require human intervention to label the examples that are used for learning, unsupervised ML methods do not need tagged examples as a guidance. These algorithms can be used to group instances that share common patterns, to reduce the problem dimensionality or identify regularities in the data, to detect outliers, and for other multiple tasks that are relevant in real-world applications.

Since the availability of labeled data is scarce in many practical problems, unsupervised ML tasks are frequent in a variety of application domains such as computer vision [33, 61], natural language processing [10, 89], physics [6, 57], and other fields [12, 53, 80]. In many areas, unsupervised learning is important also from a conceptual or theoretical point of view since unsupervised approaches can provide clues about the underlying structure of the problem or reveal the existence of anomalies or unusual behaviors in a system.

R. Santana (✉)

University of the Basque Country (UPV/EHU), San Sebastian, Gipuzkoa, Spain
e-mail: roberto.santana@ehu.eus

As in other areas of ML, evolutionary algorithms have been applied to solve unsupervised ML tasks, and have been also hybridized with classical unsupervised ML algorithms. In this chapter, we discuss the main achievements of Evolutionary Machine Learning (EML) in unsupervised learning tasks. We also present some of the current challenges in EML for this type of problem.

The best known unsupervised learning task is perhaps clustering. EML for clustering is covered in Chap. 5. Therefore, the focus of this chapter is on a set of other four commonly found unsupervised learning tasks: Data preparation, Outlier detection, Dimensionality reduction, and Association rule mining. Certainly, there are other unsupervised ML tasks that can be found in real-world applications. However, we consider that the ones selected are among the most relevant in the field and are indeed the tasks where the application of EML algorithms has produced the biggest impact. When reviewing EML methods for the imputation and dimensionality reduction tasks, we consider their application together with both, supervised and unsupervised algorithms, since in the general case these two tasks do not require any type of supervisory signal for their implementation, and can be considered as unsupervised problems on their own.

The chapter begins with a brief presentation of the unsupervised ML tasks, and some of the main concepts involved. Then, in the central part of the chapter, work on the use of EML for each of the four tasks is reviewed. The paper concludes with a discussion on the main trends and challenges in EML research for unsupervised ML tasks.

3.2 Main Concepts

In this section, we cover some of the main concepts of unsupervised ML tasks.

3.2.1 *Data Preparation*

Data preparation comprises the steps required before an ML algorithm can be applied to solve the task. These steps may include the application of several techniques that are usually organized in a pipeline. The choice of the particular methods to include in the ML pipeline depends, both, on the characteristics of the ML task to be addressed, and on the requirements of the ML algorithm that will be used to solve the task. Some of the most common tasks in data preparation are instance selection, feature discretization, and data imputation.

3.2.1.1 Instance Selection

Instance selection consists of selecting a subset of the original instances while preserving as much as possible the information relevant to the task. Instance selection methods can remove instances considered as redundant and reduce the time required for learning the ML model. A straightforward approach to instance selection consists of associating a binary variable to each instance and interpreting that the instance is selected if the corresponding variable has value 1, and not selected if the value is 0. For real-world problems in which the dataset can be very large, the complexity of the problem significantly increases and efficiency is essential for feasible instance selection algorithms [63].

3.2.1.2 Feature Discretization

Feature discretization consists of transforming the original (continuous) features of the problem to discrete ones. This preprocessing step can be necessary for ML algorithms that require discrete inputs. Feature discretization can be also applied to problems with a discrete representation when a reduction in the cardinality of the variables can lead to gains in terms of the efficiency for the ML method. Feature discretization is challenging since interactions among the variables should be taken into account for the discretization process. A rich variety of methods have been proposed for solving this problem [42].

3.2.1.3 Data Imputation

Missing data is a common problem in real-world applications. Most of ML techniques cannot deal with missing values in datasets. One typical strategy is deleting the incomplete instances from the dataset, but this approach is not effective for scenarios where the amount of data is limited. Another approach is data imputation which aims to replace the missing observations in the original dataset with a likely value or “educated guess”.

A variety of imputation methods exist and their effectiveness depends on different factors such as the characteristics of the dataset (e.g., tabular data, spatial data, and time series), the particular type of missing data, and the ML algorithm being applied [26]. For instance, in prediction problems associated with spatial data, the putative imputed values should take into account the localization of the missing measurements. Similarly, in time series datasets, there is usually a temporal dependence between the elements of the data that should be taken into consideration for imputation.

There are many challenges in the design of imputation algorithms, which include the generation of realistic and diverse imputation values and the need for computationally efficient methods that can be deployed in real-world scenarios.

3.2.2 *Outlier or Anomaly Detection*

The outlier detection or anomaly detection task consists of the identification of instances that are abnormal or depart significantly from the distribution of the majority of instances in the dataset. Outlier detection is important because outliers may indicate corrupted or noisy data, or can correspond to anomalous behavior of particular interest for the practitioner.

The outlier detection problem is complicated by the masking and smearing effects [73]. Masking occurs when an outlier masks another preventing its detection. Smearing occurs when one outlier makes another, non-outlier observation, to appear as an outlier. Masking and smearing determine that the approaches that detect outliers one by one can be inaccurate.

Different variants of the anomaly detection task exist, associated with the particular type of data or the domain of application. For example, in time series models, the change-point detection problem [32] consists of identifying abrupt changes in the time series data. In intrusion detection systems [81], the goal is to detect malicious or anomalous behaviors in the system.

3.2.3 *Dimensionality Reduction*

Dimensionality reduction [77] consists of reducing the original number of features to a lower dimensionality while keeping the most relevant information for the problem. This task is critical for problems that involve a large number of features. It can also be part of different ML pipelines, and be used for various purposes such as information visualization, or improving the performance of supervised ML methods. Among the most common approaches to dimensionality reduction are feature selection, feature construction, and manifold learning.

3.2.3.1 **Feature Selection and Feature Extraction**

Feature selection involves the selection of a subset of the original problem features that contain the most relevant information for the tasks addressed and are more suitable for the application of the ML algorithm. The typical approaches to feature selection can be divided into three groups: i) Filter methods, which use information measures to evaluate each set of features; ii) Wrapper-based strategies, which use some performance metric of a supervised learning algorithm as the evaluation criterion for the features; and iii) Hybrid approaches, which combine characteristics of the two previous methods.

Feature construction is the process of deriving new features which condense information of multiple original features.

3.2.3.2 Manifold Learning

The goal of manifold learning (MaL) algorithms is to create a mapping from the original problem representation to a lower embedded representation in such a way that it preserves some meaningful properties of the original dataset, e.g., the local neighborhoods are preserved at every point of the underlying manifold. MaL algorithms propose different nonlinear transformations of the original data. There are many aspects that influence the difficulty of manifold learning for a particular problem, which include the original dimensionality of the data and their distribution, as well as the number of instances.

3.2.4 Association Rule Mining

Association rules [2, 88] allow to extract patterns that characterize relationships between items in a dataset. An association rule has an *if-then* condition form in which the antecedent (*if* part) expresses the condition for the occurrence of the consequent (*then* part). The antecedent is a combination of items found in the dataset, and the consequent is an item that frequently co-occurs with the antecedent. The degree of uncertainty of an association rule is measured using a support, also called coverage, and the confidence of the rule. The support is the number of transactions that include all items in the antecedent and consequent parts of the rule, i.e., how often the rule appears in the dataset. The confidence is the number of instances that the rule predicts correctly, expressed as a proportion of all instances to which it applies.

Association rule mining (ARM) [31] is one of the most used and investigated methods of data mining. One of the reasons for its popularity is that association rules are usually considered interpretable by humans. However, the complexity of mining association rules rapidly increases with the number of instances in the dataset and the number of items. Therefore, many of the developments in the area have been oriented to increase the computational efficiency of these algorithms.

3.3 EML for Data Preparation

There are a number of ways in which EML algorithms can be used for data preparation. In this section, we organize the review of EML applications according to the particular tasks involved in data preparation.

3.3.1 EML for Instance Selection

One of the limitations in the application of evolutionary algorithms (EAs) to instance selection is the computational complexity of the problem as the number of instances grows. Some authors have proposed methods to deal with this question. In [40], a strategy that divides the original dataset using a fuzzy clustering algorithm is proposed. By splitting the dataset into different clusters and applying a genetic algorithm (GA) to each of the clusters, the time required for instance selection reduces significantly. The divide and conquer technique also improves the predictive model performance.

In some scenarios, instance selection can be used as an initial step for data imputation. For example, in [3], a genetic programming (GP) method is proposed that performs instance selection on datasets with incomplete data and uses the selected instances as the source for imputing missing data.

Instance selection has been also tackled as a previous step to regression. In [41], the Non-Dominated Sorting Genetic Algorithm II (NSGA-II) multi-objective EA [18] is used to search for the optimal subset of instances in the training dataset. For this purpose, NSGA-II is combined with the k-Nearest Neighbors (KNN) algorithm which is used for evaluating the solutions during the instance selection process. The multi-objective approach allows to obtain a pool of solutions with a different balance between the number of selected instances and the accuracy of the regression.

In [11], an analysis of two evolutionary instance selection algorithms is presented. The results of the algorithms are compared against classical instance selection algorithms on two applications. For the selected evaluation problems considered, the EAs outperform the classical algorithms.

In [34], a GA and two other instance selection methods are used to determine whether a combination of instance selection from the observed data and missing value imputation performs better than applying the imputation method alone. The authors conclude that for most numerical datasets, performing instance selection to filter out some noisy data, can improve the imputation's results.

The interested readers can consult [19] for a survey on EML applied to instance selection and generation.

3.3.2 EML for Feature Discretization

An early work on the application of EML methods for multivariate discretization was presented by Kwedlo and Kretowski [46], which investigates the discretization problem in the context of decision rule induction. The introduced EA uses six different search operators including crossover and a mutation-like operator. The algorithm is tested on eight datasets, and the results indicate that both classification accuracy and complexity of the discovered rules are comparable with the results of other classifiers.

Kim and Han [38] propose a GA to search the optimal or near-optimal thresholds for feature discretization. The discretized values are passed as inputs of a neural network and used for the prediction of stock price index. The authors conclude that the combination of GA-based discretization and ANN prediction is able to reduce the dimensionality of the feature space, enhancing the generalizability of the predictive model.

In [24], a GA approach is presented for finding a subset of cut-points that defines the best possible discretization scheme of a dataset. The discretization method is validated by using it together with the C4.5 and Naive Bayes (NB) classifiers. This work is extended in [67] to investigate feature discretization in problems where there are high interactions among multiple attributes. The introduced evolutionary multivariate discretizer selects the best combination of boundary cut-points to create discrete intervals. The experimental results show that the algorithm is competitive with other discretization schemes in terms of accuracy and the obtained solutions require a lower number of cut-points than the other discretizers.

Zhu et al. [91] research discretization problems with class-attribute interdependence of the discretization scheme. They analyze information measures of the interdependence between two discrete variables, and propose the use of an improved information distance criterion to evaluate this interdependence. An Artificial Colony Optimization (ACO) algorithm is proposed to detect the optimal discretization scheme, and its evaluation on a real industrial database verifies the effectiveness of the algorithm.

Flores et al. [23] propose the simultaneous discretization of multiple variables using an Estimation of Distribution Algorithm (EDA) [47, 48] approach within a wrapper scheme. EDAs construct a probabilistic model of the best solutions which is then sampled to guide the search. In [23], the solutions sampled by the EDA were evaluated using the Naive Bayes classifier. The algorithm was tested using artificial and real datasets, and the experimental results suggest that the method is effective for feature discretization.

Chen et al. [13] introduce an adaptive GA for discretizing high-resolution remote sensing big data. The GA uses a binary representation and a fuzzy rough model for the definition of the fitness function. The algorithm is compared to other four EML methods and to seven supervised discretization algorithms. The comparison criteria include the running time, search efficiency, interval quality, and classification accuracy. The experimental results show that the new algorithm can effectively solve the problem of feature discretization.

3.3.3 *EML for Imputation*

Al-Helali et al. [4] use a combination of GP and weighted KNN for imputing incomplete observations in symbolic regression. The idea of this hybrid approach is to use both the instance-based similarity of KNN clustering and the feature-based predictability of GP regression. They conduct an extensive evaluation of the algorithm on

10 datasets comparing the results with state-of-the-art algorithms. The results show that the proposed algorithm is significantly better than some of the other imputation methods.

Gaciarena et al. [25, 27] add imputation methods as a component of ML pipelines generated using the Tree-based Pipeline Optimization Tool (TPOT) [62], a tool for automatic ML. They use GP to evolve pipelines that include the appropriate imputation method for a given problem. A different approach to the combination of EAs and imputation methods is presented in [74], where the authors combine multiple imputation with a GP approach that learns a classifier for each pattern of missing values. The algorithm is compared to other three imputation methods on a number of datasets from UCI, showing better results in terms of accuracy and computational time.

Lobato et al. [55] propose a multi-objective GA for data imputation which is based on NSGA-II. They evaluate the new algorithm on 30 datasets with induced missing values and show that the algorithm presents a good trade-off between the different evaluation measures investigated.

EML has also been used for imputation problems defined on spatial data and time series. In [21], a GA was proposed for conditional spatial simulation. The random key genetic algorithm (RKGA) solution encoding was used. The proposed algorithm is applied to impute missing data in optical measurements of gas turbines, and it was much faster than the simulated annealing approach applied to the same problem. In [22], a GA approach was proposed for time series imputation and tested on four weather time series datasets. The proposed algorithm was shown to outperform the expectation-maximization (EM) method commonly used for this problem.

While GAs have been the class of EA most extensively applied to data imputation, other types of EAs have been also applied. Krishna and Ravi [44] introduce an imputation algorithm based on Particle Swarm Optimization (PSO). They use PSO to optimize different fitness functions that evaluate the quality of the imputation process. The authors report that the results are competitive to a hybrid data imputation method based on the K-means clustering algorithm and on a multi-layer perceptron.

Another question analyzed in the literature is how the quality of the imputation process changes as the number of missing data increases. In [1], a hybrid approach combining GA and two types of neural networks is proposed and evaluated in the scenario in which the number of missing cases within a single record increases. The accuracy of the method was shown not to reduce significantly.

3.3.4 EML for Dimensionality Reduction

Dimensionality reduction can be implemented following different strategies; among them are feature selection, feature construction, and manifold learning. In this section we review the applications of EML to these tasks.

3.3.4.1 EML for Feature Selection

Raymer et al. [68] propose a hybrid GA-KNN method to simultaneously solve the problems of feature selection, feature extraction, and classifier training. They evaluated the algorithm on two medical datasets and on the identification of favorable water-binding sites on protein surfaces. The derived GA feature extractor was compared to the sequential floating forward selection algorithm (SFFS) [66] for feature selection for each dataset showing competitive results.

Neshatian and Zhang [60] propose a GP-based approach for feature selection in an image-based face classification problem. To evaluate the quality of the solutions, the fitness functions incorporates a variation of the NB classification algorithm. The authors evaluate their proposal on a face detection dataset and show that according to the performance metrics considered, the classification results improve while the dimensionality of the problem is significantly reduced.

In addition to GA and GP approaches, other EAs have been applied to dimensionality reduction. Yin et al. [85] introduce the immune clonal strategy, a variant of artificial immune systems (AIS), for dimensionality reduction of hyperspectral images. The algorithm selects a subset of bands in the hyperspectral images which are used to solve the classification problem. The hyperspectral band selection problem has been also tackled using a multi-objective immune algorithm in [87]. The authors model the problem considering the objectives of information preservation and redundancy removal. The MOEA approach is tested on three real hyperspectral image datasets and is shown to efficiently explore the solutions with optimal trade-offs between these objectives.

ACO is used in [37] for feature selection in an approach in which features are represented as nodes of a graph. The algorithm is compared to other swarm optimization approaches on 12 datasets and the authors conclude that the proposed algorithm achieves a better feature set in terms of classification accuracy and number of selected features. Xue et al. [82] introduce two bi-objective PSO algorithms with the goal of simultaneously minimizing the classification error rate while minimizing the number of features selected. The two MO-PSO algorithms were compared to single-objective PSO. The results indicate that the algorithms are able to obtain solutions with a small number of features and achieve higher classification performance than methods that use all features.

In [90], three different variants of multi-objective PSO algorithms are introduced for discretization-based feature selection. The algorithms can select an arbitrary number of cut-points for discretization helping to better explore the relevant features. Experiments are conducted on ten benchmark microarray gene datasets comparing the algorithm to other three PSO approaches and two traditional feature selection algorithms. The results show that the multi-objective PSO algorithms significantly outperforms the other methods in terms of test classification accuracy.

EDAs have also been applied to feature selection in a number of works. In [36], two variants of EDAs that respectively use univariate and bivariate marginal distribution models are applied to feature selection in the problem of predicting the survival of cirrhotic patients that have received a treatment. The bivariate model assumes the

existence of some kind of dependencies among the attributes of the database and is shown in the experiments to produce the best average accuracy results for each classifier.

An EDA using Bayesian network models for feature selection is proposed in [35]. The algorithm uses a wrapper approach, over NB and ID3 learning algorithms, to evaluate each candidate solution. The experiments conducted on eight datasets show that the algorithm can significantly reduce the number of features used by the classifiers while keeping a similar predictive accuracy. Saeys et al. [69] use a univariate marginal distribution model to represent the distribution of the best solutions. They propose to use the univariate probabilities to rank features according to relevance. The algorithm is applied to the splice site prediction problem.

Said et al. [70] study a common limitation of EML for discretization-based feature selection, i.e., the fact that encoding the problem solution as a sequence of cut-points can lead to the deletion of possibly important features that have a single or a low number of cut-points. They propose to attack the problem as a bi-level optimization problem and solve it using a co-evolutionary algorithm. The introduced algorithm performs feature selection at the upper level while discretization is done at the lower level. The results of the experiments show that the algorithm outperforms state-of-the-art methods in terms of classification accuracy, generalization ability, and feature selection bias.

While most of feature selection and feature construction methods are applied to supervised ML tasks, there are some situations where no supervision is available. In these cases dimensionality reduction can be required and EML has been also applied in these scenarios. In [39], Kim et al. introduce a bi-objective evolutionary local selection algorithm that simultaneously solves feature selection and clustering. The evolutionary search is hybridized with the application of two clustering algorithms (K-means and EM). The experimental results show that the algorithm can find an appropriate clustering model and also identifies relevant features.

There are also EML proposals that try to solve more than one unsupervised ML task simultaneously. In [5], a GA is applied as the building block of a reduction algorithm that can perform feature selection and instance selection simultaneously. The algorithm is tested on nine real-world datasets with varying difficulties. The experimental results show that the proposed algorithm can reduce both the number of features and the number of instances. The characteristics of the fitness landscape of GP-generated ML pipelines that comprise a feature selection component have been investigated in [28].

3.3.4.2 EML for Feature Construction

One of the first proposals on EML for feature construction was presented in [76], where the authors advanced the idea of using GAs to form combinations of existing features via a well-chosen set of operators. The suggested crossover operator for the GA representation was similar to those used to combine tree-structured programs in GP. The approach was tested on the problem of recognizing visual concepts in

texture data, and feature construction was applied after an initial feature selection step. The results indicated a small but significant improvement in the classification and recognition of real-world images.

The Evolution-COnstructed (ECO) feature construction method is introduced in [54]. It uses a GA to discover highly discriminative series of transforms. The approach focuses on image feature extraction, and the GA representation encodes a sequence of transforms, their operating parameters, and the regions of the image the transforms will operate on. The features generated by the algorithm were used by an AdaBoost classifier and evaluated on several image datasets. The results show that the features generated by the algorithm are highly generalizable and also have high discriminative properties. The ECO method is further enhanced in [86] by reducing the dimensionality of the ECO features.

Otero et al. [65] introduce a GP-based single feature construction method in which each program is encoded using the standard tree-structure representation. To measure the quality of the features, they use the information gain ratio as a fitness function. The algorithm was tested using four datasets and the C4.5 classifiers. For two of the datasets, the error rate of C4.5 with the constructed attribute was smaller than the error rate of C4.5 without it. In the other two cases, there were no statistical differences in the results.

In [59], another GP-based single feature construction method is presented. The authors focus on the analysis of decision tree classification algorithms that used the constructed features. GP uses information gain or the Gini index information measures as the fitness function. The algorithms were tested using four classifiers and five datasets, and one of the conclusions from the experiments was that all classifiers benefit from the inclusion of the evolved feature.

A GP-based algorithm for multiple feature construction is proposed in [43] where the tree-structure representation is used, but each individual's genotype is split into two disjoint parts, *evolving features* and *hidden features*, and each individual decides on its own which features should and should not be visible to the genetic operators. The algorithm was tested using six datasets. The results indicate that the GP-based approach to feature construction provides remarkable gains in terms the predictive accuracy of the ML classifiers.

Ma and Gao [56] propose the use of GP within a filter-based multiple feature construction approach. Each GP program encodes the transformation that creates a single feature. The top β individuals are stored in a storage pool as a way to represent multiple features. The algorithm is compared to other three state-of-the-art feature construction algorithms on nine datasets. The experimental results show that the algorithm can obtain better performance than classifiers that use the original features and outperform the other state-of-the-art algorithms in most cases.

Most EML contributions to feature construction are based on the use of GAs and GP algorithms. However, other evolutionary algorithms have been also applied to this problem.

Xue et al. [83] introduce the use of PSO to feature construction. The algorithm uses a binary encoding to represent the low-level features that are selected, and for each solution perform a local search among a set of arithmetic operators to select those to

construct a new high-level feature. The PSO method is evaluated using seven datasets and three classification algorithms. The results show that using the single constructed feature achieved competitive results than using all the original features, and when the constructed feature is added to the original set, it significantly improves their classification performance in most of the cases. Two other solution representations for PSO algorithms are proposed in [16]. The rationale behind the pair representation and the array representation is to allow PSO to directly evolve function operators. Experiments are conducted on seven datasets and from the results, it is concluded that the two PSO algorithms can increase the classification performance of the original PSO representation for feature construction presented in [83].

In addition to PSO, other EML approaches that have been used for feature construction include Gene Expression Programming [20], Kaizen Programming [17], and EDAs [71]. Several other works apply EML techniques for feature selection and feature construction. The interested readers can consult [82] for a detailed survey on this area.

3.3.4.3 EML for Manifold Learning

The application of EML algorithms to dimensionality reduction tasks other than feature selection and feature construction has not received the same attention in the literature. Nevertheless, there are some recent works that propose the use of EML for MaL.

Orzechowski et al. [64] propose two different GP-based MaL methods and compared them to other eight classical MaL algorithms over a large collection of 155 datasets. One of the proposed GP-based methods, ManiGP, showed far more separable results than any other MaL included in the comparison. However, the high computational time required by this method makes its practical use very limited.

Lensen et al. [50] propose the use of GP for solving MaL tasks where the goal is the preservation of local neighborhoods. This work is extended in [52] by learning manifolds with different number of dimensions using a bi-objective approach. In [51], an approach similar to the one introduced in [50] is compared to other five baseline methods on 12 datasets. The results show that the algorithm is competitive with all baselines for higher dimensionality ($d > 8$) and often outperforms them at lower dimensionality.

Uriot et al. [75] evaluate different fitness functions to guide GP in MaL tasks. They compare the GP algorithm to other three classical algorithms for dimensionality reduction and report that GP is a competitive approach to obtain interpretable mappings for this task.

3.3.5 *EML for Outlier or Anomaly Detection*

Outlier detection was one of the ML problems earliest addressed using EML. In [49], Leardi presented a GA-based algorithm that, while solving a feature selection problem within a cross-validation framework, was able to detect outliers. The idea was to identify deletion groups (folds) for which the results of the classifier were not as accurate as in the majority of the other folds, suggesting the presence of one or more outliers. The GA was tested using six different datasets, and authors reported that the algorithm was able to easily identify anomalous examples unidentified by classical methods.

Crawford and Wainwright [14] presented one of the initial EML approaches to outlier detection. They researched the question of outlier diagnostics, i.e., rather than minimizing the effect of outlying data, outlier diagnostics try to assess the influence of each point in the dataset. They used a GA with a variant of two-parent uniform order-based crossover and accomplished the identification of potential outlier subsets. Experiments were conducted using five different datasets and the authors concluded that the GA approach was effective for generating subsets for multiple-case outlier diagnostics.

Bandyopadhyay and Santra [8] also investigated the question of estimating the “outlierness value” of an outlier group as a relative measure of how strong an outlier group is. They propose a GA that projects the original points to a lower dimension and identifies outliers in this lower dimensional space as those with inconsistent behavior (defined in terms of their sparsity values). The algorithm was evaluated on three artificial and three real-life datasets showing the effectiveness of the new algorithm.

Tolvi [73] introduces a GA which searches among different possible groupings of the data into outlier and non-outlier observations. The analysis is focused on outlier detection linear regression models. The algorithm is validated on two small datasets, and its scalability is investigated when the number of instances is increased.

In [29], a hybrid GA that includes a multi-start metaheuristic is proposed for anomaly detection. The rationale behind the use of the multi-start mechanism is to generate a suitable number of detectors with high anomaly detection accuracy. It also incorporates the K-means clustering algorithm as a way to reduce the size of the training dataset while keeping the diversity. The new proposal is applied to the evaluation of intrusion detection and compared to other six ML techniques. The experimental results suggest that the algorithm outperforms the other methods in terms of test accuracy.

Cucina et al. [15] introduce a GA approach to detect outliers in multivariate time series. The rationale behind the use of GAs in this context is that it can process multiple outliers simultaneously, being less vulnerable to the masking and smearing effects. To test the algorithm, authors use four simulated time series models. The experimental results show that the GA is able to provide a better performance than other methods for the case of consecutive outliers, while the results were similar for the case of well-separated outliers.

In [58], a PSO algorithm is proposed for the outlier detection problem and compared to a Local Outlier Factor (LOF) algorithm on five real datasets. In most of the datasets, the PSO approach outperforms LOF.

3.3.6 EML for Association Rule Mining

One straightforward representation of association rules is to associate two bits with each possible item to indicate whether it belongs to the antecedent part, the consequent part, or to none of them. This is the approach used in [79] where if these two bits are 00 then the corresponding attribute appears in the antecedent part, and if it is 11 then the attribute appears in the consequent part. The other two combinations, 01 and 10, will indicate the absence of the attribute in either of these parts. The problem with this representation is that the length of the chromosome grows significantly for problems with a large number of items. A different type of representation is proposed in [84] where each item has associated one binary variable, and there is an additional integer variable that indicates the position in the chromosome that separates the antecedent from the consequent.

Kuo et al. [45] advocate the use of PSO for the generation of association rules from a database. The algorithm is applied to the analysis of investors' stock purchase behavior. The authors state that while the commonly used Apriori algorithm requires the minimal support and confidence to be determined subjectively, the PSO approach can determine these two parameters quickly and objectively.

Ghosh and Nath [30] propose a multi-objective GA for ARM in which, together with the predictive accuracy, the comprehensibility and the interestingness measures computed for each rule were considered as objectives. The paper introduces some strategies to diminish the computational cost of the algorithm. Instead of using the entire dataset, the algorithm works on a sample of the original database, and the sample may not truly reflect the actual database. Another multi-objective approach is presented in [79], where the use of an external archive with the non-dominated solutions is proposed as a way not to lose rules that were generated at intermediate generations.

In [9], a multi-objective PSO algorithm for ARM is presented. The objectives to optimize include confidence, comprehensibility, and interestingness. The algorithm is compared to other four evolutionary algorithms in three datasets. The authors conclude that the new algorithm shows a robust behavior for association rule mining with a simple particle design and without requiring a priori discretization.

For comprehensive surveys on the use of evolutionary algorithms for ARM, the interested reader is referred to [7, 72, 78].

3.4 Conclusions

In this chapter, we have presented an overview of the use of EML to unsupervised ML tasks. Our analysis shows that most of the EML approaches are based on GAs and GP although other evolutionary algorithms such as PSO and ACO are also applied. As a summary of this chapter, we have extracted a number of observations that can guide the reader at the time of applying EML techniques to unsupervised ML tasks or when developing new EML approaches.

Dependence between Different ML Tasks and Holistic EML Approaches

EML is usually applied to particular ML tasks (e.g., dimensionality reduction, and imputation). However, it is clear that the solutions found for some ML task of the ML pipeline can affect the solution of the next stages of the pipeline. Therefore, when applied to solve real-world problems, holistic EML approaches are required that take into consideration the dependencies between tasks.

There are works in the field of EML that simultaneously confront more than one ML task. For example, discretization-based feature selection EML algorithms can find an appropriate discretization of the features that allow to select relevant features more accurately [70, 90]. There is also an interaction between instance selection and imputation, and some EML methods have been proposed that exploit this relationship [3]. As ML approaches become more sophisticated and involve more steps, EML algorithms should pay more attention to these dependencies.

The Potential of Multi-objective EML

Multi-objective EML strategies are involved in many scenarios where EML outperforms classical ML methods for unsupervised ML problems. As in other domains, it is common in ML problems the necessity to obtain solutions that satisfy multiple performance criteria, which are often conflicting. Multi-objective EML methods are natural candidates for the solution of these problems.

New Ways of Hybridization Between EML and Classical ML Techniques

In several of the EML applications, the evolutionary algorithm is combined with supervised and unsupervised classical ML methods. Traditional ML metrics, and clustering and classification algorithms are frequently incorporated as part of the fitness function. Novel ways of integrating the ML techniques as part of the EML could lead to more powerful EML strategies.

Computational Efficiency as a Barrier to Scalability

Increasing the population size of population-based EML algorithms in which the evaluation of a single individual implies processing the complete dataset can become impractical for very large datasets. EML algorithms that can work with small samples of the dataset have been proposed to deal with the scalability challenge. New strategies are required in order to make EML more competitive with other ML methods.

References

1. Abdella, M., Marwala, T.: The use of genetic algorithms and neural networks to approximate missing data in database. In: IEEE 3rd International Conference on Computational Cybernetics, 2005, pp. 207–212. IEEE (2005)
2. Agrawal, R., Imieliński, T., Swami, A.: Mining association rules between sets of items in large databases. In: Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, pp. 207–216 (1993)
3. Al-Helali, B., Chen, Q., Xue, B., Zhang, M.: Gp with a hybrid tree-vector representation for instance selection and symbolic regression on incomplete data. In: 2021 IEEE Congress on Evolutionary Computation (CEC), pp. 604–611. IEEE (2021)
4. Al-Helali, B., Chen, Q., Xue, B., Zhang, M.: A new imputation method based on genetic programming and weighted KNN for symbolic regression with incomplete data. *Soft. Comput.* **25**(8), 5993–6012 (2021)
5. Albuquerque, I.M.R., Nguyen, B.H., Xue, B., Zhang, M.: A novel genetic algorithm approach to simultaneous feature selection and instance selection. In: 2020 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 616–623. IEEE (2020)
6. Andreassen, A., Feige, I., Frye, C., Schwartz, M.D.: JUNIPR: a framework for unsupervised machine learning in particle physics. *Eur. Phys. J. C* **79**, 1–24 (2019)
7. Badhon, B., Jahangir, M.M., Kabir, S.X., Kabir, M.: A survey on association rule mining based on evolutionary algorithms. *Int. J. Comput. Appl.* **43**(8), 775–785 (2021)
8. Bandyopadhyay, S., Santra, S.: A genetic approach for efficient outlier detection in projected space. *Pattern Recogn.* **41**(4), 1338–1349 (2008)
9. Beiranvand, V., Mobasher-Kashani, M., Bakar, A.A.: Multi-objective PSO algorithm for mining numerical association rules without a priori discretization. *Expert Syst. Appl.* **41**(9), 4259–4273 (2014)
10. Berg-Kirkpatrick, T., Bouchard-Côté, A., DeNero, J., Klein, D.: Painless unsupervised learning with features. In: Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics, pp. 582–590 (2010)
11. Cano, J.R., Herrera, F., Lozano, M.: Instance selection using evolutionary algorithms: an experimental study. In: Advanced Techniques in Knowledge Discovery and Data Mining, pp. 127–152 (2005)
12. Casolla, G., Cuomo, S., Cola, V.S.D., Piccialli, F.: Exploring unsupervised learning techniques for the internet of things. *IEEE Trans. Industr. Inf.* **16**(4), 2621–2628 (2019)
13. Chen, Q., Huang, M., Wang, H., Guangquan, X.: A feature discretization method based on fuzzy rough sets for high-resolution remote sensing big data under linear spectral model. *IEEE Trans. Fuzzy Syst.* **30**(5), 1328–1342 (2021)
14. Crawford, K.D., Wainwright, R.L.: Applying genetic algorithms to outlier detection. In: Proceedings of The Sixth International Conference on Genetic Algorithms (ICGA-1995), pp. 546–550 (1995)
15. Cucina, D., Di Salvatore, A., Protopapas, M.K.: Outliers detection in multivariate time series using genetic algorithms. *Chemometr. Intell. Labor. Syst.* **132**, 103–110 (2014)
16. Dai, Y., Xue, B., Zhang, M.: New representations in PSO for feature construction in classification. In: Applications of Evolutionary Computation: 17th European Conference, EvoApplications 2014, Granada, Spain, April 23–25, 2014, Revised Selected Papers 17, pp. 476–488. Springer (2014)
17. de Melo, V.V., Banzhaf, W.: Kaizen programming for feature construction for classification. In: Genetic Programming Theory and Practice XIII, pp. 39–57 (2016)
18. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* **6**(2), 182–197 (2002)
19. Derrac, J., García, S., Herrera, F.: A survey on evolutionary instance selection and generation. In: Modeling, Analysis, and Applications in Metaheuristic Computing: Advancements and Trends, pp. 233–266. IGI Global (2012)

20. Drozdz, K., Kwasnicka, H.: Feature set reduction by evolutionary selection and construction. In: In Proceedings of the 4th KES International Symposium on Agent and Multi-Agent Systems: Technologies and Applications, KES-AMSTA-2010, Part II, pp. 140–149. Springer (2010)
21. Eklund, N.H.W.: Using genetic algorithms to estimate confidence intervals for missing spatial data. *IEEE Trans. Syst., Man, Cybern., Part C (Appl. Rev.)* **36**(4), 519–523 (2006)
22. García, J.C.F., Kalenatic, D., Bello, C.A.L.: Missing data imputation in time series by evolutionary algorithms. In: Advanced Intelligent Computing Theories and Applications. With Aspects of Artificial Intelligence: Proceedings of the 4th International Conference on Intelligent Computing, ICIC-2008, pp. 275–283. Springer (2008)
23. Flores, J.L., Inza, I., Larrañaga, P.: Wrapper discretization by means of estimation of distribution algorithms. *Intell. Data Anal.* **11**(5), 525–545 (2007)
24. García, S., López, V., Luengo, J., Carmona, C.J., Herrera, F.: A preliminary study on selecting the optimal cut points in discretization by evolutionary algorithms. In: International Conference on Pattern Recognition Applications and Methods (ICPRAM-2012), pp. 211–216 (2012)
25. Garcíarena, U., Mendiburu, A., Santana, R.: Towards a more efficient representation of imputation operators in TPOT (2018). CoRR, [arXiv:1801.04407](https://arxiv.org/abs/1801.04407)
26. Garcíarena, U., Santana, R.: An extensive analysis of the interaction between missing data types, imputation methods, and supervised classifiers. *Expert Syst. Appl.* **89**, 52–65 (2017)
27. Garcíarena, U., Santana, R., Mendiburu, A.: Evolving imputation strategies for missing data in classification problems with TPOT (2017). CoRR, [arXiv:1706.01120](https://arxiv.org/abs/1706.01120)
28. Garcíarena, U., Santana, R., Mendiburu, A.: Analysis of the complexity of the automatic pipeline generation problem. In: IEEE Congress on Evolutionary Computation (CEC-2018), pp. 1–8. IEEE (2018)
29. Ghanem, T.F., Elkilani, W.S., Abdul-Kader, H.M.: A hybrid approach for efficient anomaly detection using metaheuristic methods. *J. Adv. Res.* **6**(4), 609–619 (2015)
30. Ghosh, A., Nath, B.: Multi-objective rule mining using genetic algorithms. *Inf. Sci.* **163**(1–3), 123–133 (2004)
31. Hipp, J., Güntzer, U., Nakhaeizadeh, G.: Algorithms for association rule mining—a general survey and comparison. *ACM SIGKDD Explorat. Newsl* **2**(1), 58–64 (2000)
32. Horváth, L., Hušková, M.: Change-point detection in panel data. *J. Time Ser. Anal.* **33**(4), 631–648 (2012)
33. Zhengping, H., Li, Z., Wang, X., Zheng, S.: Unsupervised descriptor selection based meta-learning networks for few-shot classification. *Pattern Recognit.* **122**, 108304 (2022)
34. Huang, M.W., Lin, W.C., Tsai, C.F.: Outlier removal in model-based missing value imputation for medical datasets. *J. Healthcare Eng.* **2018** (2018)
35. Inza, I., Larrañaga, P., Etxeberria, R., Sierra, B.: Feature subset selection by Bayesian network-based optimization. *Artif. Intell.* **123**(1–2), 157–184 (2000)
36. Inza, I., Merino, M., Larranaga, P., Quiroga, J., Sierra, B., Girala, M.: Feature subset selection by genetic algorithms and estimation of distribution algorithms: a case study in the survival of cirrhotic patients treated with TIPS. *Artif. Intell. Med.* **23**(2), 187–205 (2001)
37. Kashef, S., Nezamabadi-pour, H.: An advanced ACO algorithm for feature subset selection. *Neurocomputing* **147**, 271–279 (2015)
38. Kim, K., Han, I.: Genetic algorithms approach to feature discretization in artificial neural networks for the prediction of stock price index. *Expert Syst. Appl.* **19**(2), 125–132 (2000)
39. Kim, Y.S., Nick Street, W., Menczer, F.: Evolutionary model selection in unsupervised learning. *Intell. Data Anal.* **6**(6), 531–556 (2002)
40. Kordos, M., Blachnik, M., Scherer, R.: Fuzzy clustering decomposition of genetic algorithm-based instance selection for regression problems. *Inf. Sci.* **587**, 23–40 (2022)
41. Kordos, M., Łapa, K.: Multi-objective evolutionary instance selection for regression tasks. *Entropy* **20**(10), 746 (2018)
42. Kotsiantis, S., Kanellopoulos, D.: Discretization techniques: a recent survey. *GESTS Int. Trans. Comput. Sci. Eng.* **32**(1), 47–58 (2006)
43. Krawiec, K.: Genetic programming-based construction of features for machine learning and knowledge discovery tasks. *Genet. Program Evolvable Mach.* **3**, 329–343 (2002)

44. Krishna, M., Ravi, V.: Particle swarm optimization and covariance matrix based data imputation. In: 2013 IEEE International Conference on Computational Intelligence and Computing Research, pp. 1–6. IEEE (2013)
45. Kuo, R.J., Chao, C.M., Chiu, Y.T.: Application of particle swarm optimization to association rule mining. *Appl. Soft Comput.* **11**(1), 326–336 (2011)
46. Kwedlo, W., Kretowski, M.: An evolutionary algorithm using multivariate discretization for decision rule induction. In: In Proceedings of the Third European Conference on Principles of Data Mining and Knowledge Discovery, PKDD-99, pp. 392–397. Springer (1999)
47. Larrañaga, P., Karshenas, H., Bielza, C., Santana, R.: A review on probabilistic graphical models in evolutionary computation. *J. Heurist.* **18**(5), 795–819 (2012)
48. Larrañaga, P., Lozano, J.A. (eds.): *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*. Kluwer Academic Publishers, Boston (2002)
49. Leardi, R.: Application of a genetic algorithm to feature selection under full validation conditions and to outlier detection. *J. Chemom.* **8**(1), 65–79 (1994)
50. Lensen, A., Xue, B., Zhang, M.: Can genetic programming do manifold learning too? In Proceedings of the 22nd European Conference on Genetic Programming, EuroGP-2019, pp. 114–130. Springer (2019)
51. Lensen, A., Xue, B., Zhang, M.: Genetic programming for manifold learning: preserving local topology. *IEEE Trans. Evol. Comput.* **26**(4), 661–675 (2021)
52. Lensen, A., Zhang, M., Xue, B.: Multi-objective genetic programming for manifold learning: balancing quality and dimensionality. *Genet. Program Evolvable Mach.* **21**(3), 399–431 (2020)
53. Liakos, K.G., Busato, P., Moshou, D., Pearson, S., Bochtis, D.: Machine learning in agriculture: A review. *Sensors* **18**(8), 2674 (2018)
54. Lillywhite, K., Lee, D.-J., Tippetts, B., Archibald, J.: A feature construction method for general object recognition. *Pattern Recogn.* **46**(12), 3300–3314 (2013)
55. Lobato, F., Sales, C., Araujo, I., Tadaiesky, V., Dias, L., Ramos, L., Santana, A.: Multi-objective genetic algorithm for missing data imputation. *Pattern Recogn. Lett.* **68**, 126–131 (2015)
56. Ma, J., Gao, X.: A filter-based feature construction and feature selection approach for classification using genetic programming. *Knowl.-Based Syst.* **196**, 105806 (2020)
57. Metodiev, E.M., Nachman, B., Thaler, J.: Classification without labels: Learning from mixed samples in high energy physics. *J. High Energy Phys.* **2017**(10), 1–18 (2017)
58. Mohemmed, A.W., Zhang, M., Browne, W.N.: Particle swarm optimisation for outlier detection. In: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, pp. 83–84 (2010)
59. Muhamram, M.A., Smith, G.D.: Evolutionary feature construction using information gain and gini index. In: Proceedings of the 7th European Conference on Genetic Programming, EuroGP-2004, pp. 379–388. Springer (2004)
60. Neshatian, K., Zhang, M.: Dimensionality reduction in face detection: a genetic programming approach. In: 24th International Conference on Image and Vision Computing, pp. 391–396. IEEE (2009)
61. Noroozi, M., Favaro, P.: Unsupervised learning of visual representations by solving jigsaw puzzles. In: Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part VI, pp. 69–84. Springer (2016)
62. Olson, R.S., Moore, J.H.: TPOT: A tree-based pipeline optimization tool for automating machine learning. In: Workshop on Automatic Machine Learning, pp. 66–74. PMLR (2016)
63. Olvera-López, J.A., Carrasco-Ochoa, J.A., Martínez-Trinidad, J.F., Kittler, J.: A review of instance selection methods. *Artif. Intell. Rev.* **34**, 133–143 (2010)
64. Orzechowski, P., Magiera, F., Moore, J.H.: Benchmarking manifold learning methods on a large collection of datasets. In: Proceedings of the 23rd European Conference on Genetic Programming, pp. 135–150. Springer (2020)
65. Otero, F.E.B., Silva, M.M.S., Freitas, A.A. and Nievolà, J.C.: Genetic programming for attribute construction in data mining. In: In Proceedings of the 6th European Conference on Genetic Programming, EuroGP-2003, pp. 384–393. Springer (2003)

66. Pudil, P., Novovičová, J., Kittler, J.: Floating search methods in feature selection. *Pattern Recogn. Lett.* **15**(11), 1119–1125 (1994)
67. Ramírez-Gallego, S., García, S., Benítez, J.M., Herrera, F.: Multivariate discretization based on evolutionary cut points selection for classification. *IEEE Trans. Cybern.* **46**(3), 595–608 (2015)
68. Raymer, M.L., Punch, W.F., Goodman, E.D., Kuhn, L.A., Jain, A.K.: Dimensionality reduction using genetic algorithms. *IEEE Trans. Evolut. Comput.* **4**(2), 164–171 (2000)
69. Saeyns, Y., Degroeve, S., Aeyels, D., Van de Peer, Y., Rouzé, P.: Fast feature selection using a simple estimation of distribution algorithm: A case study on splice site prediction. *Bioinformatics* **19**(2), ii179–ii188 (2003)
70. Said, R., Elarbi, M., Bechikh, S., Coello, C.A.C., Said, L.B.: Discretization-based feature selection as a bi-level optimization problem. *IEEE Trans. Evolut. Comput.* (2022)
71. Shelton, J., Dozier, G., Bryant, K., Small, L., Adams, J., Popplewell, K., Abegaz, T., Alford, A., Woodard, D.L. and Ricanek, K.: Genetic and evolutionary feature extraction via X-TOOLS. In: Proceedings of the International Conference on Genetic and Evolutionary Methods (GEM), p. 1 (2011)
72. Telikani, A., Gandomi, A.H., Shahbahrami, A.: A survey of evolutionary computation for association rule mining. *Inf. Sci.* **524**, 318–352 (2020)
73. Tolvi, J.: Genetic algorithms for outlier detection and variable selection in linear regression models. *Soft. Comput.* **8**, 527–533 (2004)
74. Tran, C.T., Zhang, M., Andraeae, P., Xue, B.: Multiple imputation and genetic programming for classification with incomplete data. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 521–528 (2017)
75. Uriot, T., Virgolin, M., Alderliesten, T. and Bosman, P.A.N.: On genetic programming representations and fitness functions for interpretable dimensionality reduction. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 458–466 (2022)
76. Vafaei, H., De Jong, K.: Genetic algorithms as a tool for restructuring feature space representations. In: Proceedings of 7th IEEE International Conference on Tools with Artificial Intelligence, pp. 8–11. IEEE (1995)
77. Van Der Maaten, L., Postma, E., Van den Herik, J., et al.: Dimensionality reduction: a comparative review. *J. Mach. Learn. Res.* **10**(66–71), 13 (2009)
78. Ventura, S., Luna, J.M.: Pattern Mining with Genetic Algorithms, pp. 63–85. Springer International Publishing, Cham (2016)
79. Wakabi-Waiswa, P.P., Baryamureeba, V.: Extraction of interesting association rules using genetic algorithms. *Int. J. Comput. ICT Res.* **2**(1), 26–33 (2008)
80. Wang, J., Biljecki, F.: Unsupervised machine learning in urban studies: a systematic review of applications. *Cities* **129**, 103925 (2022)
81. Wu, S.X., Banzhaf, W.: The use of computational intelligence in intrusion detection systems: A review. *Appl. Soft Comput.* **10**(1), 1–35 (2010)
82. Xue, B., Zhang, M., Browne, W.N., Yao, X.: A survey on evolutionary computation approaches to feature selection. *IEEE Trans. Evolut. Comput.* **20**(4), 606–626 (2015)
83. Xue, B., Zhang, M., Dai, Y., Browne, W.N.: PSO for feature construction and binary classification. In: Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation, pp. 137–144 (2013)
84. Yan, X., Zhang, C., Zhang, S.: Genetic algorithm-based strategy for identifying association rules without specifying actual minimum support. *Expert Syst. Appl.* **36**(2), 3066–3076 (2009)
85. Yin, J., Wang, Y., Jiankun, H.: A new dimensionality reduction algorithm for hyperspectral image using evolutionary strategy. *IEEE Trans. Industr. Inf.* **8**(4), 935–943 (2012)
86. Zhang, M., Lee, D.-J.: Efficient training of evolution-constructed features. In: Proceedings of the 11th International Symposium on Advances in Visual Computing, ISVC-2015, Part II, pp. 646–654. Springer (2015)
87. Zhang, M., Gong, M., Chan, Y.: Hyperspectral band selection based on multi-objective optimization with high information and low redundancy. *Appl. Soft Comput.* **70**, 604–621 (2018)

88. Zhao, Q., Bhowmick, S.S.: Association Rule Mining: A Survey, vol. 135. Nanyang Technological University, Singapore (2003)
89. Zhou, M., Duan, N., Liu, S., Shum, H.-Y.: Progress in neural NLP: modeling, learning, and reasoning. *Engineering* **6**(3), 275–290 (2020)
90. Zhou, Y., Kang, J., Kwong, S., Wang, X., Zhang, Q.: An evolutionary multi-objective optimization framework of discretization-based feature selection for classification. *Swarm Evol. Comput.* **60**, 100770 (2021)
91. Zhu, W., Wang, J., Zhang, Y., Jia, L.: A discretization algorithm based on information distance criterion and ant colony optimization algorithm for knowledge extracting on industrial database. In: 2010 IEEE International Conference on Mechatronics and Automation, pp. 1477–1482. IEEE (2010)

Chapter 4

Evolutionary Computation and the Reinforcement Learning Problem



Stephen Kelly and Jory Schossau

Abstract Evolution by natural selection has built a vast array of highly efficient lifelong learning organisms, as evidenced by the spectacular diversity of species that rapidly adapt to environmental change and acquire new problem-solving skills through experience. Reinforcement Learning (RL) is a machine learning problem in which an agent must learn how to map situations to actions in an unknown world in order to maximise the sum of future rewards. There are no labelled examples of situation→action mappings to learn from and we assume that no model of environment dynamics is available. As such, learning requires active trial-and-error interaction with the world. Evolutionary Reinforcement Learning (EvoRL), the application of evolutionary computation in RL, models this search process at multiple time scales: individual learning during the lifetime of an agent (i.e., operant conditioning) and population-wide learning through natural selection. Both modes of adaptation are wildly creative and fundamental to natural systems. This chapter discusses how EvoRL addresses some critical challenges in RL including the computational cost of extended interactions, the temporal credit assignment problem, partial-observability of state, nonstationary and multi-task environments, transfer learning, and hierarchical problem decomposition. In each case, the unique potential of EvoRL is highlighted in parallel with open challenges and research opportunities.

S. Kelly ()

Department of Computing and Software, McMaster University, Hamilton, Ontario L8S 4L7,
Canada

e-mail: kellys32@mcmaster.ca

J. Schossau

Department of Computer Science & Engineering, Michigan State University, 428 S. Shaw Ln.,
East Lansing, MI 48824-1226, USA

4.1 Introduction

Intelligent behaviour emerges in natural systems as a result of adaptation occurring over at least two vastly different timescales: evolution and lifetime learning. Both modes of adaptation are wildly creative and critical to the development of robust intelligence. The cause and effect relationship between evolution and learning is so nuanced and complex that it is a field of study in its own right [22, 66] and is colloquially referred to as the “Baldwin Effect” [11, 49, 119]: that phenotypic plasticity might have a direct effect on inherited genetics and its evolution, which of course can in turn affect phenotypic plasticity.

RL has a long research heritage in psychology and ethology as a model for how animals learn through trial-and-error interaction with the environment over their lifetime [149, 173]. In machine RL, computational agents learn to maximise the sum of future rewards through interactions with unknown environments. A variety of lifetime learning algorithms have been formalised and extensively researched to solve the RL problem [163]. Their integration with deep learning has achieved remarkable success solving complex sequential decision tasks [118]. The construction of behavioural agents has also been a fertile playground for evolutionary computation since its inception. Early work by Holland [75] and Koza [97] describe methods of evolving game-playing strategies and controllers for complex dynamical systems. Since then, EvoRL, the application of evolutionary computation to RL, has emerged to represent a family of evolutionary search algorithms that consider both individual learning during the lifetime of an agent (i.e., operant conditioning) *and* population-wide learning through natural selection. While the development of behavioural agents is typically the focus, EvoRL represents a form of meta-learning in which artificial agents (body and brain), their environment and the interaction among them might all be subject to adaptation, Fig. 4.1.

In addition to developing strong sequential prediction algorithms for real-world problem-solving, EvoRL has a rich heritage from the field of Artificial Life, seeking to better understand brains, behaviour, and their evolution in the natural world [2, 15, 109]. An underlying theme from this perspective is how knowledge from evolutionary ecology and cognitive science might guide evolution in building intelligent agents that learn and creatively discover solutions to problems [35, 115]. Creativity is characterised as a system’s capacity to generate new things of value. It is inherently a two-part process: (1) *Explore* the environment to experience new structures and strategies for problem solving; and (2) *Exploit* valuable discoveries, making sure to store that information within the agent. This creative process is abundant in natural evolution, as evidenced by the spectacular diversity of well-adapted species, and is fundamental to animal learning, perception, memory, and action [19]. AI researchers have long wondered how machines could be creative, and the digital evolution community is perpetually surprised by unplanned, emergent creativity in their systems [105]. In particular, creativity is the driving force behind active trial-and-error problem-solving. It is emphasised in parts of this review because it can

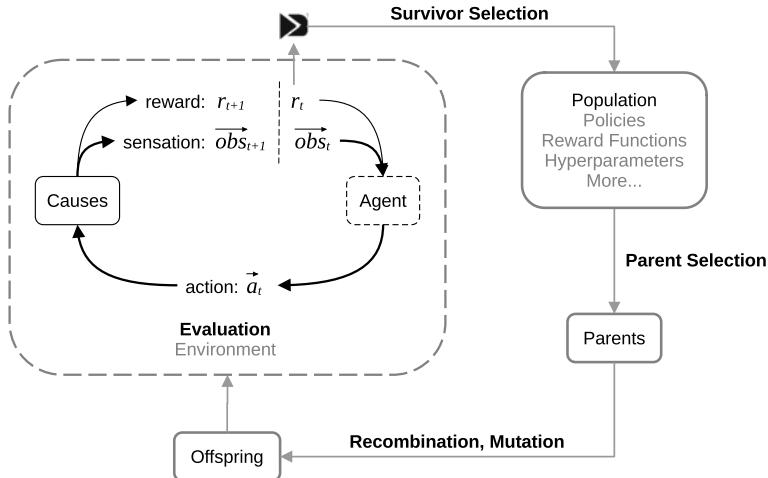


Fig. 4.1 Adaptation at two timescales in EvoRL. In the inner loop, an individual agent learns to predict the effect of its actions in the environment. In the outer loop, an evolutionary cycle of selection and variation operates on a population of individuals. Any part of the system might evolve including decision policies, reward functions, environment transition functions, initial states, goal states, hyperparameters, and more

be a general engineering principal to guide evolution towards building computer programs with the structure and efficiency of the biological brain [141].

This chapter is organised to highlight how the holistic EvoRL search paradigm can be applied to the critical challenges in RL. Section 4.2 reviews RL as a machine learning problem and outlines a taxonomy of RL and EvoRL algorithms. The remaining sections discuss EvoRL in context with unique aspects of the RL problem including the computational cost of evaluating policies through sequential interaction with large state spaces (Sect. 4.3), the temporal credit assignment problem (Sect. 4.4), partial-observability of state (Sect. 4.5), nonstationary and multi-task environments (Sect. 4.6), transformational learning and hierarchical problem decomposition (Sect. 4.7). While these challenges are all interrelated, we use this decomposition as a structure to explore how evolutionary computation has been studied and applied in RL, discussing select historical and state-of-the-art algorithms to illustrate major achievements and open research opportunities. We hope this perspective complements surveys in the field [9, 145, 185].

4.2 Machine Reinforcement Learning

How can we build artificial agents that learn from trial-and-error experience in an under-explored world?

Given an environment (typically a complex and stochastic dynamical system), RL is the problem of mapping situations to actions in order to maximise the sum of future discounted rewards. The problem can be formally modelled as a Markov Decision Process (MDP) with $\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{S}_0 \rangle$ in which $s \in \mathcal{S}$ represents the state space, $a \in \mathcal{A}$ the action space, \mathcal{T} is a transition function that computes a probability distribution over possible next states $P(s_{t+1}|s_t, a_t)$, a reward function \mathcal{R} computes $P(r_{t+1}|s_t, a_t) \in \mathbb{R}$, and \mathcal{S}_0 is the initial state distribution. The agent's decision-making *policy* is denoted by $\pi_\theta(a_t|s_t)$ where π is the policy's algorithm and $\theta \in \mathbb{R}^n$ is a set of parameters.

Interactions in RL are typically episodic, beginning with the agent situated in a start state defined by the task environment. From there, time unfolds in discrete timesteps (t) in which the agent observes the environment via sensory inputs \overrightarrow{obs}_t , takes an action based on the observation $\vec{a}_t|\overrightarrow{obs}_t$, and receives feedback in the form of a reward signal $r_t \in \mathbb{R}$.¹ The process repeats in a loop until a task end state is encountered or the episode ends for another reason, such as a time constraint. Rewards are often time delayed. The agent's objective is therefore to select actions that maximise the long-term cumulative reward from its current state s_t , or $G_t = \sum_{k=0}^{\infty} \gamma^k \mathcal{R}(s_{t+k}, a_{t+k})$ where $\gamma \in [0.1]$ is a discount factor on future rewards (See inner loop of Fig. 4.1). The reward signal is used to communicate what we want the policy to achieve without specifying *how* to achieve it. Unlike supervised learning, there are no prior labelled examples of $\overrightarrow{obs} \rightarrow \vec{a}$ associations to learn from. As such, the problem is considered *semi-supervised*. Given this constraint, the broad challenge of RL is how to discover sequential decision policies through direct interaction with the task environment.

4.2.1 Reinforcement Learning Algorithms

A common approach in RL is to attempt to discover the optimal policy π_θ^* , which selects with probability 1 the action expected to yield the largest return when the agent continues to act optimally thereafter:

$$\pi_\theta^*(s_t, a_t) = \begin{cases} 1, & \text{if } a_t = \arg \max_{a \in \mathcal{A}} (\mathbb{E}_{\pi_\theta^*}(G_t(s_t, a))) \\ 0, & \text{otherwise} \end{cases} \quad (4.1)$$

RL algorithms for attempting to discover optimal policies π_θ^* include three general phases:

1. **Explore:** Generate experience by interacting with the environment following policy π_θ^k (inner loop of Fig. 4.1).

¹ In real-world tasks, observations \overrightarrow{obs}_t and actions \vec{a}_t are likely to be multidimensional and expressed as a mix of discrete and continuous values.

2. **Evaluate:** Assess the quality of π_θ^k 's actions using the reward signal \mathcal{R} .
3. **Update:** Generate a new policy π_θ^{k+1} based on evaluations.

In the context of EvoRL, it is informative to characterise RL algorithms by the timescale(s) over which these phases are computed.

Step-based RL methods explore in action space at each timestep. In this case, evaluation estimates the quality of state-action pairs, and the gradient of the reward signal is used to guide learning updates, potentially **online**, i.e., within an episode. A representative example is Temporal Difference (TD) learning, which aims to minimise the error between *predicted* future rewards following a policy from a given state, and the *actual* rewards received post-execution. For example, some form of TD algorithm is used by methods that optimise an action-value function (Q-function) to predict the value of a state-action pair. The policy algorithm can then query the value function at each timestep to select an action. Q-learning [163] is a well-known example, which has been combined with deep convolutional neural network architectures in Deep Q-Networks (DQN) and many of its variants [80, 118]. Step-based RL requires exploration steps (eg. random actions) within the episode, which can result in jittery behaviour during learning and produce unreproducible trajectories for an exploration-free policy. Furthermore, while step-based RL with value functions can efficiently learn a high-quality policy under tabular settings [81], it often has difficulty scaling to large state and action spaces and can even diverge with function approximation in continuous versions of those spaces [188]. It is possible to avoid value functions and directly adjust the policy, π_θ^k , in the direction of the performance gradient, for example, using the Policy Gradient Theorem [163]. This approach is assumed by policy-based algorithms such as REINFORCE [190] and, more recently, Proximal Policy Optimisation (PPO) [143]. Policy-based RL algorithms scale better to high-dimensional state and action spaces, which are common in real-world RL tasks such as robot control. However, Policy-based RL can suffer from high variance in the estimation of gradients and thus requires many samples for an accurate estimate, which can be inefficient. Actor-critic methods represent a hybrid approach that attempts to capture the strengths of value-based and policy-based RL by simultaneously optimising a policy function (actor) and value function (critic) [96, 117, 134]. Interestingly, actor-critic methods can achieve greater stability by operating at two timescales: the critic changes rapidly using TD learning, while the actor updates at a slower timescale in an approximate gradient direction based on information provided by the critic [96]. In short, step-based RL methods perform a learning update relative to each decision and the immediate reward within the temporal sequence.

Episode-based RL methods explore in policy space at each episode. Evaluation estimates the quality of the policy as a whole using the cumulative reward received after following the policy for an entire episode. This is traditionally the approach taken in EvoRL, in which each learning update effectively creates a new policy (e.g., by selection and variation operators in the Evolutionary Algorithm (EA), see outer loop of Fig. 4.1), and thus the search process is performed over the space of possible decision policies within a particular representation. Episode-based RL can be characterised as a form of *black-box* or *derivative-free* optimisation since policy

updates do not rely on reward signal gradients. Episode-based RL offers several benefits in comparison to step-based RL. It generates smooth control trajectories that are reproducible under test, it can handle non-Markovian reward definitions, and the resulting exploration in policy space is well-suited for adaptation in environments with sparse reward. The relative challenges and merits of step-based and episode-based RL are discussed in more detail in the context of the temporal credit assignment problem in Sect. 4.4.2.

In this chapter we focus on **model-free** RL, meaning that no prior model of the environment’s transition or reward function is available. As such, trial-and-error interaction with the environment is required for the agent to gain experience (collect data) and learn. However, note that as soon as learning occurs, the agent itself becomes a world model of sorts (i.e., environment dynamics are encoded in learned behaviours). Furthermore, building internal world models is an explicit requirement for agents operating in dynamic and partially observable environments. For example, if complete information about the state of the environment is not available from each observation \overrightarrow{obs}_t , it may be necessary for the agent to explicitly learn a world model that can be queried in order to predict intermediate variables that—when combined with the sensory input \overrightarrow{obs}_t —form a complete representation of the state of the environment. Memory models are discussed in more detail in Sect. 4.5.

Finally, note that RL algorithms are also categorised as **on-policy** methods that learn by executing their current policy in the environment, while **off-policy** methods learn from experience collected by a different policy. For example DQN is an off-policy method in which learning updates are performed based on samples from a *replay memory* of many past experiences, i.e., experience $e_t = (\overrightarrow{obs}_t, \vec{a}_t, r_t, s_{t+1})$.

4.2.2 Evolutionary RL Algorithms

There are two distinct approaches to evolving a policy π_θ : searching in the *parameter* space, and searching in the *program* space. For example, those that search in the *parameter* space might assume a fixed structure for the policy’s algorithm π as a neural network often does, optimising only the weight parameters θ . Those that search in *program* space simultaneously build the policy algorithm π *and* optimise its parameters θ . The latter approach is more general and open-ended, since no prior constraints are placed on how the policy works or how complex it should be in order to solve a particular problem. Indeed, any behaviour algorithm could conceivably be evolved, given an expressive programming language and suitable memory [13].

All EvoRL methods for policy search assume some form of EA in which a population of candidate behaviours are iteratively improved through selection, crossover, and mutation. Most approaches can be further characterised as variants of Evolutionary Strategies (ES), Genetic Programming (GP), or Neuroevolution (NE).

In the context of RL, ES are typically used as continuous black-box optimization algorithms suitable for search in large *parameter spaces*, for example optimising the

weights of a neural network.² They have recently gained significant attention in RL for their ability to scale over multiple CPUs and rapidly train deep neural networks to a performance level comparable to that achieved by traditional gradient descent based optimisers [139].

GP is the special case of an EA in which the individuals being evolved are computer programs. In the most general EvoRL case, GP builds policy algorithms from scratch starting with basic mathematical operations and memory structures as building blocks. As such, GP performs an evolutionary search in *program space*, simultaneously discovering a policy algorithm and optimising its parameters. Early EvoRL experiments established GP’s power to discover general problem solvers in simulated control tasks [97, 98] and build adaptive controllers for real robots [130]. GP has since been used to evolve graph-structured policies (tangled program graphs [87] and cartesian genetic programming [191]), which rival multiple deep RL algorithms in Atari visual RL tasks while being significantly more efficient to evolve *and* deploy post-search/training. GP has also been used to evolve loss functions for step-based RL algorithms to optimise [33]. In this case, GP rediscovers the TD method from scratch, producing domain-agnostic RL algorithms capable of learning control tasks, gridworld-type tasks, and Atari games.

NE has a rich history in EvoRL, exploring how evolutionary computation can evolve the weights, topology, learning rules, and activation functions of neural networks for behavioural agents [155, 156, 187]. In particular, pairing NE with deep RL has significant potential to leverage the best of evolution and stochastic gradient descent [145] learning.

In short, EvoRL represents a diverse universe of algorithms that use digital evolution to address RL problems.

4.3 Complex Evaluations

Machine RL potentially incurs significant computational cost. In complex tasks, the agent is likely to observe the environment through a high-dimensional sensory interface (e.g., a video camera). However, scaling to high-dimensional state observations presents a significant challenge for machine learning, and RL in particular. Each individual input from the sensory interface \overrightarrow{obs}_t , or state variable, may have a large (potentially infinite) number of possible values. Thus, as the number of state variables increases, there is a significant increase in the number of environmental observations required for the agent to gain the breadth of experience required to build a strong policy. This is broadly referred to as the Curse of Dimensionality (Sect. 1.4 of [18]). The temporal nature of RL introduces additional challenges: complete information about the environment is not always available from a single observation (i.e., the environment is only partially observable) and delayed rewards are common, requiring the agent to make thousands of decisions before receiving enough feedback to

² In this case ES might be considered a form of neuroevolution.

assess the quality of its behaviour [83]. As a result, RL systems benefit from policy algorithms with low computational complexity and learning methods that are *sample-efficient*. EvoRL has been shown to address the RL scaling issue through at least three mechanisms, namely, incremental growth, dynamic inference complexity, and sensory space selectivity.

4.3.1 Incremental Growth

Several EvoRL methods leverage evolution to build the policy algorithm starting from the simplest working devices and incrementally complexifying through interaction with the task environment. This implies that the computational cost of inference is low early in evolution, and only increases as more complex structures provide fitness and performance gains. In particular, NE and GP exhibit these properties in a range of RL scenarios.

Neuroevolution of Augmenting Topologies (NEAT) [156] is an evolutionary algorithm for building neural network architectures and optimising their weight parameters. NEAT uses a variable-length linear representation consisting of node genes and connection genes. Node genes encode input, hidden, and output node types. Connection genes specify the possibly recurrent network links and their weights. *Historical markings* annotate connection genes with unique innovation identifiers that are inherited by descendants. These markings facilitate crossover by allowing genes representing compatible network structure to be aligned. NEAT uses *Speciation* to maintain diversity and prevent a single niche of network architectures (and associated weights) from dominating the population, protecting novel architectures by allowing them to develop within a niche until they become competitive. Populations are initialised containing small neural networks and since increases in complexity must pass natural selection, the search is bound by *incremental growth*. The initial version of NEAT was analysed on the classic double inverted pendulum balancing control task [156], where it is several times more efficient than previously existing neuroevolution methods. NEAT has since been improved and demonstrated in various tasks including building complex policy architectures in robotics [157], strategic decision-making in *fractured* environments where the correct action varies discontinuously as the agent moves from state to state [95], and building minimal policy architectures in Atari video games from compact state representations [176]. From these origins, NEAT has been extended in several ways that benefit EvoRL [132], and its various innovations appear in nearly every section of this chapter.

Sample-Efficient Automated Deep Reinforcement Learning (SEARL) [55] simultaneously evolves neural architectures, trains them in an off-policy setting, and optimises their hyperparameters. SEARL uses a population of agents with varying structure and hyperparameters such as the learning rate. During evaluation, each structure-hyperparameter combination gets a fitness score and the environment interactions are stored as state-action-state transitions in a shared replay memory. Based on the fitness score, the best structure-hyperparameter combinations are selected, the hyperparameters are mutated and trained with samples from a shared replay mem-

ory and then evaluated again. Shared experience replay memory stores trajectories from *all* evaluations and thus provides a diverse set of samples for each agent during the training phase. This speeds up training and reduces the potential of over-fitting. Diverse training samples and incremental growth allow SEARL to build and train agents using up to ten times fewer environment interactions as random search or PBT [79] (a closely related, state-of-the-art neuroevolution method) while matching or improving on their final performance.

4.3.2 Dynamic Inference Complexity

Tangled Program Graphs (TPGs) are a relatively new GP framework developed specifically for multi-task RL in large (e.g., visual) state spaces with partial-observability [87, 90]. They have been extended to support large-scale parallelisation [44] and ultra-fast inference on embedded hardware through the generation of optimised standalone C code [43]. The critical contribution of TPGs is to enable the construction of hierarchical agents through *compositional evolution*: evolving policy algorithms that combine multiple models (algorithms), which were previously adapted independently [184]. In addition to incremental growth, this allows TPGs to mirror how the neocortex is highly modular, containing many semi-independent predictive models with similar underlying structure. These models are organised hierarchically and compete to form a multi-faceted and efficient prediction machine [8, 68, 122]. The inference time complexity of a Tangled Program Graph (TPG) policy’s many-model architecture is *dynamic* because only a subset of models require execution at each timestep, improving efficiency. Conversely, most deep learning neural networks model the world in terms of a hierarchy of concepts, with each concept defined through its relation to simpler concepts (e.g., [118]). This implies that every level of the hierarchy contributes to every prediction, incurring an excessive computational cost for inference. Such networks do not account for the fact that neocortical regions are organised into columns and layers, and how this structure suggests a theory of intelligence in which thousands of semi-independent world models are switched in and out of the overall inference process over time [8, 68, 122]. Recently, sparse deep learning architectures have appeared which dynamically activate sub-components instead of the full graph [53]. Interestingly, evolution is being used to automatically decompose the task into subcomponent models and combine them into multi-task graphs [57]. The application of these methods to RL problems is an emerging field [110]. Other systems also have this dynamic inference complexity property of much more minimal computation, such as Linear Genetic Programming [26], WireWorld [146], and Hierarchical Temporal Memory [68].

4.3.3 Sensor Space Selectivity

Sensor space selectivity implies that a policy is able to ignore inputs (variables in \overrightarrow{obs}_t) not relevant to the task at hand [7, 160], which potentially improves learning efficiency by simplifying or abstracting the input space [82]. In GP, an evolved policy’s use of sensor data in \overrightarrow{obs}_t (eg. image pixels) is entirely an evolved property. In the case of TPGs, evolution builds saccade-like perception behaviour in which the multi-model agent jumps between subsets of sensor data in \overrightarrow{obs}_t that are critical to prediction and avoids processing irrelevant inputs [87, 88]. This behaviour can take many evolutionary cycles to emerge, but the result is more efficient than sweeping a convolution filter over the entire input space as in deep RL [118]. The sensory abstraction provided by convolution filters undoubtedly improves learning but does not capitalise on the efficiencies associated with learning when to ignore particular state variables.

4.3.4 Parallel Algorithms and Hardware Acceleration

Scaling through parallelisation is an intrinsic benefit of population-based search and optimisation methods. RL naturally lends itself to parallelisation since, in the simplest cases, each episodic evaluation is independent and can thus be executed in parallel if appropriate hardware is available. Secondly, if there is diversity in the population, then the time for convergence to optima scales with population size, thus making parallel evolutionary algorithms much more effective than their linear counterparts [175]. Using a novel inter-process communication strategy, ES was recently shown to scale to over a thousand parallel workers, distributing parameters such that many unique deep RL models are evaluated in parallel [139]. This framework shows impressive parallelisation results for two benchmarks: MuJoCo 3D humanoid walking, and Atari games. The framework solves MuJoCo 3D humanoid walking in 10 minutes using 1,440 CPU cores, which takes 18 CPU cores over 11 hours to solve. For Atari games, competitive results were often achieved after just one hour of training. Furthermore, ES is relatively invariant to action frequency and delayed rewards, and tolerant of long episode horizons. These benefits are largely due to adopting episode-based RL, which is not sensitive to temporal discounting as compared to RL techniques, such as Q-learning and Policy Gradients.

In asynchronous, multi-agent RL tasks such commercial Real-Time Strategy video games, the RL interaction loop typically runs asynchronously for multiple interconnected agents. Lamarckian [10] is an open-source, high-performance EvoRL platform that supports an asynchronous Markov Decision Process (MDP) and can scale up to thousands of CPU cores. An asynchronous tree-shaped data broadcasting method is proposed to reduce *policy-lag*, an inefficiency in distributed off-policy RL that occurs when an actor’s policy falls several updates behind the learner’s policy. Multiple current EvoRL algorithms are benchmarked at scale such as NSGA-

II [42] for Game Balancing in Real-time strategy (RTS) games, i.e., discovering diverse Non-Player-Characters (NPCs) that keep gameplay interesting for humans, and Population-Based-Training (PBT) [79] for general Atari Video game playing. Lamarckian significantly improves sampling efficiency and training speed as compared to the state-of-the art RLlib [107].

Multiple recent platforms build on APIs for GPU acceleration to support neuroevolution EvoRL. EvoJax [168] is a scalable neuroevolution toolkit to train neural networks running in parallel across multiple TPU/GPUs. EvoJAX implements the evolutionary algorithm, neural network, and tasks all in NumPy, which is compiled just-in-time to run on accelerators. EvoJax and its sister projects QDax [108] and evosax [103] (JAX-Based Evolution Strategies) include robot control tasks simulated in Brax [56] and natively support evolutionary strategy variants including CMA-ES and Map-Elites, while EvoX [78] supports the complete OpenAI Gym RL testbed and includes implementations of Differential Evolution (DE) and Particle Swarm Optimisation (PSO). The Parallel Evolutionary and Reinforcement Learning (PEARL) Library [169] proposes a novel combination of OpenAI’s Natural Evolutionary Strategy [139] with the Adam optimiser equation [92]. These GPU-accelerated neuroevolution frameworks can optimise policies in Atari or MuJoCo robotics tasks with significantly less time than using parallel CPUs.

4.3.5 Discussion and Open Challenges

Incremental growth is an intrinsic property of natural and digital evolution. The ability to scale the complexity of problem solvers through interaction with the problem is a practical method of optimising both the model and sample complexity of RL algorithms. Adaptive complexification also reduces the burden of prior knowledge, which would otherwise be necessary to estimate the correct solution complexity.

Evolutionary computation also naturally lends itself to parallelisation, and most EvoRL implementations scale to many CPUs by performing independent computations in parallel, for example, each episodic policy evaluation can be computed simultaneously in single-agent settings, and asynchronously in multi-agent settings. In addition to scaling the speed of evolution, parallel and multi-population evolutionary models support diversity by allowing individuals to develop in isolated niches, with occasional migration as a means to distribute this diversity throughout the experiment. There is significant opportunity to further improve efficiency and scaling potential of EvoRL by modelling properties of biological and societal niches. For example, [77] demonstrated how hierarchically organising populations according to fitness level can help scale evolutionary search to complex problems by nurturing a diverse set of intermediate “stepping stones” on the path to the most fit individuals. Biological niches are also time-dependent. Natural environments change over time (e.g., weather becomes colder or hotter), and individuals that accumulate neutral variations under a static environment may have an advantage if these variations happen to become advantageous when the environment changes. Dynamic

environments are ubiquitous in RL settings, where policies are exposed to unique sequences of tasks that change over abrupt or gradual intervals (discussed in more detail in Sect. 4.6). More generally, more research is needed to synergise the speed potential of parallel processing while also leveraging the diversity potential of parallel evolution in nature.

Finally, artificial neural networks benefit greatly from hardware acceleration because some matrix manipulations required during learning and inference can be computed in parallel on GPUs. While neuroevolution can also leverage GPU acceleration, a similar breakthrough for non-connectionist program-search methods such as GP remains an open problem. Field-programmable gate arrays (FPGAs) are programmable computing platforms in which digital circuitry can be synthesised on-the-fly, without physical integrated circuit development. Their potential for accelerating GP is alluring and has a long history, with recent results showing that GP representations can be engineered such that an FPGA executes individual programs in a single clock cycle and transition between separate programs within a single cycle [37, 72]. For EvoRL, simulation of the environment is generally a more significant compute bottleneck than executing the agent policy. However, the target application for RL policies is often autonomous robotics or other low-power edge/embedded applications, where fast and efficient execution is essential. Hardware-accelerated program search, potentially entirely implemented on a single chip [177], is a promising direction for future study.

4.4 Exploration and Temporal Credit Assignment Ambiguity

Credit assignment is the mechanism used to modify policy behaviour relative to information obtained through the reward signal during exploration. In sequential decision problems, the task environment typically provides the policy with a reward in response to each action taken (i.e., at every cycle in the inner loop of Fig. 4.1). However, it is often difficult to determine which specific decision(s) influence the reward gradient over time. For example, even actions with a neutral or negative stepwise reward may ultimately contribute to a successful outcome. On top of this, the reward signal from each interaction is often noisy. This means that many approaches to RL are unstable during learning and overly sensitive to initial conditions of the environment and/or hyperparameter settings. This is known as the temporal credit assignment problem [74, 163].

4.4.1 Exploration

Exploration at the evolutionary timescale in EvoRL is driven by variation operators (namely mutation and crossover) that introduce structural and behavioural changes to individuals within a population of policies. This variation causes agents to interact with their environment in new ways and gain new experiences. However, additional effort is often required in order to ensure that mutation (or other diversity generating operators) results in new organisms that are measurably different (in a phenotypic and/or genotypic sense) from what exists in the current population. In particular, explicit maintenance of a diverse policy population is important to ensure a thorough search of the behaviour space. Indeed, diversity maintenance is a well-known factor in preventing evolving agent populations from getting stuck on local optima [125], even for behavioural tasks [39]. Furthermore, in multi-agent settings, or if a single agent will ultimately be represented by a group behaviour, then diversity maintenance might be critical in developing individual group members that complement each other in a group setting that is, producing individuals that *specialise* or succeed or fail in different ways. This section identifies two generic approaches to exploration through population diversity maintenance that could potentially be used in combination: quality diversity algorithms and competitive coevolution.

4.4.1.1 Quality Diversity

Quality Diversity (QD) evolutionary algorithms employ a fitness function that measures properties other than goal-directed problem-solving. At one extreme is novelty *as* the objective, i.e., **Novelty Search (NS)**. In an influential study, Lehman and Stanley [106] discover that abandoning the objective (i.e., ignoring the reward signal) and searching for behavioural novelty alone can help policies escape local optima in a deceptive maze navigation task. NS is based on the observation that the path to a desirable or innovative behaviour often involves a series of divergent stepping stones that may not resemble the final behaviour and cannot be known ahead of time. Interestingly, novelty search also potentially leads to increasing complexity because the search will continually exhaust the behaviour space potential of simpler structures. Two important design issues now appear:

- For a behaviour to be considered novel, it must be measurably different from other behaviours in the current population and all behaviours that have *ever* been discovered during the search. If behavioural novelty is described as the observable characteristics of policies that are interacting with the environment, then how could behavioural novelty be characterised and compared in a meaningful way? Designing a Behaviour Characterization (BC) and similarity metrics is a difficult task [123]. If multiple BCs are available, combining them in a single run has been shown to increase performance in robotics tasks, and sidesteps the need to choose the single most appropriate metric. Combining BCs by switching between them over time often leads to better results than taking the mean behavioural diversity,

while requiring less computational power [24, 46, 86]. Similarly, switching goals during evolution potentially also promotes diversity, speeds up evolution, and leads to better generalisation [84, 133].

- How can the EA and policy representation support the continual production of novelty and complexity over time? This is the research question explored in the field of open-ended evolution [12], often studied in EvoRL research (e.g., [87, 183]).

Initially demonstrated under the deceptive maze navigation and bipedal walking tasks, behavioural novelty search outperformed objective-based search. However, it was subsequently found that novelty search alone is most effective when the BC is correlated with the objective function. For example, in maze navigation, characterising an agent’s behaviour by its final position in a maze will naturally imply that novelty search explores the objective space of navigating to the maze target location (e.g., [106]). However, when the BC and objective function are decoupled, optimising novelty *in combination* with a measure of behavioural quality is required [38]. To address this, **Quality diversity (QD)** algorithms elaborate on the ideas of NS, multiple BCs, and goal-switching by keeping track of many different behavioural niches that are evolved simultaneously, and in effect trying to discover stepping stones by periodically testing the abilities of agents from one niche in other niches (i.e., goal-switching). If search gets stuck in one goal dimension, it might make progress in other dimensions, and thereby get around deceptive search spaces. A well-known Quality Diversity (QD) approach is the Multidimensional Archive of Phenotypic Elites (MAP-Elites) algorithm [39, 124]. In [124], Map-Elites is used to build a diverse repertoire of walking policies for a legged robot. If the robot is damaged, it may use this prior repertoire as a skill library to guide a trial-and-error algorithm that rapidly discovers a behaviour to compensate for the damage. While effective for diversity maintenance, Map-Elites has difficulty scaling because there can be an exponential explosion of niches as the number BCs (or objective functions) increases. Several EvoRL studies have emerged that mitigate this issue in robotics and control [34, 178]. Finally, building on Map-Elites, Go-Explore [50] specifically addresses *incremental exploration* following the key ideas of remembering states, returning to them (GO), and exploring from them (Explore). Go-Explore was the first (Evo) RL method to succeed in two Atari game titles with extremely sparse, deceptive reward signals: Montezuma’s Revenge and Pitfall.

Novelty quantifies the degree to which a policy differs from prior ways of solving a particular problem. *Surprise* is an alternative similarity measurement which quantifies the degree to which a policy is different from what is expected, given prior knowledge or intuition about a particular problem environment. Since a prediction of trends that evolution is following from one generation to the next is a necessary component for modelling evolutionary surprise, it can be characterised as a temporal novelty metric. **Surprise search** is intended to mimic the self-surprise cognitive process of creativity over the (much longer) evolutionary timescale. In a deceptive maze navigation task, allowing surprise *and* novelty to operate together leads to QD algorithms of higher efficiency, speed, and robustness [64]. Surprise can also

be encouraged by rewarding policies for displaying *curiosity* during their lifetime. Curiosity-ES [174] rewards policies for discovering transitions that are different from those seen previously. The idea is that curiosity naturally leads to a diversity of policies because reward is provided for covering unexplored or under-explored transitions. Curiosity-ES discovers optimal policies faster than Novelty Search Evolutionary Strategies [36] and, unlike novelty search, no BCs are required.

Finally, the evolvability of a system refers to the system's potential to continuously discover new organisms with higher fitness than their parents. Novelty search leads indirectly to evolvability by favouring individuals that are very mobile in the behaviour space, and thus highly evolvable [47]. **Evolvability search** aims to optimise this directly by calculating the behavioural diversity of each individual's immediate offspring and selecting for organisms with increased offspring variation. While this process is computationally expensive, it discovers successful policies faster than novelty search in the deceptive maze navigation task.

4.4.1.2 Competitive Coevolution

Competitive coevolution simultaneously searches for policies and training cases (e.g., environment configurations), coupling their fitness such that training cases are encouraged to challenge and discriminate between the performance of policies without resulting in disengagement [31]. The paradigm is akin to predator/prey or host/parasite relationships in nature. It has been used extensively in EvoRL to drive exploration by coevolving policies with problem environments (e.g., opponent behaviours in games) [27, 129, 135, 147, 165, 166]. Recently, the Paired Open-Ended Trailblazer (POET) [183] combines NS, QD, and goal-switching to coevolve an open-ended progression of diverse and increasingly challenging environments together with agents capable of solving them. POET is evaluated in a 2D obstacle course domain, where it ultimately generates and solves diverse and challenging environments, many of which cannot be solved by direct RL, or a hand-designed curriculum tailored for each generated environment.

If the goal of competitive coevolution is to build multiple policies that will eventually form a group behaviour, credit assignment will need to strike a balance between rewarding generalists (policies that solve the multiple training cases) versus specialists (those that solve specific, typically more challenging training cases). Depending on the representation for agents, it can be exceedingly challenging to resolve the two into a single policy [32].

4.4.2 Credit Assignment

The (temporal) credit assignment problem is addressed differently by step-based and episode-based learning algorithms [14, 121].

Episode-based learning takes place in EvoRL in the outer loop of Fig. 4.1. In this case, decision-level credit is applied implicitly since policies that make better decisions will receive higher cumulative reward (hence higher fitness) and produce more offspring. Thus, evolution manages the temporal credit assignment problem by directing the search in favour of agents that make decisions that contribute to a positive overall outcome.

Step-based (i.e., lifetime) learning can be characterised as explicitly rewarding agents for their ability to predict future states of the environment. Such forward-prediction is an indispensable quality in nonstationary and partially observable RL environments (Sects. 4.6 and 4.5). However, the ability of brains to model the world emerged from evolution because it helped organisms survive and thrive in their particular environments, not because forward-prediction was explicitly selected. We can characterise any dynamic behaviour as *lifetime learning*—that is, a behaviour that undergoes change during its lifetime (i.e., within a single episode) in response to environmental interaction. For example, some GP models use temporal memory mechanisms to allow information acquisition from their environments while they solve problems, and to change their future behaviour in response to such information [90, 130], or even to reprogram themselves on-the-fly [99]. Importantly, this change need not be directed by gradient information available from a stepwise reward signal. Nonetheless, several EvoRL methods combine evolution and learning by leveraging such lifetime reward gradient information to address two primary objectives: (1) Improve unstable learning and brittle convergence properties that are extremely sensitive to hyperparameters, commonly associated with (gradient-based) Deep RL [117]; and (2) Improve the sample efficiency of learning.

4.4.2.1 Lifetime Reward Gradients in EvoRL

There are various ways to resolve the trade-off between individual and group credit assignment by combining the episode-based and step-based approaches. Group-level selection can be used to encourage group-level performance, by rewarding the entire group by its combined success and replicating together [150]. However, group-level selection emphasises the group over the individual. Other interesting proposals exist that shape rewards in a more detailed way to balance group and individual, such as the Shapley Q-value approach that considers the magnitude of individual contributions [182], or the Minimum-Reward Attribution approach that rewards a group by the poorest performer to achieve an implicit balance [142].

OpenAI solved this another way. For their OpenAI Five multiplayer Dota bot, there was shaping in two ways: behavioural shaping and reward shaping. The five sub-players were constrained to stay within their spatial regions of the game for the first portion of the match and then allowed to venture for the remainder of the match. This had the effect of encouraging self-sufficiency and mitigating early poor choices. The team of five sub-players was also encouraged to operate first independently, then cooperatively, through reward shaping. This was another way to reward cooperation, but to first ensure self-sufficiency had been established [17].

Other examples of EvoRL using lifetime reward gradients include the recently proposed Evolution-Guided Policy Gradient in Reinforcement Learning (ERL) algorithm [91]. In ERL, parameters learned via lifetime gradient descent are periodically inserted into a population of evolving neural networks, leading to faster learning (i.e., improved sample efficiency). CEM-RL [136] combines the cross-entropy method (CEM) with Twin Delayed Deep Deterministic policy gradient (TD3). In this case, policy parameters (weights) learned through stepwise reward gradients are directly applied to half of the CEM agents at each iteration to increase training/sample efficiency. Asynchronous Evolution Strategy-Reinforcement Learning (AES-RL) [104] integrates ES with off-policy gradient RL methods to take advantage of the stability of ES while improving sample efficiency. Evolution-based Soft Actor-Critic (ESAC) [162] combines ES with Soft Actor-Critic (SAC) to explore policies in weight space using evolution and exploit gradient-based knowledge using the SAC framework. The method achieves greater rewards in several MuJoCo and DeepMind control suite tasks compared to SAC, TD3 PPO, and ES while producing stable policies and reducing the average wall-clock time for training compared with ES on equivalent hardware.

Enrico et al. [113] and Shibe et al. [196] both demonstrate how efficiency and stability gains of combining evolutionary policy search with gradient descent can be extended to on-policy RL. Supe-RL [113] periodically generates a set of children by adding Gaussian mutation to the gradient-learned policy weights, and then gradually updating (soft updating) the on-policy learner with weights from the best child. Soft updating integrates evolutionary weight changes slowly, greatly improving the stability of learning while maintaining efficiency gains. Supe-RL has outperformed ERL and PPO across a variety of MuJoCo domains with continuous action spaces, some of which use raw pixels for observations.

All of these methods manage to capture strengths from evolutionary and gradient-based methods: evolution improves the stability of learning by mitigating the temporal credit assignment issue while gradient-based learning increases the speed of convergence by reducing the number of policy-environment interaction samples (See [145] for a comprehensive review of algorithms combining evolution with gradient-base deep RL). However, it is not clear a reward signal would be available to the policy when deployed in the field, outside of the *semi-supervised* RL training environment. As such, these methods do not directly address a critical benefit of lifetime learning: the ability to continuously adapt to novel environments that are not known at training or search time [100]. Learning of this nature is discussed in Sect. 4.6.

4.4.2.2 Gradient-Free EvoRL and Cooperative Coevolution

EvoRL is increasingly exploited as a general framework for building agents that can learn to predict what is important for their survival *without* being explicitly rewarded for it by tracking reward signal gradients during their lifetime. From the perspective of credit assignment, modularity (behavioural and structural) and coevolution are recurring themes in order to incrementally construct heterogenous models from

multiple coevolving subsystems. Three aspects of the system are simultaneously managed by coevolution: (1) Automatic decomposition of the problem into multiple simpler subtasks or smaller search spaces; (2) Development of partial solutions, or *specialist* models that solve one or more subtasks, each representing a partial solver relative to the ultimate objective; (3) Crediting the behavioural contributions of the different specialist components to the ultimate success of the heterogeneous (multi-model) agent. The methodology is synonymous with *compositional evolution* discussed in Sect. 4.3, but here we discuss its implication on credit assignment in RL.

In natural ecosystems, organisms of diverse species compete and cooperate with different species in their quest for survival. The fitness of each individual organism is coupled to that of other individuals inhabiting the environment. As species evolve they specialise and co-adapt their survival strategies to those of other species. This phenomenon of coevolution has been used to encourage complex behaviours in EAs and has particular relevance to credit assignment in RL tasks. Species represent policy components, or partial world models, that decompose the problem in time and space. Each individual forms a part of a complete problem solver but need not represent meaningful behaviour individually. Cooperative coevolution implies that the components are evolved by measuring their contribution to complete problem solvers, and recombining those that are most beneficial to solving the task [61].

Symbiotic Adaptive Neuro-Evolution (SANE) demonstrates how a population of individual neurons can be coevolved to form a fully functional neural network [120]. SANE coevolves two different populations simultaneously: a population of neurons and a population of network blueprints that specify how the neurons are combined to form complete networks. In this system, neurons assume diverse (specialised) roles that partially overlap, resulting in a robust encoding of control behaviour in the Khepera robot simulator. SANE is more efficient, more adaptive, and maintains higher levels of diversity than the previous network-based population approaches. A critical aspect of SANE’s neurons (and modular, many-model systems in general) is that each neuron forms a complete input to output mapping, which makes every neuron a primitive problem solver in its own right. Such systems can thus form subsumption-style architectures [29], where certain neurons provide basic functionality and other neurons perform higher level functions that fix or build on basic partial solvers. Cooperative Synapse Neuroevolution (CoSyNE) [61] uses cooperative coevolution at the level of individual synaptic weights. For each network connection, there is a separate subpopulation consisting of real-valued weights. Networks are constructed by selecting one member from each subpopulation and plugging them into a predefined network topology. In a challenging double pole balancing task with properties including continuous state and action spaces, partial-observability, and non-linearity, CoSyNE is compared with a wide variety of step-based RL methods (including Policy Gradient RL, Q-learning with MLP, SARSA(λ), and others) and episode-based methods (including NEAT, SANE, ESP, CMA-ES, and others). It was found that episodic methods in general—particularly coevolutionary search with CoSyNE—can solve these problems much more reliably and with fewer episodic evaluations

than step-based RL approaches. The authors suggest this is because “the networks can [much more] compactly represent arbitrary temporal, non-linear mappings” and that the CoSyNE algorithm aggressively explores weight combinations. Though, they note that aggressive exploration may be detrimental for larger networks.

In the context of multi-agent robot control, [4, 94] present a method for providing local evaluation functions to directly evolve individual components in a coevolutionary system. They address how individual evaluation functions are selected so that the components evolve to optimise those evaluation functions *and* display coordinated behaviour. Design considerations are discussed for component evaluation functions which are aligned with the global task objective *and* sensitive to the fitness changes of specialised components, while remaining relatively insensitive to fitness changes of other components in the system. More recently, population-based training (PBT) has been used to automatically learn internal rewards for teams of (step-based) reinforcement learners in a 3D multiplayer first-person video game [79], and automatically optimise the relative importance between a set of shaping rewards for the multi-agent MuJoCo soccer game [111]. Both methods aim to align the myopic shaping rewards with the sparse long-horizon team rewards and generate cooperative behaviours. However, it is noted that variance stemming from temporal credit assignment in RL updates remains an open issue.

The Symbiotic Bid-Based GP (SBB) [48] explicitly addresses credit assignment in RL by coevolving teams of programs in which each individual program represents an action-value function for a particular discrete, task-specific action. SBB uses symbiotic coevolution to build teams (host) of diverse programs (symbiont), each of which models the *context* in which a given action is appropriate, relative to the observable environmental state \vec{s}_t . Inter-program bidding determines which program will “win” the right to define the policy’s action at each timestep. The team as a whole thus represents a collective decision-making organism. Teams (policies) are built through episode-based RL in which fitness is only assigned at the team level. Mutation operators may add and remove programs from a team. Programs may appear in multiple teams, but individual programs that are left with no team memberships are permanently deleted. In this way, programs that specialise in some part of the problem (i.e., a particular region of the state space or a particular temporal region in the task dynamics within an episode) will be protected as long as they are a member of a team that achieves high fitness. In order words, selective pressure at the phylogenetic level effectively performs temporal credit assignment for individual components without needing to assign credit or blame to components directly. Explicit diversity maintenance, care of fitness sharing [58], is critical to building teams with diverse/robust program complements. SBB’s team-based approach to problem decomposition and credit assignment is the basis for tangled program graphs, which support highly efficient hierarchical RL policies, discussed for visual RL and nonstationary, multi-task RL in Sects. 4.3 and 4.6.

More recently, cooperative coevolution has been employed in Deep RL frameworks to construct and manage the training of the components in modular, heterogeneous network architectures [67, 138]. Deep Innovation Protection (DIP) addresses

the credit assignment problem in training complex heterogeneous neural network models end-to-end. DIP uses a multi-component network consisting of a vision component, memory component (LSTM), and controller component that is trained end-to-end with a simple EA. Mutations add Gaussian noise to the parameter vectors of the networks. An age marker keeps track of mutations to the visual or memory systems, allowing the framework to automatically throttle selection pressure on specific components that have recently been subject to mutation. The main insight is that when components upstream in the network change, such as the visual or memory system in a policy, components downstream need time to adapt to changes in those learned representations. In the VizDoom Take Covertask, DIP policies predict upcoming events in their environment, such as whether an approaching fireball would hit the agent, without a specific forward-prediction loss derived from reward gradients.

4.4.3 Discussion and Open Challenges

Population-based learning over multiple timescales (evolutionary and lifetime learning) allows EvoRL to effectively address the temporal credit assignment problem in RL through coevolution. First, evolutionary search effectively performs multiple parallel searches, as opposed to optimising a single model, and generally provides more exploration than gradient-decent methods [115]. Second, the distributed nature of population-based search allows the system as a whole to decompose a problem in both time and space, finding a diverse set of partial solutions. Third, cooperative coevolutionary algorithms automatically recombine partial solutions to form composite policies. In doing so, behaviours that are specialised for particular spatial and temporal regions of an RL problem can be identified (credited) and exploited within a generalised problem solver. Finally, lifetime learning—with or without the use of step-based reward gradients—is fully supported in EvoRL, often improves efficiency, and is essential in many cases (e.g., nonstationary or partially observable environments). Multiple parallel algorithms drive evolutionary learning (e.g., competitive and cooperative coevolution, quality and diversity search). Likewise, multiple *lifetime* RL systems are active in the brain. For example, different regions perform model-free and model-based control, allowing learners to switch between rapid action selection from action-values (learned via TD with dopamine signals conveying reward predictions) vs. slower (but more accurate) forward simulation of the consequences of actions using a learned model of the environment [41, 128]. Each parallel system has unique implications for temporal credit assignment. Future EvoRL research should consider how multiple parallel action inference and lifetime learning systems, operating at *many* timescales, will interact with evolutionary policy development.

4.5 Partial Observability in Space and Time

RL policies observe their environment through a sensory interface that provides a set of *state variables* at each timestep t , \overrightarrow{obs}_t . In many cases, these observations do not contain complete information required to determine the best action, i.e., the environment is only *partially observable*. For example, consider a maze navigation task in which \overrightarrow{obs}_t does not contain a global map of the maze. This is the case in many robotics tasks or in first-person Real-Time Strategy (RTS) video games. In partially observable environments, the policy must identify and store salient features of \overrightarrow{obs}_t in memory over time, encoding a representation of the environment that captures spatial and temporal properties of the current state [63]. This implies that *active perception* [131] and world modelling form a significant aspect of behaviour: constructing and managing a representation of the environment in memory. This is an example of automatic *model-based* RL, i.e., the world model is constructed on-the-fly. This property distinguishes policies with memory from purely reactive, *model-free* policies that define a direct association from sensation to action without any internal representation of state, and thus no temporal integration of experience over time. Interestingly, operation in partially observable environments also places more importance on *lifetime* exploration vs. exploitation because policies must now explore enough of the environment to gain a breadth of experience (and possibly build an internal model), while also selecting actions that satisfy their objective.

A policy representation may incorporate temporal memory in at least two ways: (1) Recursive structure; and (2) External data structures that support dynamic read/write access. The two memory models differ in their potential for operation in dynamic environments, where it is critical for the memory mechanism to support dynamic access. As discussed in detail in Sect. 4.6, many realistic RL environments are highly dynamic, with transition functions, objectives, and observable properties of the world that change over time. In this case, the memory model must be capable of reading and writing to distinct memory locations at any moment, based on the state of the environment. In effect, the memory model must represent a read-write spatiotemporal problem decomposition, or *working memory* [89, 193]. Evolution and learning can then solve problems such as how to encode information for later recall and identify what to remember in each situation. In effect, the agent is free to access multiple encoded memories with distinct time delays and integrate this information with \overrightarrow{obs}_t to predict the best output for the current situation. Purely recursive memory is potentially limited in this respect. While some memory tasks are enabled by integrating experience over time through feedback, enabling working memory through feedback is nontrivial [102, 195].

Milestone innovations in neuroevolution such as ESP [62], NEAT [156], and CoSyNN [61] established the potential to evolve recurrent networks that solved relatively simple partially observable tasks such as double pole balancing without velocity information. However, the focus of these studies is on neuroevolution properties rather than temporal memory models. Recurrent Neural Network (RNN)s have difficulty modelling temporal processes with long time delays because they can leak

information over time [73]. Also, it may be difficult to separate the action-prediction functionality of the network from the world-modelling functions [5]. Long Short-Term Memory (LSTM) neural networks introduce specific “gated” connections for explicitly learning what information to keep or forget, and are potentially less susceptible to such issues.

To overcome the difficulty RNNs have in discovering long-term and dynamic temporal dependencies, [137] extend NEAT to build LSTM networks in two phases. First, networks are pre-trained (using NEAT) by optimising a novel unsupervised information maximisation (Info-max) objective. LSTM units incrementally capture and store unique, temporally extended features from the environment during pre-training. Second, the networks are evolved to a sequence recall task, where they significantly outperform neuroevolution of RNNs, especially as the temporal depth of the sequence increases. Interestingly, optimising a policy’s *information* can indirectly lead to exploratory behaviours and resemble the concept of curiosity [141], since the policy is explicitly rewarded for exploring areas in the environment that provide new information (Curiosity is discussed further Sect. 4.7.2 in relation to hierarchical RL).

Markov Brains are evolvable neural networks that differ from conventional networks in that each unit or neuron potentially performs a unique computation [71]. As such, they combine elements from GP (i.e., program search) and neuroevolution. In tasks that require temporal memory, Markov Brains develop world models that are localised and sparsely distributed. By contrast, evolved RNNs and LSTM networks tend to *smear* information about the world over the entire network [93]. Markov brains are more robust to sensor noise, possibly due to their distributed world models, however, extensive analysis is now probing the information flow in cognitive and genetic networks to better understand *information fragmentation* in learned spatiotemporal representations [20, 21].

In GP, *functional modularity* has long been the focus of approaches attempting to simultaneously build a world model and define a policy to act by querying the model for prediction. Andre [6] proposes a multi-phase tree-GP algorithm in which each program consists of two branches, one of which will be executed in each phase of the policy’s behaviour. In the first phase, the policy explores the world, examining sensory information and possibly reading and writing information to indexed memory, but does not predict action outputs. In the second phase, the agent is blind with respect to the sensor input and must use only its stored representation to produce a multi-step plan of action. The plan is then evaluated and its fitness is determined. Fitness is only evaluated after phase 2, thus phase-specific credit assignment was not required. Over multiple evaluations in unique environments, the policy learns a general, modularised (2-phase) memory-prediction policy. A key limitation of this approach is that explicitly decomposing the program representation and evaluation procedure into separate structures and processes for world-model building and planning implied that the movement in memory is directly tied to movement in the world. With this hard-coding, the generality of the framework is limited because no spatiotemporal compression or abstraction of experience within the memory model is encouraged.

Teller [172] explored a less constrained approach to model-based policies through the use of indexed memory in GP. In this work, each program includes a memory vector of size M which is set to 0 at the start of each episode. Extending the GP function set with parameterised read/write operations allows programs to dynamically access/modify memory by vector indexing as they interact with the environment. Programs in this language, supplemented with indexed memory, can support any data structure and implement any algorithm, i.e., a genetically evolved program in this representation is Turing-complete. However, there is no straightforward heuristic to select the size of M , and simply choosing a value so large that memory space is unlikely to saturate removes pressure for evolution to find compact representations for stored information, and again potentially limits the generality of evolved mental models. Furthermore, there is no evidence that this particular representation is an *efficient* means of building memory-prediction machines. To address these issues without resorting to hard representational constraints, subsequent explorations of temporal memory for GP policies leverage some form of *emergent* functional modularity, discussed next.

Modular temporal memory structures have been explored extensively in TPG, specifically to support efficient policy search in partially observable visual RL environments such as VizDoom [89] and Dota 2 [151]. An RL policy represented by a tangle program graph is essentially a hierarchical network of programs (linear register machines [26]) constructed from the bottom up via compositional evolution (Sect. 4.3). A linear program representation has the advantage that temporal memory can be implemented simply by not clearing memory registers between program execution, thus all programs are *stateful*. The probabilistic memory framework in [89, 152] extends TPG by adding a global bank of indexed vector memory which is shared by all policies in the population. The motivation behind global shared memory is to encourage all agents to evolve a common/shared concept for what constitutes useful memory content and where to find context-dependent data. Shared memory potentially also aids in exploration since experiences from multiple unique agents are now communicated/shared globally. Programs are augmented with read/write instructions for memory access. Write operations are probabilistic, with the data being distributed across shared memory to simulate distinct long- and short-term memory regions. However, no explicit functional decomposition is enforced relative to memory management or action prediction, thus each program is free to specialise on one or the other *or* generalise to both. Furthermore, the size of indexed memory is significantly less than the size of a single observation \overrightarrow{obs}_t (screen frame), thus agents are required to select and/or compress relevant information from the observable state into a compact memory model. Empirical evaluation demonstrates that TPG agents have the capacity to develop strategies for solving a multi-faceted navigation problem that is high dimensional ($> 76,000$ state variables), partially observable, nonstationary, and requires the recall of events from long- and short-term memory. Moreover, the policies are sufficiently general to support navigation under a previously unseen environment.

Related EvoRL studies explore how to encode temporal memory directly in the tangled program graph structure [85, 90]. Rather than a global shared memory, each program’s internal *stateful* memory bank may be shared by any other program in the *same* graph (policy). This removes the need to define parameterised read/write operations or maintain external indexed memory. Instead, the graph execution now establishes the order in which local banks of memory are read from/written to. Policies in this representation are able to play the Atari breakout video game *without* the use of frame stacking (i.e., *autoregressive* state which sidesteps partial-observability)[85]. In addition, multi-task policies are evolved to solve multiple classic control tasks in which the observation space does not include velocity information, implying partial-observability. Analysing the policy’s hierarchical distributed memory reveals accurate modelling of all system velocities, with separate regions of the hierarchy contributing to modelling disjoint tasks (e.g., Cartpole and Pendulum) while similar tasks were modelled in a single region (e.g., Discrete-action Mountain Car and Continuous-action Mountain Car) [90].

4.5.1 Discussion and Open Challenges

Realistic temporal decision and control environments are only partially observable, thus learned memory models are fundamental to cognition and a critical consideration in scaling RL to complex tasks. When exposed to memory-intensive problems, EvoRL has repeatedly shown its capacity for building problem solvers with modularity in their structure and in the nature of information flow from input, through storage, to output. Developing tools to analyse information flow will increase our understanding of this functional modularity and make artificial cognitive systems more interpretable and trustworthy [70]. Analysing how different learning methods (e.g., evolution and gradient descent) differ with respect to constructing memory models (e.g., [69]) will help us leverage these models in hybrid frameworks (e.g., [67]). Furthermore, imagining new ways of evolving memory mechanisms will not only address partial-observability, but also support operation in dynamic environments (e.g., [16]), discussed in the next section.

4.6 Non-stationary and Multi-Task Environments

Autonomous agents deployed in real-world environments will face changing dynamics that require their behaviour to change over time in response. In robotics, dynamic properties such as physical wear-and-tear, as well as characteristics of the external environment such as new terrain or interaction with other agents imply that the system’s transition function, reward function, and observation space potentially change over time. In these situations, a static controller that always associates the same observation with the same action is unlikely to be optimal. Real-world agents must

be capable of continuously adapting their policy in response to a changing environment. To achieve this capability, the agent must solve two equally important subtasks in real-time: (1) Recognising a change in the environment without an external cue (\overrightarrow{obs} , contains no information about potential changes); and (2) Updating their policy in response. Both tasks must be completed without prior knowledge of the upcoming change, and fast adaptation provides increased reward. Video games are another common example of nonstationary tasks: as the player interacts with the game, non-player-characters may change their behaviour to increase the difficulty of play and new *levels* of play are periodically encountered in which the physics of the simulation may abruptly change (e.g., new entities are introduced or the perspective changes from birds-eye-view to first person) [192]. Such changes may require new skills or new internal world models. As such, adaptation and Multi-Task Reinforcement Learning (MTRL) are fundamentally linked. In addition to adaptability, algorithms operating under these conditions must contend with the following challenges [181]:

1. *Scalability.* Jointly learning N tasks should not take N times longer than learning each task individually, and the resulting multi-task agent should not be N times as complex.
2. *Distraction Dilemma.* The magnitude of each task's reward signal may be different, causing certain tasks to appear more salient than others.
3. *Catastrophic Forgetting.* When learning multiple tasks in sequence, the agent must have a mechanism to avoid unlearning task-specific behaviours that are intermittently important over time.
4. *Negative Transfer.* If the environments and objectives are similar, then simultaneously learning multiple tasks might improve the learning or search process through positive inter-task transfer. Conversely, jointly learning multiple dissimilar tasks is likely to make MTRL more difficult than approaching each task individually.
5. *Novel Environmental Dynamics.* Agents deployed in many real environments will need to adapt to dynamics not experienced during search/training.

IMPROBED [116] uses an extension of Cartesian Genetic Programming (CGP) to evolve developmental neural networks to solve multi-task problems by alleviating catastrophic forgetting in at least two primary ways: (1) By evolving networks that undergo topological changes during learning and (2) by growing numerous connections between pairs of neurons, thus lessening the influence of individual weighted connections. They point to a large body of research indicating that learning and environmental interaction are strongly related to structural changes in neurons, and the most significant period of learning in animals happens in infancy, when the brain is developing. Developmental models are an alternate approach to evolutionary incremental growth (Sect. 4.3), allowing the topology and size of computational networks to emerge rather than having to be predefined. They show how a pair of evolved programs can build a single neural network from which multiple ANNs can be developed, each of which can solve a different computational problem including two RL and two classification problems (incrementally or all at the same time). The principal limitation is that developed networks required a “problem type” input to discriminate between tasks.

TPG can evolve highly generalised policies capable of operating in 6 unique environments from the control literature [90], including OpenAI’s entire Classic Control suite [28]. This requires the policy to support discrete and continuous actions simultaneously. No task-identification inputs are provided, thus policies must identify tasks from the dynamics of state variables alone *and* define control policies for each task. Hierarchical program graphs built through compositional evolution (Sect. 4.3) support multi-task environments through automatic, hierarchical problem decomposition. Policies can recombine multiple generalist and specialist behaviours, and dynamically switch between them at inference time. This allows policies to exploit positive inter-task transfer when tasks are related, and avoid negative transfer between disjoint tasks that require specialised behaviours. The resulting policies are competitive with state-of-the-art single-task policies in all 6 environments and can adapt to any task on-the-fly, within a single episode. However in this initial study, it is assumed that populations are exposed to all task dynamics during evolution. The principal open question is how TPGs can be extended to support adaptation to environmental dynamics *not* experienced during evolutionary search.

Several approaches based on evolutionary strategies have made progress towards agents that can adapt to *unseen* tasks. SO-CMA [194] employs a transfer methodology that first learns a diverse family of policies simultaneously. In the unseen environment, it searches directly for a policy in the family that performs the best and fine-tunes it further using CMA-ES and a minimal number of samples. SO-CMA can overcome large discrepancies between seen and unseen tasks. ES-MAML [154] trains a meta-policy on a variety of tasks such that agents can adapt to new environment dynamics using only a small number of training samples. The method is demonstrated on nonstationary OpenAI Gym benchmarks. Instance weighted incremental evolution strategies (IW-IESs) proposed a method to adjust a previously learned policy to a new one incrementally whenever the environment changes, without any structural assumptions or prior knowledge on the dynamics of the ever changing environment. They incorporate an instance weighting mechanism with ES to facilitate its learning adaptation. During parameter updating, higher weights are assigned to instances that contain more new knowledge, thus encouraging the search distribution to move towards new promising areas of parameter space. IW-IES is able to handle various dynamic environments that change in terms of the reward function, the state transition function, or both in tasks ranging from robot navigation to locomotion. The principal limitation with these ES methods is adaptation speed. While they can all adapt to novel environments, their adaptation is not efficient enough to work online within a single RL episode.

Synaptic plasticity allows natural organisms to adapt to novel situations, and neuroevolution has recreated this property in artificial networks to address continual learning in nonstationary environments. [54] evolved neural networks with local synaptic plasticity based on the Hebbian rule, which strengthens the connection proportionally to correlated activation, and compared them to fixed-weight networks in a dynamic evolutionary robotics task. The policies were evolved to turn on a light, and then move to a specified location. Local learning rules helped networks quickly change their policy, transitioning from one task to another. Interestingly,

[158] evolved recurrent networks with fixed weights which can also respond differently in changing environmental conditions, and required fewer evaluations than networks evolved with local learning rules. A major question for future research is how to characterise what type of change requires semi-supervised (re)learning, and what types of change require only that the policies be *stateful*, i.e., capable of storing and/or integrating environmental readings over time. To this end, recent work combining evolution and learning suggests that unsupervised lifetime learning with synaptic plasticity is a more biologically accurate model for learning [144] and will be critical in cases where the agent is required to rapidly adapt to novel, previously unseen dynamics and goals without human intervention [153]. For instance, [126] demonstrated how evolved local learning rules allow a neural network with initially random weights to adapt to unseen environments within a single episode.

4.6.1 Discussion and Open Challenges

Humans routinely adapt to environmental characteristics during our lifetime that are too novel to have influenced evolutionary adaptation (e.g., climate change, technology such as computers, global travel and communication). Rather, evolution through natural selection produced excellent lifelong learning machines, allowing us to rapidly learn and remember how to solve new problems. EvoRL has great potential to simulate the multitude of biological systems foundational to lifelong learning, some of which are itemised in [100]. Two examples stand out: (1) Not only can we evolve brain-like algorithms, but we can evolve systems responsible for cognition and learning outside the brain such as Gene Regulatory Networks (GRNs). The potential for autonomous robotics is intriguing, since GRNs are biological systems that might control adaptive morphogenesis (e.g., growth and regeneration) *and* decision-making (See Sect. 5.1.3 of [40]); (2) Multisensory integration implies that behaviour is informed by multiple types of sensor signal. While actively studied in neuroscience, some artificial systems have not received significant attention in (Evo) RL (e.g., [167]).

4.7 Transformational Learning and Hierarchical Decomposition

RL problems often involve large, complex spaces that are difficult for agents to navigate in time or space. Two methodologies that enable more efficient learning in these settings are Transfer Learning and Hierarchical Reinforcement Learning. Rather than tackling the big, complex problem directly, these approaches decompose the overall problem into more manageable subproblems. The agent first learns to solve these simpler subproblems, the solutions of which can then be leveraged to solve the

overall, more complex task. While this is conceptually similar to the discussion of coevolution and credit assignment in Sect. 4.4.2, here we examine how this divide-and-conquer method facilitates faster learning and convergence by reducing and transforming the problem space.

4.7.1 Transfer Learning

Transfer learning in RL refers to the reuse of solvers from easier source tasks in an attempt to solve a more difficult target task [170]. Unlike multi-task learning explored in Sect. 4.6, transfer learning assumes that source and target tasks are related. As such, it is expected that learning them together (in sequence or in parallel) improves the learning or search process through positive inter-task transfer. For example, transfer learning was used heavily in the multi-month training of the Dota 2 bot, because training from scratch for every new game update pushed by the game developer would have been prohibitively costly in time and money [17]. The key issues in transfer learning can be summarised in terms of: (1) What knowledge is transferred; and (2) How to implement the knowledge transfer between tasks that may differ with respect to sensors, actions, and objectives. *Incremental Evolution* [60] and *Layered Learning* [161] have similar motivation and share the same broad issues as transfer learning. Incremental evolution assumes that the source tasks follow a sequence in which each source task is incrementally more complex than the last. For example, [23] defines a *behaviour chain* to incrementally learn dynamic legged locomotion towards an object followed by grasping, lifting, and holding of that object in a realistic three-dimensional robotics environment. In this case, a single champion individual is transferred between each task as a seed for the population on the next source task. Thus, variation operators applied to the genotype of this champion provide a starting point for generating the next population. Variations on this approach pass the entire population between consecutive tasks (e.g., [3]) or transfer multiple champion individuals [76]. One potential issue is that an evolutionarily good sequence of source tasks can be quite unintuitive, making the utility of the human-designed source tasks unpredictable or even detrimental [59]. Rather than assume that source tasks represent a sequence of explicitly related problems of incremental complexity, Layered Learning [159] may define source tasks that are independent. For example, in the keep-away soccer game, source tasks may include independent behaviours for *get open*, *pass*, and *intercept*. Each task represents a separate training environment with unique states and actions. Transfer is accomplished through an evolved switching policy that joins independent solvers for each source task [86, 186].

Finally, transfer learning must also consider what happens when the sensor and action spaces have altered meaning or dimensionality between source and target tasks. A learned inter-task mapping is one method of addressing this issue [171]. In addition, Hyper-NEAT has been proposed as a way to evolve generalised state representations such that a policy can transfer to more complex environments without additional learning, assuming the objective remains the same [179]. In both

incremental and layered learning, entire (autonomous) policies are transferred from the source task to target task, where they either continue adapting under the more challenging task environment (incremental evolution) or are reused as-is within a hierarchical switching policy (layered learning).

4.7.2 Hierarchical Reinforcement Learning

The principal goal of Hierarchical Reinforcement Learning (HRL) is to allow sequential decision policies to reason at levels of abstraction above the raw sensory inputs and atomic actions assumed by the policy-environment interface. *Temporal abstraction* implies that the policy can encapsulate a sequence of actions as a single abstract action, or option [164]. Options are typically associated with specific sub-goals. A higher level decision-maker is now required in order to map environmental situations to options. Once identified, the selected option assumes control until its (sub-)goal state is reached. If a suitable task decomposition is known *a priori*, then multiple options can be trained individually to solve the (human-specified) sub-goals and then reused within a fixed hierarchy to solve a more complex task [45]. However, sub-goals can also be identified automatically, leading to the automated discovery of options and potentially also learning the hierarchical policy [114].

Recent research has emphasised temporal abstraction in reinforcement learning, specifically the importance of automatically discovering sub-goals and options. This implies the emergence of hierarchical structure [25]. In this perspective, HRL is framed more similarly to learning in biological organisms considering both brain and behaviour.

From a psychological perspective, the role of intrinsic motivation, or curiosity [140] in motivating an agent's exploration of action-sequences not explicitly rewarded by the environment has been suggested as a way of kickstarting option discovery. Indeed, explicitly rewarding *surprise* (Sect. 4.4.1) and the learning opportunities, or *salient events*, that result has shown promise when implemented within HRL [101]. Evolution is also assumed to play a role in the development of hierarchically organised behaviour in animals [65, 148], and several hybrid TD/EA methods have appeared in the HRL literature. For example, evolutionary methods have been proposed that search for the useful (intermediate) reward functions in order for TD methods to reach the overall goal more efficiently, or *shaping rewards* [52, 127]. [51] also demonstrate that once subtasks have been defined and learned using TD methods, GP can be employed to evolve option hierarchies.

4.7.3 Discussion and Open Challenges

HRL can be characterised as performing search over two levels: (1) Search in a problem space; and (2) (Meta-)Search of a problem space through abstraction. [35] relate this directly to the Creative Systems Framework [189]:

The first level of search achieves exploratory creativity, such as the analytic discovery of new theorems from axioms, or of a control policy to achieve a task. The second level achieves the more elusive transformational creativity, which consists of a radical change of the domain being investigated.

Taken together, a bi-level search of this nature may facilitate *insight*, a fundamental aspect of creative problem-solving in which an individual arrives at a solution through a sudden change in perspective (an “Aha!” moment). Critical steps towards computational creativity and insight, namely accumulation of experience, meta-level thinking through abstraction [180], and transfer [30] can be leveraged as engineering design principals for scaling RL algorithms to the most complex tasks. EvoRL, as a meta-search paradigm, is uniquely placed to capture these properties. Existing studies achieve sample-efficient learning and good performance on sparse-reward tasks [1, 112], but more work is needed. Among many open challenges, [35] identify automatic option discovery and lifelong *hierarchical* learning as important areas for future research.

4.8 Conclusion

RL is characterised by a number of challenges that all relate to the temporal nature of trial-and-error problem-solving. EvoRL is a powerful meta-search paradigm that has the potential to combine the best of population-based search with individual, lifetime development and temporal-difference learning. This chapter has reviewed many examples in which EvoRL efficiently builds sequential decision agents that are themselves efficient lifelong learning machines, capable of problem-solving even when the observable space is large and incomplete, error signals are delayed, and environment dynamics are continuously changing in unpredictable ways. EvoRL is especially effective when little prior information is available regarding *how* to solve a problem. It can search in program space and parameter space, and can thus be employed to learn parameters for a predefined policy, directly build policies from scratch, or search for entire RL frameworks by coevolving policies, environments, and their interaction dynamics. With this flexibility comes a massive search space, and thus EvoRL frameworks require careful guidance towards the most promising regions. Future work could form this guidance by observing the hardware and software of the brain, the creative process, and their evolution in the natural world.

References

1. Abramowitz, S., Nitschke, G.: Scalable evolutionary hierarchical reinforcement learning. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO '22, pp. 272–275. Association for Computing Machinery, New York, NY, USA (2022)
2. Adami, C.: Making artificial brains: Components, topology, and optimization. *Artif. Life* **28**(1), 157–166 (2022)
3. Agapitos, A., Togelius, J., Lucas, S.M.: Evolving controllers for simulated car racing using object oriented genetic programming. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, GECCO '07, pp. 1543–1550. Association for Computing Machinery, New York, NY, USA (2007)
4. Agogino, A., Turner, K.: Efficient evaluation functions for evolving coordination. *Evol. Comput.* **16**(2), 257–288 (2008)
5. Al Masalma, M., Heywood, M.: Benchmarking ensemble genetic programming with a linked list external memory on scalable partially observable tasks. *Genet. Program. Evolvable Mach.* **23**(Suppl 1), 1–29 (2022)
6. Andre, D.: Evolution of mapmaking: learning, planning, and memory using genetic programming. In: Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence, vol. 1, pp. 250–255 (1994)
7. Andre, D., Russell, S.J.: State abstraction for programmable reinforcement learning agents. In: Eighteenth National Conference on Artificial Intelligence, pp. 119–125. American Association for Artificial Intelligence, USA (2002)
8. Badcock, P.B., et al.: The hierarchically mechanistic mind: An evolutionary systems theory of the human brain, cognition, and behavior. *Cognitive Affect. Behav. Neurosci.* **19**(6), 1319–1351 (2019)
9. Bai, H., Cheng, R., Jin, Y.: Evolutionary reinforcement learning: A survey. *Intell. Comput.* **2**, 0025 (2023)
10. Bai, H., Shen, R., Lin, Y., Xu, B., Cheng, R.: Lamarckian platform: Pushing the boundaries of evolutionary reinforcement learning towards asynchronous commercial games. *IEEE Trans. Games* 1–14 (2022)
11. Baldwin, J.M.: A new factor in evolution. In: Adaptive Individuals in Evolving Populations: Models and Algorithms, pp. 59–80 (1896)
12. Banzhaf, W., et al.: Defining and simulating open-ended novelty: Requirements, guidelines, and challenges. *Theory Biosci.* **135**(3), 131–161 (2016)
13. Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.D.: Genetic Programming: An Introduction: On the Automatic Evolution of Computer Programs and its Applications. Morgan Kaufmann Publishers Inc. (1998)
14. Barreto, A.M.S., Augusto, D.A., Barbosa, H.J.C.: On the characteristics of sequential decision problems and their impact on evolutionary computation and reinforcement learning. In: Pierre, C., Nicolas, M., Pierrick, L., Marc, S., Evelyne, L. (eds.) *Artificial Evolution*, pp. 194–205. Springer, Berlin (2010)
15. Bedau, M.A.: Artificial life. In: Matthen, M., Stephens, C. (eds.) *Philosophy of Biology*. Handbook of the Philosophy of Science, pp. 585–603. North-Holland, Amsterdam (2007)
16. Ben-Iwhiwhu, E., Ladosz, P., Dick, J., Chen, W-H., Pilly, P., Soltoggio, A.: Evolving inborn knowledge for fast adaptation in dynamic pomdp problems. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference, GECCO '20, pp. 280–288. Association for Computing Machinery, New York, NY, USA (2020)
17. Berner, C., Brockman, G., Chan, B., Cheung, V., Debiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al.: Dota 2 with large scale deep reinforcement learning. ArXiv preprint [arXiv:1912.06680](https://arxiv.org/abs/1912.06680) (2019)
18. Bishop, C.M.: Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag, Berlin (2006)
19. Boden, M.A.: Creative Mind: Myths and Mechanisms, 2nd edn. Routledge, USA (2003)

20. Bohm, C., Kirkpatrick, D., Cao, V., Adami, C.: Information fragmentation, encryption and information flow in complex biological networks. *Entropy* **24**(5) (2022)
21. Bohm, C., Kirkpatrick, D., Hintze, A.: Understanding memories of the past in the context of different complex neural network architectures. *Neural Comput.* **34**(3), 754–780 (2022)
22. Bolles, R.C., Beecher, M.D.: Evolution and Learning. Psychology Press (2013)
23. Bongard, J.: Behavior chaining: Incremental behavior integration for evolutionary robotics. *Artif. Life* **11**(64), 01 (2008)
24. Bonson, J.P.C., Kelly, S., McIntyre, A.R., Heywood, M.I.: On synergies between diversity and task decomposition in constructing complex systems with gp. In: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion, GECCO '16 Companion, pp. 969–976. Association for Computing Machinery, New York, NY, USA (2016)
25. Botvinick, M.M., Niv, Y., Barto, A.G.: Hierarchically organized behavior and its neural foundations: A reinforcement learning perspective. *Cognition* **113**(3), 262–280 (2009) Reinforcement learning and higher cognition
26. Brameier, M., Banzhaf, W.: Linear Genetic Programming. Springer (2007)
27. Brant, J.C., Stanley, K.O.: Minimal criterion coevolution: A new approach to open-ended search. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '17, pp. 67–74. Association for Computing Machinery, New York, NY, USA (2017)
28. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: OpenAI Gym. arXiv **1606**, 01540 (2016)
29. Brooks, R.A.: Intelligence without representation. *Artif. Intell.* **47**(1), 139–159 (1991)
30. Buchanan, B.G.: Creativity at the metalevel: Aaaai-2000 presidential address. *AI Mag.* **22**(3), 13 (2001)
31. Cartlidge, J., Bullock, S.: Combating coevolutionary disengagement by reducing parasite virulence. *Evol. Comput.* **12**(2), 193–222 (2004)
32. Chong, S.Y., Tino, P., Yao, X.: Relationship between generalization and diversity in coevolutionary learning. *IEEE Trans. Comput. Intell. AI Games* **1**(3), 214–232 (2009)
33. Co-Reyes, J.D., Miao, Y., Peng, D., Real, E., Le, Q.V., Levine, S., Lee, H., Faust, A.: Evolving reinforcement learning algorithms. In: International Conference on Learning Representations (2021)
34. Colas, C., Madhavan, V., Huizinga, J., Clune, J.: Scaling map-elites to deep neuroevolution. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference, GECCO '20, pp. 67–75. Association for Computing Machinery, New York, NY, USA (2020)
35. Colin, T.R., Belpaeme, T., Cangelosi, A., Hemon, N.: Hierarchical reinforcement learning as creative problem solving. *Robot. Auton. Syst.* **86**, 196–206 (2016)
36. Conti, E., Madhavan, V., Such, F.P., Lehman, J., Stanley, K.O., Clune, J.: Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems, NIPS'18, pp. 5032–5043. Curran Associates Inc, Red Hook, NY, USA (2018)
37. Crary, C., Piard, W., Stitt, G., Bean, C., Hicks, B.: Using fpga devices to accelerate tree-based genetic programming: A preliminary exploration with recent technologies. In: Gisele, P., Mario, G., Zdenek, V. (eds.) Genetic Programming, pp. 182–197. Springer Nature Switzerland, Cham (2023)
38. Cuccu, G., Gomez, F.: When novelty is not enough. In: Proceedings of the 2011 International Conference on Applications of Evolutionary Computation—Volume Part I, EvoApplications'11, pp. 234–243. Springer-Verlag, Berlin (2011)
39. Cully, A., Clune, J., Tarapore, D., Mouret, J.-B.: Robots that can adapt like animals. *Nature* **521**(7553), 503–507 (2015)
40. Cussat-Blanc, S., Harrington, K., Banzhaf, W.: Artificial gene regulatory networks—a review. *Artif. Life* **24**(4), 296–328 (2019)
41. Daw, N.D., Niv, Y., Dayan, P.: Uncertainty-based competition between prefrontal and dorsolateral striatal systems for behavioral control. *Nat. Neurosci.* **8**(12), 1704–1711 (2005)

42. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Trans. Evol. Comput.* **6**(2), 182–197 (2002)
43. Desnos, K., Bourgoin, T., Dardaillon, M., Sourbier, N., Gesny, O., Pelcat, M.: Ultra-fast machine learning inference through c code generation for tangled program graphs. In: 2022 IEEE Workshop on Signal Processing Systems (SiPS), pp. 1–6 (2022)
44. Desnos, K., Sourbier, N., Raumer, P.Y., Gesny, O., Pelcat, M.: Gegelati: Lightweight artificial intelligence through generic and evolvable tangled program graphs. In: Workshop on Design and Architectures for Signal and Image Processing (DASIP), International Conference Proceedings Series (ICPS). ACM, Budapest, Hungary (2021)
45. Dietterich, T.G.: Hierarchical reinforcement learning with the maxq value function decomposition. *J. Artif. Int. Res.* **13**(1), 227–303 (2000)
46. Doncieux, S., Mouret, J.-B.: Behavioral diversity with multiple behavioral distances. In: 2013 IEEE Congress on Evolutionary Computation, pp. 1427–1434 (2013)
47. Doncieux, S., Paolo, G., Laflaquière, A., Coninx, A.: Novelty search makes evolvability inevitable. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference, GECCO '20, pp. 85–93. Association for Computing Machinery, New York, NY, USA (2020)
48. Doucette, J.A., Lichodziewski, P., Heywood, M.I.: Hierarchical task decomposition through symbiosis in reinforcement learning. In: Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation, GECCO '12, pp. 97–104. Association for Computing Machinery, New York, NY, USA (2012)
49. Dyar, H.G.: Stated meeting. *Trans. New York Acad. Sci.* **15**, 141–143 (1896)
50. Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K.O., Clune, J.: First return, then explore. *Nature* **590**(7847), 580–586 (2021)
51. Elfwing, S., Uchibe, E., Doya, K., Christensen, H.I.: Evolutionary development of hierarchical learning structures. *IEEE Trans. Evol. Comput.* **11**(2), 249–264 (2007)
52. Elfwing, S., Uchibe, E., Doya, K., Christensen, H.I.: Co-evolution of shaping rewards and meta-parameters in reinforcement learning. *Adapt. Behav.* **16**(6), 400–412 (2008)
53. Fedus, W., Zoph, B., Shazeer, N.: Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *J. Mach. Learn. Res.* **23**(1) (2022)
54. Floreano, D., Urzelai, J.: Evolutionary robots with on-line self-organization and behavioral fitness. *Neural Networks* **13**(4), 431–443 (2000)
55. Franke, J.K.H., Köhler, G., Biedenkapp, A., Hutter, F.: Sample-efficient automated deep reinforcement learning. In: International Conference on Learning Representations (2021)
56. Freeman, C.D., Frey, E., Raichuk, A., Girgin, S., Mordatch, I., Bachem, O.: Brax—a differentiable physics engine for large scale rigid body simulation. ArXiv preprint [arXiv:2106.13281](https://arxiv.org/abs/2106.13281) (2021)
57. Gesmundo, A., Dean, J.: Munet: Evolving Pretrained Deep Neural Networks into Scalable Auto-tuning Multitask Systems (2022)
58. Goldberg, D.E., Richardson, J.: Genetic algorithms with sharing for multimodal function optimization. In: Proceedings of the Second International Conference on Genetic Algorithms on Genetic Algorithms and Their Application, pp. 41–49. L. Erlbaum Associates Inc, USA (1987)
59. Goldsby, H.J., Young, R.L., Schossau, J., Hofmann, H.A., Hintze, A.: Serendipitous scaffolding to improve a genetic algorithm's speed and quality. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 959–966 (2018)
60. Gomez, F., Miikkulainen, R.: Incremental evolution of complex general behavior. *Adapt. Behav.* **5**(3–4), 317–342 (1997)
61. Gomez, F., Schmidhuber, J., Miikkulainen, R.: Accelerated neural evolution through cooperatively coevolved synapses. *J. Mach. Learn. Res.* **9**, 937–965 (2008)
62. Gomez, F.J., Miikkulainen, R.: Solving non-markovian control tasks with neuroevolution. In: Proceedings of the 16th International Joint Conference on Artificial Intelligence—Volume 2, IJCAI'99, pp. 1356–1361. Morgan Kaufmann Publishers Inc, San Francisco, CA, USA (1999)

63. Gomez, F.J., Schmidhuber, J.: Co-evolving recurrent neurons learn deep memory pomdps. In: Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation, GECCO '05, pp. 491–498. ACM, New York, NY, USA (2005)
64. Gravina, D., Liapis, A., Yannakakis, G.N.: Quality diversity through surprise. *IEEE Trans. Evol. Comput.* **23**(4), 603–616 (2019)
65. Greenfield, P.M.: Language, tools and brain: The ontogeny and phylogeny of hierarchically organized sequential behavior. *Behav. Brain Sci.* **14**(4), 531–551 (1991)
66. Gupta, A., Savarese, S., Ganguli, S., Fei-Fei, L.: Embodied intelligence via learning and evolution. *Nat. Commun.* **12**(1), 5721 (2021)
67. Ha, D., Schmidhuber, J.: Recurrent world models facilitate policy evolution. In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) *Advances in Neural Information Processing Systems*, vol. 31. Curran Associates, Inc. (2018)
68. Hawkins, J., Ahmad, S., Cui, Y.: A theory of how columns in the neocortex enable learning the structure of the world. *Front. Neural Circuits* **11**, 81 (2017)
69. Hintze, A., Adami, C.: Neuroevolution gives rise to more focused information transfer compared to backpropagation in recurrent neural networks. In: *Neural Computing and Applications* (2022)
70. Hintze, A., Adami, C.: Detecting information relays in deep neural networks. *Entropy* **25**(3) (2023)
71. Hintze, A., Edlund, J.A., Olson, R.S., Knoester, D.B., Schossau, J., Albantakis, L., Tehrani-Saleh, A., Kvam, P.D., Shememan, L., Goldsby, H., Bohm, C., Adami, C.: Markov brains: A technical introduction. *CoRR arXiv:abs/1709.05601* (2017)
72. Hintze, A., Schossau, J.: Towards an fpga accelerator for markov brains. In: *Artificial Life Conference Proceedings* 34, vol. 2022, p. 34. MIT Press One Rogers Street, Cambridge, MA 02142–1209, USA (2022)
73. Hochreiter, S., Bengio, Y., Frasconi, P., Schmidhuber, J., et al.: Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies, pp. 237–243. In: *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE Press In (2001)
74. Holland, J.H.: Properties of the bucket brigade. In: *Proceedings of the 1st International Conference on Genetic Algorithms*, pp. 1–7. L. Erlbaum Associates Inc, USA (1985)
75. Holland, J.H.: *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. MIT Press, Cambridge, MA, USA (1992)
76. Hsu, W.H., Harmon, S.J., Rodríguez, E., Zhong, C.A.: Empirical comparison of incremental learning strategies for genetic programming-based keep-away soccer agents. In: *Papers from the 2004 AAAI Fall Symposium* (2004)
77. Hu, J., Erik, G., Kisung, S., Zhun, F., Rondal, R.: The hierarchical fair competition (hfc) framework for sustainable evolutionary algorithms. *Evol. Comput.* **13**(2), 241–277 (2005)
78. Huang, B., Cheng, R., Jin, Y., Tan, K.C.: A distributed gpu-accelerated library towards scalable evolutionary computation. *EvoX* (2023)
79. Jaderberg, M., Czarnecki, W.M., Dunning, I., Marrs, L., Lever, G., Castañeda, A.G., Beattie, C., Rabinowitz, N.C., Morcos, A.S., Ruderman, A., Sonnerat, N., Green, T., Deason, L., Leibo, J.Z., Silver, D., Hassabis, D., Kavukcuoglu, K., Graepel, T.: Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science* **364**(6443), 859–865 (2019)
80. Jain, A., Mehrotra, A., Rewariya, A., Kumar, S.: A systematic study of deep q-networks and its variations. In: *2022 2nd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*, pp. 2157–2162 (2022)
81. Jin, C., Allen-Zhu, Z., Bubeck, S., Jordan, M.I.: Is q-learning provably efficient? In: Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., Garnett, R. (eds.) *Advances in Neural Information Processing Systems*, vol. 31. Curran Associates, Inc. (2018)
82. Jong, N.K., Stone, P.: State abstraction discovery from irrelevant state variables. In: *Proceedings of the 19th International Joint Conference on Artificial Intelligence, IJCAI'05*, pp. 752–757. Morgan Kaufmann Publishers Inc, San Francisco, CA, USA (2005)

83. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: A survey. *J. Artif. Int. Res.* **4**(1), 237–285 (1996)
84. Kashtan, N., Noor, E., Alon, U.: Varying environments can speed up evolution. *Proceed. Nat. Acad. Sci.* **104**(34), 13711–13716 (2007)
85. Kelly, S., Banzhaf, W.: Temporal Memory Sharing in Visual Reinforcement Learning, pp. 101–119. Springer International Publishing, Cham (2020)
86. Kelly, S., Heywood, M.I.: Discovering agent behaviors through code reuse: Examples from half-field offense and ms. pac-man. *IEEE Trans. Games* **10**(2), 195–208 (2018)
87. Kelly, S., Heywood, M.I.: Emergent solutions to high-dimensional multitask reinforcement learning. *Evol. Comput.* **26**(3), 347–380 (2018)
88. Kelly, S., Smith, R.J., Heywood, M.I.: Emergent Policy Discovery for Visual Reinforcement Learning Through Tangled Program Graphs: A Tutorial, pp. 37–57. Springer International Publishing, Cham (2019)
89. Kelly, S., Smith, R.J., Heywood, M.I., Banzhaf, W.: Emergent tangled program graphs in partially observable recursive forecasting and vizdoom navigation tasks. *ACM Trans. Evol. Learn. Optim.* **1**(3) (2021)
90. Kelly, S., Voegerl, T., Banzhaf, W., Gondro, C.: Evolving hierarchical memory-prediction machines in multi-task reinforcement learning. *Genet. Program. Evol. Mach.* **22**(4), 573–605 (2021)
91. Khadka, S., Tumer, K.: Evolution-guided policy gradient in reinforcement learning. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems, NIPS’18, pp. 1196–1208. Curran Associates Inc, Red Hook, NY, USA (2018)
92. Kingma, D.P., Ba, J.: A Method for Stochastic Optimization, Adam (2017)
93. Kirkpatrick, D., Hintze, A.: The role of ambient noise in the evolution of robust mental representations in cognitive systems. In: ALIFE 2019: The 2019 Conference on Artificial Life, pp. 432–439. MIT Press (2019)
94. Knudson, M., Tumer, K.: Coevolution of heterogeneous multi-robot teams. In: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, GECCO ’10, pp. 127–134. Association for Computing Machinery, New York, NY, USA (2010)
95. Kohl, N., Miikkulainen, R.: Evolving neural networks for strategic decision-making problems. *Neural Networks* **22**(3), 326–337 (2009). Goal-Directed Neural Systems
96. Konda, V., Tsitsiklis, J.: Actor-critic algorithms. In: Solla, S., Leen, T., Müller, K. (eds.) Advances in Neural Information Processing Systems, vol. 12. MIT Press (1999)
97. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press (1992)
98. Koza, J.R., Andre, D., Bennett, F.H., Keane, M.A.: Genetic Programming III: Darwinian Invention & Problem Solving. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1999)
99. Koza, J.R., Goldberg, D.E., Fogel, D.B., Riolo, R.L.: Programming, Genetic: Proceedings of the First Annual Conference, vol. 7, July 28–31 (1996). Stanford University. The MIT Press (1996)
100. Kudithipudi, D., Mario, A-S., Jonathan, B., Maxim, B., Douglas, B., Josh, B., Andrew, P.B., Suraj, C.R., Nick, C., Jeff, C., Anurag, D., Stefano, F., Peter, H., Leslie, K., Nicholas, K., Zsolt, K., Soheil, K., Jeffrey, L.K., Sam, K., Michael, L., Sandeep, M., Santosh, M., Ali, M., Bruce, M., Risto, M., Zaneta, N., Tej, P., Alice, P., Praveen, K.P., Sebastian, R., Terrence, J.S., Andrea, S., Nicholas, S., Andreas, S., Tolias, D.U., Francisco, J.V-C., Gido, M.V., Joshua, T., Vogelstein, F.W., Ron, W., Angel, Y-G., Xinyun, Z., Hava, S.: Biological underpinnings for lifelong learning machines. *Nat. Mach. Intell.* **4**(3), 196–210 (2022)
101. Kulkarni, T.D., Narasimhan, K., Saeedi, A., Tenenbaum, J.: Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In: Lee, D., Sugiyama, M., Luxburg, U., Guyon, I., Garnett, R. (eds.) Advances in Neural Information Processing Systems, vol. 29. Curran Associates, Inc. (2016)
102. Landi, F., Baraldi, L., Cornia, M., Cucchiara, R.: Working memory connections for lstm. *Neural Networks* **144**, 334–341 (2021)

103. Lange, R.T.: evosax: Jax-based evolution strategies. ArXiv preprint [arXiv:2212.04180](https://arxiv.org/abs/2212.04180) (2022)
104. Lee, K., Lee, B-U., Shin, U., Kweon, I.S.: An efficient asynchronous method for integrating evolutionary and gradient-based policy search. In: Proceedings of the 34th International Conference on Neural Information Processing Systems, NIPS'20. Curran Associates Inc, Red Hook, NY, USA (2020)
105. Lehman, J., et al.: The surprising creativity of digital evolution: A collection of anecdotes from the evolutionary computation and artificial life research communities. *Artif. Life* **26**(2), 274–306 (2020)
106. Lehman, J., Stanley, K.O.: Abandoning objectives: Evolution through the search for novelty alone. *Evol. Comput.* **19**(2), 189–223 (2011)
107. Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Goldberg, K., Gonzalez, J., Jordan, M., Stoica, I.: RLlib: Abstractions for distributed reinforcement learning. In: Jennifer, D., Andreas, K., (eds.) Proceedings of the 35th International Conference on Machine Learning, vol. 80 of Proceedings of Machine Learning Research, pp. 3053–3062. PMLR, 10–15 (2018)
108. Lim, B., Allard, M., Grillotti, L., Cully, A.: Accelerated quality-diversity for robotics through massive parallelism. arXiv preprint [arXiv:2202.01258](https://arxiv.org/abs/2202.01258) (2022)
109. Lim, S.L., Bentley, P.J.: The “agent-based modeling for human behavior” special issue. *Artif. Life* **29**(1), 1–2 (2023)
110. Lin, Q., Liu, H., Sengupta, B.: Switch Trajectory Transformer with Distributional Value Approximation for Multi-task Reinforcement Learning (2022)
111. Liu, S., Lever, G., Merel, J., Tunyasuvunakool, S., Heess, N., Graepel, T.: Emergent coordination through competition. In: 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6–9, 2019. OpenReview.net (2019)
112. Long, Q., Zhou, Z., Gupta, A., Fang, F., Wu, Y., Wang, X.: Evolutionary population curriculum for scaling multi-agent reinforcement learning. In: 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26–30, 2020. OpenReview.net (2020)
113. Marchesini, E., Corsi, D., Farinelli, A.: Genetic soft updates for policy evolution in deep reinforcement learning. In: International Conference on Learning Representations (2021)
114. McGovern, E.A., Barto, A.G.: Autonomous Discovery of Temporal Abstractions from Interaction with an Environment. PhD thesis, University of Massachusetts Amherst (2002). AAI3056259
115. Miikkulainen, R.: Creative ai through evolutionary computation: Principles and examples. *SN Comput. Sci.* **2**(3), 163 (2021)
116. Miller, J.F.: IMPROBED: Multiple problem-solving brain via evolved developmental programs. *Artif. Life* **27**(3–4), 300–335 (2022)
117. Mnih, V., Puigdomènech Badia, A., Mirza, M., Graves, A., Harley, T., Lillicrap, T.P., Silver, D., Kavukcuoglu, K.: Asynchronous methods for deep reinforcement learning. In: Proceedings of the 33rd International Conference on International Conference on Machine Learning—Volume 48, ICML’16, pp. 1928–1937. JMLR.org (2016)
118. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. *Nature* **518**(7540), 529–533 (2015)
119. Morgan, C.L.: Habit and Instinct. E. Arnold (1896)
120. Moriarty, D.E., Miikkulainen, R.: Forming neural networks through efficient and adaptive coevolution. *Evol. Comput.* **5**(4), 373–399 (1997)
121. Moriarty, D.E., Schultz, A.C., Grefenstette, J.J.: Evolutionary algorithms for reinforcement learning. *J. Artif. Int. Res.* **11**(1), 241–276 (1999)
122. Mountcastle, V.B.: The columnar organization of the neocortex. *Brain* **120**, 701–722 (1997)
123. Mouret, J.B., Doncieux, S.: Encouraging behavioral diversity in evolutionary robotics: An empirical study. *Evol. Comput.* **20**(1), 91–133 (2012)
124. Mouret, J.-B., Clune, J.: Illuminating search spaces by mapping elites. CoRR [arXiv:1504.04909](https://arxiv.org/abs/1504.04909) (2015)

125. Mouret, J.-B., Doncieux, S.: Using behavioral exploration objectives to solve deceptive problems in neuro-evolution. In: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, GECCO '09, pp. 627–634. ACM (2009)
126. Najarro, E., Risi, S.: Meta-Learning through hebbian plasticity in random networks. In: Advances in Neural Information Processing Systems (2020)
127. Niekum, S., Barto, A.G., Spector, L.: Genetic programming for reward function search. *IEEE Trans. Autonom. Mental Develop.* **2**(2), 83–90 (2010)
128. Niv, Y.: Reinforcement learning in the brain. *J. Math. Psychol.* **53**(3), 139–154 (2009). Special Issue: Dynamic Decision Making
129. Noble, J., Watson, R.A.: Pareto coevolution: Using performance against coevolved opponents in a game as dimensions for pareto selection. In: Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation, GECCO'01, pp. 493–500. Morgan Kaufmann Publishers Inc, San Francisco, CA, USA (2001)
130. Nordin, P., Banzhaf, W., Brämeier, M.: Evolution of a world model for a miniature robot using genetic programming. *Robot. Autonom. Syst.* **25**, 105–116 (1998)
131. Oh, J., Chockalingam, V., Singh, S., Lee, H.: Control of Memory, Active Perception, and Action in Minecraft. ArXiv [arXiv:1605.09128](https://arxiv.org/abs/1605.09128) (2016)
132. Papavasileiou, E., Cornelis, J., Jansen, B.: A Systematic Literature Review of the Successors of “NeuroEvolution of Augmenting Topologies.” *Evol. Comput.* **29**(1), 1–73 (2021)
133. Parter, M., Kashtan, N., Alon, U.: Facilitated variation: How evolution learns from past environments to generalize to new environments. *PLOS Comput. Biol.* **4**(11), 1–15 (2008)
134. Peters, J., Schaal, S.: Natural actor-critic. *Neurocomputing* **71**(7), 1180–1190 (2008). Progress in Modeling, Theory, and Application of Computational Intelligence
135. Pollack, J.B., Blair, A.D.: Co-evolution in the successful learning of backgammon strategy. *Mach. Learn.* **32**(3), 225–240 (1998)
136. Pourchot, S.: CEM-RL: Combining evolutionary and gradient-based methods for policy search. In: International Conference on Learning Representations (2019)
137. Rawal, A., Miikkulainen, R.: Evolving deep lstm-based memory networks using an information maximization objective. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016, GECCO '16, pp. 501–508. Association for Computing Machinery, New York, NY, USA (2016)
138. Risi, S., Stanley, K.O.: Deep innovation protection: Confronting the credit assignment problem in training heterogeneous neural architectures. Proceed. AAAI Conf. Artif. Intell. **35**(14), 12391–12399 (2021)
139. Salimans, T., Ho, J., Chen, X., Sidor, S., Sutskever, I.: Evolution strategies as a scalable alternative to reinforcement learning (2017)
140. Schmidhuber, J.: Curious model-building control systems. In: Proceedings 1991 IEEE International Joint Conference on Neural Networks, vol.2, pp. 1458–1463 (1991)
141. Schmidhuber, J.: Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE Trans. Autonom. Mental Develop.* **2**(3), 230–247 (2010)
142. Schossau, J., Shirmohammadi, B., Hintze, A.: Incentivising cooperation by rewarding the weakest member. ArXiv preprint [arXiv:2212.00119](https://arxiv.org/abs/2212.00119) (2022)
143. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. CoRR [arXiv:1707.06347](https://arxiv.org/abs/1707.06347) (2017)
144. Sheneman, L., Hintze, A.: Evolving autonomous learning in cognitive networks. *Sci. Rep.* **7**(1), 16712 (2017)
145. Sigaud, O.: Combining evolution and deep reinforcement learning for policy search: A survey. *ACM Trans. Evol. Learn. Optim.* (2022) Just Accepted
146. Silverman, B.: The phantom fish tank: An ecology of mind. Montreal, Logo Computer Systems (1987)
147. Simione, L., Nolfi, S.: Achieving long-term progress in competitive co-evolution. In: 2017 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 1–8 (2017)
148. Singh, S., Lewis, R.L., Barto, A.G., Sorg, J.: Intrinsically motivated reinforcement learning: An evolutionary perspective. *IEEE Trans. Autonom. Mental Develop.* **2**(2), 70–82 (2010)

149. Skinner, B.F.: The Behavior of Organisms. Appleton-Century-Crofts, New York, NY (1938)
150. Smith, J.M.: Group selection and kin selection. *Nature* **201**(4924), 1145–1147 (1964)
151. Smith, R.J., Heywood, M.I.: Evolving dota 2 shadow fiend bots using genetic programming with external memory. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '19, pp. 179–187. Association for Computing Machinery, New York, NY, USA (2019)
152. Smith, R.J., Heywood, M.I.: Evolving a Dota 2 Hero Bot with a Probabilistic Shared Memory Model, pp. 345–366. Springer International Publishing, Cham (2020)
153. Soltoggio, A., Stanley, K.O., Risi, S.: Born to learn: The inspiration, progress, and future of evolved plastic artificial neural networks. *Neural Networks* **108**, 48–67 (2018)
154. Song, X., Gao, W., Yang, Y., Choromanski, K., Pacchiano, A., Tang, Y.: Es-maml: Simple hessian-free meta learning. In: International Conference on Learning Representations (2020)
155. Stanley, K.O., Clune, J., Lehman, J., Miikkulainen, R.: Designing neural networks through neuroevolution. *Nat. Mach. Intell.* **1**(1), 24–35 (2019)
156. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. *Evol. Comput.* **10**(2), 99–127 (2002)
157. Stanley, K.O., Miikkulainen, R.: Competitive coevolution through evolutionary complexification. *J. Artif. Int. Res.* **21**(1), 63–100 (2004)
158. Stanley, K.O., Bryant, B.D., Miikkulainen, R.: Evolving adaptive neural networks with and without adaptive synapses. In: The 2003 Congress on Evolutionary Computation, 2003. CEC '03, vol. 4, pp. 2557–2564 (2003)
159. Stone, P.: Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer. MIT Press, Cambridge, MA, USA (2000)
160. Stone, P.: Learning and multiagent reasoning for autonomous agents. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI'07, pp. 13–30. Morgan Kaufmann Publishers Inc, San Francisco, CA, USA (2007)
161. Stone, P.H., Veloso, M.M.: Layered Learning in Multiagent Systems. PhD thesis, Carnegie Mellon University, USA (1998). AAI9918612
162. Suri, K., Qi Shi, X., Plataniotis, K.N., Lawryshyn, Y.A.: Evolve to control: Evolution-based soft actor-critic for scalable reinforcement learning. *CoRR*, [arXiv:2007.13690](https://arxiv.org/abs/2007.13690) (2020)
163. Sutton, R.R., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press (1998)
164. Sutton, R.S., Precup, D., Singh, S.: Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.* **112**(1), 181–211 (1999)
165. Szubert, M., Jaskowski, W., Krawiec, K.: Coevolutionary temporal difference learning for othello. In: 2009 IEEE Symposium on Computational Intelligence and Games, pp. 104–111 (2009)
166. Szubert, M., Jaśkowski, W., Krawiec, K.: On scalability, generalization, and hybridization of coevolutionary learning: A case study for othello. *IEEE Trans. Comput. Intell. AI Games* **5**(3), 214–226 (2013)
167. Tan, H., Zhou, Y., Tao, Q., Rosen, J., van Dijken, S.: Bioinspired multisensory neural network with crossmodal integration and recognition. *Nat. Commun.* **12**(1), 1120 (2021)
168. Tang, Y., Tian, Y., Ha, D.: Evojax: Hardware-accelerated neuroevolution. arXiv preprint [arXiv:2202.05008](https://arxiv.org/abs/2202.05008) (2022)
169. Tangri, R., Mandic, D.P., Constantinides, A.G.: Pearl: Parallel evolutionary and reinforcement learning library (2022)
170. Taylor, M.E., Stone, P.: Transfer learning for reinforcement learning domains: A survey. *J. Mach. Learn. Res.* **10**, 1633–1685 (2009)
171. Taylor, M.E., Whiteson, S., Stone, P.: Transfer via inter-task mappings in policy search reinforcement learning. In: Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS '07. Association for Computing Machinery, New York, NY, USA (2007)
172. Teller, A.: The Evolution of Mental Models, pp. 199–217. MIT Press, Cambridge, MA, USA (1994)

173. Thorndike, E.L.: Animal Intelligence; Experimental Studies. New York, The Macmillan Company (1911). <https://www.biodiversitylibrary.org/bibliography/55072>
174. Tolguenec, P.-A.L., Rachelson, E., Besse, Y., Wilson, D.G.: Curiosity Creates Diversity in Policy Search (2022)
175. Tongchim, S., Yao, X.: Parallel evolutionary programming. In: Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753), vol. 2, pp. 1362–1367 (2004)
176. Tupper, A., Neshatian, K.: Evolving neural network agents to play atari games with compact state representations. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion, GECCO '20, pp. 99–100. Association for Computing Machinery, New York, NY, USA (2020)
177. Vasicek, Z., Sekanina, L.: Hardware accelerators for cartesian genetic programming. In: Michael, O., Leonardo, V., Steven, G., Anna Isabel, E.A., Ivanoe, D.F., Antonio, D.C., Ernesto, T. (eds.) Genetic Programming, pp. 230–241. Springer, Berlin (2008)
178. Vassiliades, V., Chatzilygeroudis, K., Mouret, J.-B.: Using centroidal voronoi tessellations to scale up the multidimensional archive of phenotypic elites algorithm. *IEEE Trans. Evol. Comput.* **22**(4), 623–630 (2018)
179. Verbanesics, P., Stanley, K.O.: Evolving static representations for task transfer. *J. Mach. Learn. Res.* **11**, 1737–1769 (2010)
180. Vigorito, C.M., Barto, A.G.: Hierarchical representations of behavior for efficient creative search. In: Creative Intelligent Systems, Papers from the 2008 AAAI Spring Symposium, Technical Report SS-08-03, Stanford, California, USA, March 26–28, 2008, pp. 135–141. AAAI (2008)
181. Varghese, N.V., Mahmoud, Q.H.: A survey of multi-task deep reinforcement learning. *Electronics* **9**(9) (2020)
182. Wang, J., Zhang, Y., Kim, T.-K., Yunjie, G.: Shapley q-value: A local reward approach to solve global reward games. *Proceed. AAAI Conf. Artif. Intell.* **34**, 7285–7292 (2020)
183. Wang, R., Lehman, J., Clune, J., Stanley, K.O.: Poet: Open-ended coevolution of environments and their optimized solutions. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '19, pp. 142–151. Association for Computing Machinery, New York, NY, USA (2019)
184. Watson, R.A., Pollack, J.B.: Modular interdependency in complex dynamical systems. *Artif. Life* **11**(4), 445–457 (2005)
185. Whiteson, S.: Evolutionary Computation for Reinforcement Learning, pp. 325–355. Springer Berlin Heidelberg, Berlin (2012)
186. Whiteson, S., Kohl, N., Miikkulainen, R., Stone, P.: Evolving soccer keepaway players through task decomposition. *Mach. Learn.* **59**(1), 5–30 (2005)
187. Whitley, D., Dominic, S., Das, R., Anderson, C.W.: Genetic reinforcement learning for neurocontrol problems. *Mach. Learn.* **13**(2–3), 259–284 (1993)
188. Wiering, M.A.: Convergence and divergence in standard and averaging reinforcement learning. In: Jean-François, B., Floriana, E., Fosca, G., Dino, P. (eds.) Machine Learning: ECML 2004, pp. 477–488. Springer, Berlin (2004)
189. Wiggins, G.A.: A preliminary framework for description, analysis and comparison of creative systems. *Knowl. Based Syst.* **19**(7), 449–458 (2006) Creative Systems
190. Williams, R.J.: Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.* **8**(3–4), 229–256 (1992)
191. Wilson, D.G., Cussat-Blanc, S., Luga, H., Miller, J.F.: Evolving simple programs for playing atari games. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '18, pp. 229–236. Association for Computing Machinery, New York, NY, USA (2018)
192. Yannakakis, G.N., Togelius, J.: Artificial Intelligence and Games. Springer (2018)
193. Yoo, A.H., Collins, A.G.E.: How working memory and reinforcement learning are intertwined: a cognitive, neural, and computational perspective. *J. Cogn. Neurosci.* **34**(4), 551–568 (2022)

194. Yu, W., Liu, C.K., Turk, G.: Policy transfer with strategy optimization. In: International Conference on Learning Representations (2019)
195. Zhou, S., Seay, M., Taxidis, J., Golshani, P., Buonomano, D.V.: Multiplexing working memory and time in the trajectories of neural networks. In: Nature Human Behaviour (2023)
196. Zhu, S., Belardinelli, F., González León, B.: Evolutionary reinforcement learning for sparse rewards. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO '21, pp. 1508–1512. Association for Computing Machinery, New York, NY, USA (2021)

Part II

Evolutionary Computation as Machine Learning

In which we discuss long existing evolutionary computation methods from the point of view of machine learning: Clustering, classification, regression and modeling, and ensembles.

Chapter 5

Evolutionary Regression and Modelling



Qi Chen, Bing Xue, Will Browne, and Mengjie Zhang

Abstract Regression and modelling, which identify the relationship between the dependent and independent variables, play an important role in knowledge discovery from data. Symbolic regression goes a step further by learning explicitly symbolic models from data that are potentially interpretable. This chapter provides an overview of evolutionary computation techniques for regression and modelling including coefficient learning and symbolic regression. We introduce the ideas behind various evolutionary computation methods for regression and present a review of the efforts on enhancing learning capability, generalisation, interpretability and imputation of missing data in evolutionary computation for regression.

5.1 Introduction

Regression, which is originally from statistical modelling, is an inductive learning task that extracts general law from a set of observed data and studies the relationship between the independent/explanatory/input variables and the dependent/response/output variable(s). The input variables are also called features or attributes in machine learning. In recent years, regression has been the standard

Q. Chen · B. Xue · M. Zhang (✉)

Evolutionary Computation Research Group at the School of Engineering and Computer Science,
Victoria University of Wellington, Wellington 6140, New Zealand

e-mail: Mengjie.Zhang@ecs.vuw.ac.nz

Q. Chen

e-mail: qi.chen@ecs.vuw.ac.nz

B. Xue

e-mail: Bing.Xue@ecs.vuw.ac.nz

W. Browne

Faculty of Engineering, School of Electrical Engineering and Robotics,
Queensland University of Technology, Brisbane, Australia
e-mail: will.browne@qut.edu.au

approach to modelling the relationship between input variables and the outcome variable(s). Unlike classification to predict discrete class labels, regression predicts numeric/continuous values. Regression models, which represent the most well-understood models in numerical simulation, are typically denoted as $Y = f(X) + \alpha$, where $X = \{x_1, x_2, \dots, x_m\}$ represents the m input variables and Y refers to the output variable(s), while α is the error term that captures all kinds of errors. A regression analysis typically has two objectives of explanatory analysis, which aims to weigh and understand the effect of the input variables on the output variable(s) and predictive analysis to predict the outputs with a combination of the input variables. Traditionally, these objectives are achieved by finding the (near-) optimal coefficients with a predefined model structure for regression, e.g. find the optimal β values for a linear regression model $f(X) = \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_m x_m$. Regression analysis is often the starting point of data science and machine learning as through which it becomes possible to understand how the varied input variables lead to the change of the response variable(s).

A range of machine learning approaches can perform regression. These approaches often assume various relationships between the explanatory and dependent variables. The most common regression models include simple or multiple linear regression, polynomial regression, etc. There are also many techniques for parameter/coefficient estimation for these regression models, e.g. least-squared estimation and gradient descent. However, the least squared-based approaches are often not able to solve some complex nonlinear regression models. It is desired to develop effective and efficient approaches to estimate parameters for complex regression tasks. Moreover, in the era of big data, with the increase in the dimensionality and volume of regression data, it becomes more and more difficult to assume the underlining distributions and corresponding model structures for regression. Evolutionary computation techniques, which can free users from these assumptions, are highly demanded.

5.2 Evolutionary Computation for Regression: Fundamentals

Many evolutionary computation (EC) algorithms have been proposed for regression analyses in several ways. According to their tasks, these techniques can be classified into two groups: (1) EC to estimate the coefficients of predefined/assumed regression models, and (2) EC to identify both the model structure and the coefficients for regression.

5.2.1 Evolutionary Computation for Learning Coefficients for Regression

Many of the existing EC methods for regression can be counted in the aforementioned group of coefficient learning. Consider the general nonlinear regression task as follows:

$$y = f(x, \theta), \quad (5.1)$$

where y denotes the target variable, x denotes the input variables, $\theta = [\theta_1, \theta_2, \dots, \theta_m]$ are the parameters to be estimated. Once the model structure $f(\cdot)$ and the error metric have been decided (or selected), regression becomes an optimisation problem with the coefficients $\theta = [\theta_1, \theta_2, \dots, \theta_m]$ as the decision variables.

Different from conventional search techniques, EC methods are population-based search methods. They are typically stochastic and use a cost/fitness function that does not require derivatives. Each solution in the population is typically encoded as a vector of real values. Each element in the vector corresponds to one coefficient in the regression model. A fitness value is obtained for each solution to measure how well the corresponding regression model fits the given data. These fitness values are the basis for the evolutionary process of EC methods, which mimic natural evolution by following the survival of the fittest theory.

Genetic algorithms (GAs) have been used for coefficient learning for regression as early as 1990 [37, 39]. Figure 5.1 shows an example of the evolutionary process of

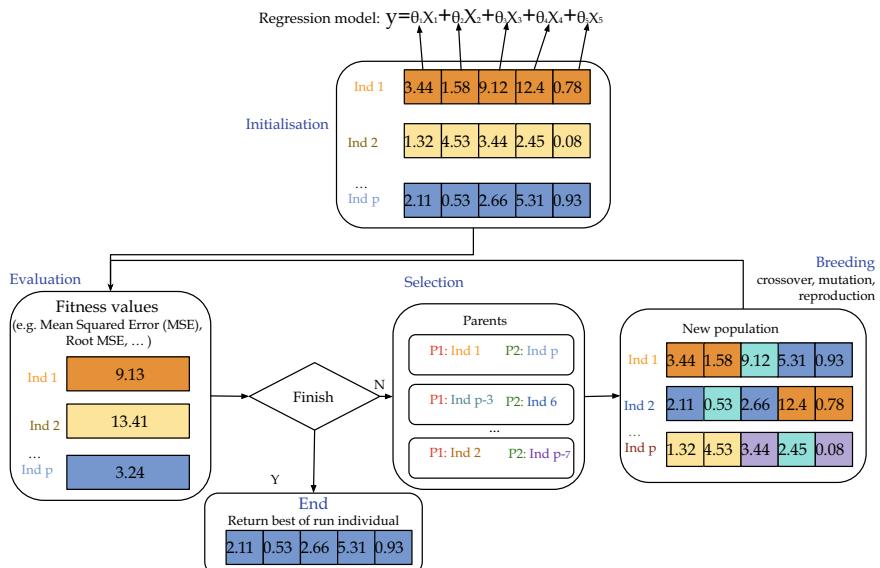


Fig. 5.1 An example of GA for coefficient learning for a regression model $y = \theta_1x_1 + \theta_2x_2 + \theta_3x_3 + \theta_4x_4 + \theta_5x_5$

a GA method for learning coefficients for a regression model. Another EC technique, Particle swarm optimization (PSO), has also been used for coefficient learning for regression more recently [50, 64]. Compared with many other EC techniques such as GAs, PSO has several attractive attributes. PSO has a memory where all particles can retain the knowledge from the good solutions/particles. Moreover, there is cooperation between particles as the swarm share information. Differential evolution (DE) and its enhanced versions [51] have also been utilised for coefficient learning for nonlinear regression. These researches show that many of the drawbacks associated with the conventional coefficient estimation methods can be circumvented when using EC methods instead. For example, there is no need for a good initial guess of the coefficients since it is not a critical requirement for convergence anymore. A broad range for the coefficients is enough for a reasonably good estimation.

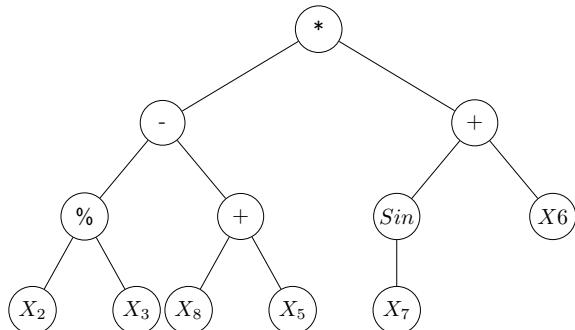
5.2.2 Genetic Programming for Symbolic Regression

Symbolic regression (SR) is a special type of regression without the need for a predefined model structure or the assumption on the underlying data distribution. This is a sharp difference from traditional regression. During the modelling procedure, SR finds the model structure in a symbolic representation and the coefficients of the model simultaneously. This is particularly attractive when there is a lack of domain knowledge and/or when the data is too complicated to derive any single or obvious distribution.

Genetic Programming [43] (GP), an EC approach, is an attractive and dominating technique for SR due to its flexible representation and symbolic nature. GP follows the Darwinian natural selection principle and automatically generates computer programs to solve various problems in many different fields [43]. Given a set of data, GP for SR (GPSR) learns the models iteratively via combining a number of features and essential mathematical expressions. The modelling process in GPSR typically begins with a population of randomly generated solutions/SR models. Utilising breeding mechanisms and fitness-based selection, this population simulates natural evolution by stochastically generating a new population in the next generation. At the end of the evolutionary process, the SR model with the best fitness will be the final solution.

In GP, candidate solutions are typically represented as *parse/expression trees*, an example of which is shown in Fig. 5.2. The nodes in the parse tree come from a function set F and a terminal set T , both of which are user predefined. For GPSR, the function set F usually consists of some basic arithmetical functions including *Addition* (“+”), *Subtraction* (“-”), *Multiplication* (“ \times ”) and *Division* (protected “/”). It can also include more advanced mathematical functions, e.g. transcendental functions and trigonometric functions. Each of these functions has a fixed number of arguments. For example, there need to be two arguments for function “ \times ” while only one argument for “Sin” (Sine) function. The terminal set T usually consists of independent/input variables and some random constants. The definition of the function set and the terminal set needs to meet the properties of *sufficiency* and *closure*.

Fig. 5.2 A GP tree representing the regression model $((x_2/x_3) - (x_8 + x_5)) * (\sin(x_7) + x_6)$



[14]. Sufficiency indicates the terminal set and the function set should be expressive enough to contain potential solutions, while closure requires that every function is able to take any possible output(s) of any other nodes in the tree. A typical example of satisfying closure is extending standard division to the protected “/”. If there is any division by 0 happens, protected “/” will return a predefined value of either 0 or 1 or some other soft-map function.

There are also many other forms of representations for GP including *linear genetic programming* [16], *gene expression programming* [88], *Cartesian GP* with a graphic structure [54] and *Grammar-based genetic programming* [53]. Among these GP variants, GP using tree representation is still most common for SR [15].

There are a few commonly used initialisation methods for GP to create the initial population. *Grow* is probably the simplest initial method. In *Grow*, a node can be selected from either the terminal set or the function set randomly before reaching the maximal tree size/depth. The current branch will stop building when a node from the terminal set is used. Consequently, *Grow* usually generates trees of different sizes and shapes. Another common initialisation method for GP is the *Full* method. This method generates GP trees with a size/depth as the predefined maximal values. Before the maximum size/depth is reached, the tree nodes consist of the operators from the function set only. A combination of the two aforementioned methods is referred to as *ramped-half-and-half*. Using *ramped-half-and-half* for initialisation, the *Full* and *Grow* methods will generate half of the initial population for GP, respectively.

The evolutionary process of GP seeks fitter individuals where a fitness function evaluated on the training set is needed to measure the performance of GP individuals. The design of the fitness measure should provide a fine-grained differentiation between candidate solutions for GP. Generally, various error measures, e.g. *sum of absolute error*, *mean/root mean squared error*, *normalised mean squared error* and *relative squared error* are commonly cast as the fitness function in GPSR.

Evolution in GP proceeds by transforming parents into offspring. After obtaining the performance of GP individuals, GP selects individuals as parents for breeding with a bias to solutions above average performance. *Selection operators* and *genetic operators* are needed during this procedure.

Tournament selection is perhaps the most common selection operator in GP. t individuals are firstly sampled, then the fittest one among these t individuals will be selected as a parent for generating child/offspring programs. The tournament size t implements selection pressure. Typically, a larger tournament size corresponds to a higher selection pressure as more fitted individuals are likely to be selected as parents.

Crossover, mutation and *reproduction* are the three most common genetic operators for GPSR. With two parent individuals, *crossover* generates two new offspring individuals via swapping the genetic materials from the parents. While mutation performs on one parent only to generate a new child. It mutates a subprogram/submodel of the parent with a randomly generated one. Reproduction works by directly copying the parent.

A summary of the key points when tackling a GPSR problem is as follows:

- Choose a *representation* scheme for GPSR and specify the elements in the terminal set and the function set. The scheme will form the regression models without any predefined model structure.
- Choose the *evaluation* mechanism for measuring the performance of GPSR solutions. Typically, various error functions can be used, e.g. mean squared error is a commonly used one.
- Determine the *selection* operator to select parents for breeding.
- Determine the *genetic operators* for breeding. Two basic genetic operators for GPSR are *subtree crossover* and *subtree mutation*.

5.2.3 Learning Classifier Systems for Regression and Function Approximation

Learning classifier systems (LCSs) are an evolutionary machine learning technique where learning is considered an adaption process that interacts with a previously unknown environment and keeps on receiving numerical rewards in solving a given task [66]. The core of an LCS is a set of rules/classifiers which typically take the form of “IF condition THEN action” plus adapted parameters such as rule fitness. The goal of LCSs is to have these classifiers collectively model the decision-making process. The learning process in LCSs is often guided by EC techniques, e.g. genetic algorithms (GAs).

The XCS classifier system [84] is a Michigan-style LCS, where individual rules form the population so must cooperate to form the overall solution in a piecewise manner. XCS is designed to learn the most accurate predictions for a given input and available actions. As an enhanced variant of XCS, XCSF [85] is designed for function approximation that evolves overlapping piecewise-linear functions. Given an input vector $\text{vec}x_i = (x_1, x_2, \dots, x_n)$, XCSF iteratively evolves a population of classifiers, each of which makes a linear prediction. XCSF relies on two major learning components. The evolutionary component is designed to evolve classifiers that

generate linear approximations of the function. The gradient descent-based component generates linear approximation and estimates the accuracy of the classifiers. The interaction between these two components is important for the success of XCSF.

In XCSF, a classifier consists of a condition C and a prediction Pre . The condition C defines a hyper rectangle with an interval of $C = [(l_1, l_2, \dots, l_n)^T, (u_1, u_2, \dots, u_n)^T]$, while the prediction Pre specifies a prediction of the input vector, which is usually linear and determined by the inner product $Pre = \mathbf{x}^{*T} \mathbf{w}$ where $\mathbf{x}^* = (x_0, x_1, \dots, x_n)$ is the input vector $\mathbf{x} = (x_1, x_2, \dots, x_n)$ augmented by a constant x_0 and the weight vector $\mathbf{w} = (w_0, w_1, \dots, w_n)^T$. While the condition parts are evolved by a GA, the other components are iteratively approximated by a gradient-based technique. In each iteration, a match set $[M]$, containing classifiers whose conditions match the input vector, is generated to predict the function value. More specifically, given an input vector \mathbf{x}_i , each classifier cl_m in $[M]$ generates a prediction $Pre_m = \mathbf{x}_i^{*T} \mathbf{w}$. The system prediction of the function value is given by the fitness-weighted average of the predictions of all matching classifiers. Each classifier in $[M]$ is updated according to its error signal using the delta update rule, i.e. $\mathbf{w} \leftarrow \mathbf{w} + \eta(y - \mathbf{x}^{*T} \mathbf{w})\mathbf{x}^*$, where η is the learning rate.

Many different variants of XCSF have been proposed for improving its capabilities, performance and efficiency. Butz et al. [17] develop a new approach to partitioning the problem space. To this aim, they first modify the geometric interpretation toward hyper-spherical shapes, then introduce hyper-ellipsoidal conditions where a distinction in stretching for each dimension is possible. Lanzi et al. [48] extend XCSF by using a polynomial approximation for prediction modelling. They find that the quadratic and cubic approximations can increase the accuracy of XCSF and generate a set of more compact rules. Loiacono et al. [52] introduce support vector regression into XCSF. The enhanced XCSF is shown to have a more accurate prediction for higher dimensional functions and converge much faster than the original XCSF with linear approximation. Stein et al. [69] augment XCSF with the Interpolation Component to take a set of sampling points and carry out an interpolation. The authors later [70] extend this method by utilising a Radial Basis Function to calculate the interpolated function value.

5.3 Genetic Programming for Symbolic Regression: Generalisation

Generalisation is a critical performance metric for any machine learning algorithm that measures the prediction accuracy of learned models on future unseen data. With a good generalisation capability, a learnt model not only has a good learning performance on the training data but also obtains a good prediction performance on the test data. An important goal of machine learning techniques is to learn a model that can minimise the expected error $Err = E[L(Y, f(X))]$ [36], where $L(Y, f(X))$ is the loss function between the target outputs Y and the model predictions $f(X)$. A set

of input X and the corresponding output Y are typically sampled from an underlying distribution $P(X, Y) = P(Y|X)P(X)$, where $P(X)$ is the distribution of the input X and the conditional distribution $P(Y|X)$ is based on the input-output relation. To measure the expected error Err , we need the joint distribution $P(X, Y)$. However, in most learning tasks and real-world applications, the joint distribution is typically unknown. Thus, learning algorithms mainly rely on the empirical risk minimisation principle, which relies on choosing the candidate model with the lowest training error [36]. In many cases, the training error/empirical risk could be a good indicator of the expected test error, but this is not the case in some difficult regression tasks, e.g. when a small number of training instances do not represent the real distribution of the data well, or when the data is noisy, or when the learned model is overly complex.

In the early work of GPSR, the learning was focused on a training set only, typically model/curve fitting [18, 28], which was described in the previous section. In ML tasks, including symbolic regression, achieving good performance on an unseen test set is more important than receiving good results on the training set only. In this section, we will provide more details on how to enhance generalisation performance on unseen data via the representation, local search, model complexity measures, evaluation metrics and the selection operator.

5.3.1 Representation

In GPSR, an expression/parse tree is a common representation for candidate regression models where each node could be a constant, an input feature/variable, or a function/operator. The flexibility of this representation makes it suitable to comprise a large set of possible functions. However, a notable drawback of this representation is that it may include redundant expressions, thus leading to unnecessarily complex models with poor generalisation.

Instead of working with an expression tree, Sotto et al. [29] propose to use linear GP for SR, which represents the regression models using linear programs. With an enhanced selection method to increase the number of effective codes in the programs, linear GP evolves compact SR models with a better generalisation ability. Multiple Regression Genetic Programming (MRGP) [13] proposed a new way to use the output of an expression tree. MRGP decouples the subtrees of GP trees and combines them in linear regression models for the target variable. Specifically, they utilise multiple LASSO regression, which contains the subtrees for each GP tree to approximate the target outputs. Due to its ensemble nature, MRGP is able to generate fitter and more robust regression models than traditional GP. A similar idea is presented by La Cava et al. [46], where they develop the Feature Engineering Automation Tool (FEAT). With FEAT, each GP individual represents a set of candidate features that are used in ridge regression. The Interaction-Transformation Evolutionary Algorithm (ITEA) [31] has a new representation named Interaction-Transformation, which does not produce overcomplex mathematical expressions and thus only allows GPSR to generate simpler regression models that could be well generalised.

GPSR works by searching a model space where models are a combination of functions/operators from a predefined function set. To ensure the model space is broad enough to cover the optimal/nearly-optimal model, the function set of GP usually contains many diverse functions, anticipating that the searching process will discover the optimal model or at least a good model. The function set typically includes the basic arithmetic operators, e.g. addition, subtraction, multiplication and the protected division. For complex SR tasks, GPSR applications could also include more advanced mathematical functions, such as trigonometric functions, exponential functions, protected logarithms and square root. Keijzer [40] demonstrated that the protected division often hinders the generalisation of GPSR as they often generate asymptotes which produce extremely large outputs when testing the learnt models on unseen data, therefore leading to a poor generalisation. Ni et al. [59] propose the Analytic Quotient (AQ) operator as a replacement for the protected division. AQ has the general properties of division, but enables regression models using it to be twice differentiable and therefore can apply regularisation techniques, thus resulting in lower generalisation error. Nicolau et al. [60] recently further investigate the detrimental effects of some other functions/operators on the generalisation of GPSR models, and propose alternative function sets that are more robust.

5.3.2 *Gradient Descent Search for Learning/Optimising Constants in GP Evolved Programs*

GP as an evolutionary algorithm can be seen as a kind of stochastic beam search. The search process of GPSR is guided by fitness-based selection. There are some situations where promising model structures are ignored due to low fitness caused by unfit coefficients. Gradient descent is a well-established search technique and can guarantee to find a local minimum that might not be the best solution but often meets the requirement for the given task. While weights in an artificial neural network are often adjusted by gradient descent, terminal constants in GP trees are typically randomly generated without much learning effort. This is a major “issue” in standard GPSR. Topchy et al. [72] empirically compare vanilla GP with a GP with the Ephemeral Random Constants (ERC) learning method. In their hybrid GP method, gradient descent seeks better model coefficients at the same time as GP searches for the regression model structure. The coefficients in every GP program are trained using a simple gradient algorithm at every generation. The learning effectiveness of the approach is demonstrated on several symbolic regression problems. Later, inspired by [86], Chen et al. [22] investigate the effectiveness of applying gradient descent with various frequencies during the learning process in GPSR and find that only applying the gradient descent to the top 20% GP programs in every k generations achieve much better generalisation than vanilla GP and using the gradient descent in every program at every generation. Figure 5.3 shows an example of gradient descent method for learning coefficients for a GP model with two numerical/constant terminals. It is

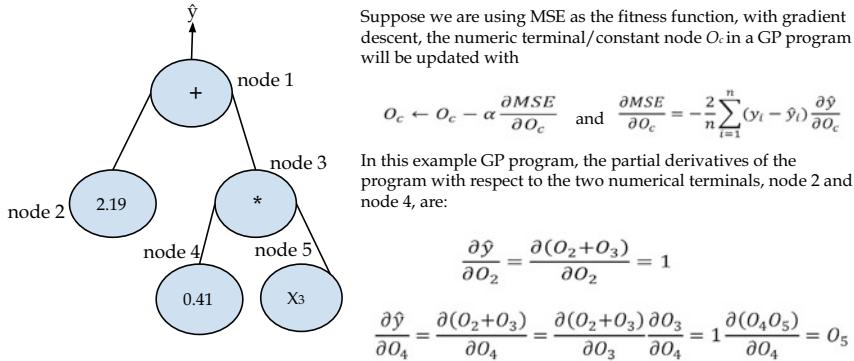


Fig. 5.3 An example of GP with gradient descent for coefficient learning

important to note that these constant terminals here are actually model coefficients, which are being modified by gradient descent search, so are not constants any more.

5.3.3 Complexity Measures and VC Dimension

There is a group of methods that rank and select GP models with a lower estimated generalisation error during the evolutionary process. They estimate the generalisation error using various functions of the training error, model complexity and the number of training instances, where a critical component in these methods is the measure of model complexity.

Generalisation and model complexity are closely related to each other in GPSR [12, 25]. Too low a model complexity could lead to a poor generalisation ability since the model may not fit the data adequately, while a too complex model tends to overfit the training data as the model has more than enough space for the noise and specific patterns in the training data (but not in the test set). A number of efforts have been devoted to control model complexity thus improving their generalisation in GPSR [27, 49]. These approaches can be broadly divided into two groups: controlling the structural/syntax complexity of models in GP and controlling their functional/behavioural complexity.

The structural complexity of GPSR models usually involves the size/length and the shape of the expression tree. Traditionally, structural complexity measures typically work by counting the number of structural elements, e.g. the number of nodes, leaves or layers of GP trees. The functional complexity of a model usually involves estimating the behaviour of models over the input space. There are a large number of structural complexity measures, while only a limited number of functional complexity measures have been investigated/proposed for GPSR to date. Functional complexity measures are typically more effective at enhancing the generalisation of

GPSR as they are more closely linked to overfitting. However, the estimation of the functional complexity of a GPSR model is difficult thus hindering its application as a standard setting in GPSR. Estimating the generalisation ability of models via model complexity frees GPSR from holding out a set of validation data. This is helpful when there is insufficient data available. However, these methods [21, 27, 79, 82] have not gained much attention so far.

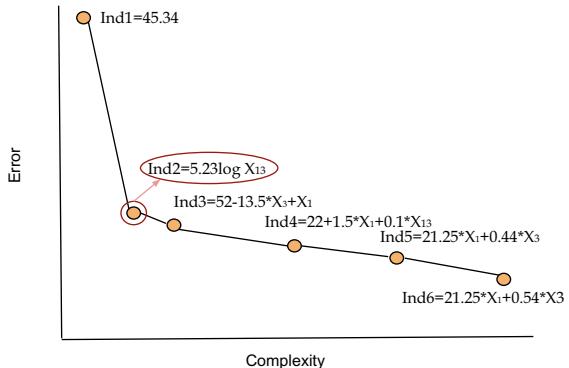
Vladislavleva et al. [82] propose a behavioural complexity measure—order of nonlinearity for GPSR where the complexity of the model is equal to the minimum order of Chebyshev polynomials best approximating the model. A notable weakness of this complexity measure lies in the difficulty to construct the Chebyshev polynomial approximation satisfying a given accuracy and the nontrivial computational procedure in obtaining the true minimal degree of Chebyshev approximation. Van Neschi et al. [79] present a complexity measure inspired by the concept of curvature. They measure the complexity of the current best model to explore the relationship between this measure with bloat and overfitting. Chen et al. [21, 27] introduce the Vapnik-Chervonenkis (VC) dimension [80] to measure the complexity of GPSR models. They introduce a method to measure the VC dimension of GPSR models empirically by minimising the difference between the theoretical and experimental maximal deviations obtained by the model on any two datasets. The obtained VC dimension is later used in the evaluation metric of structural risk minimisation to favour complexity-simpler models. GPSR achieves a good trade-off between learning accuracy and model complexity when introducing the VC dimension in the fitness function of GP. Despite the impressive benefits of generalisation, it is complicated and expensive to calculate the VC dimension of GPSR models, which hinders the proposed complexity measure to be widely used in GPSR in practice. In more recent work [25, 62], another complexity measure based on Rademacher complexity is incorporated into GPSR to reduce the functional complexity. The Rademacher complexity of a GP individual is measured by the ability to fit some random Rademacher variable(s). The Rademacher complexity also has the benefit on enhancing the generalisation of GPSR but is more economic than, and can perhaps be regarded as a simulation of the VC dimension measure.

While it is important to define model complexity measures that are highly related to the generalisation of GPSR models, it is also crucial to ensure the measures are empirically achievable. Therefore, further investigations are desired to find an efficient way to measure model complexity, which can thus be utilised for better generalisation in GPSR.

5.3.4 Multi-objective Approaches

Multi-objective approaches for increasing the generalisation ability of GPSR typically minimise the training error and a second objective reflecting the generalisation error [42, 68]. Commonly, this could be the model size or the model complexity as discussed in the previous section. Figure 5.4 shows an example of a front of GP

Fig. 5.4 An example of a front of models evolved by multi-objective GP. Among the six models, Ind 2, which has the best balance between the two objectives, is typically considered to be the best model



individuals with two objectives of error and complexity. As the model complexity/size is explicitly defined as an objective, the evolved models are either more parsimonious or simpler compared to those generated by the vanilla single-objective GPSR method thus generalising better.

Kommenda et al. [42] propose a multi-objective GPSR method where an efficient complexity measure that includes both the syntactical and the semantic information of the model is treated as another objective. Their method has been shown to evolve simpler and better generalised GPSR models.

Chen et al. [24] develop a new evaluation measure based on the correlation between variables and residuals of GPSR models to prevent learning from noisy data. They introduce a powerful statistical association measure, Maximal Information Coefficient, to identify the correlation between variables and residuals. Their multi-objective GPSR method, which minimises both the training error and the correlation between residuals and variables, has a good generalisation capacity and evolves compact regression models.

5.3.5 Lexicase Selection

Lexicase selection [38] is a parent selection method that considers the error of GP models on one training instance in a random order at a time instead of the common fitness value aggregated over all the training instances. Figure 5.5 shows three examples of lexicase selection. The original lexicase selection performs well in discrete error spaces but not in continuous spaces. A new form of lexicase selection, ϵ -lexicase selection, has been developed for GPSR [47]. The new lexicase selection redefines the pass condition for individuals on each examination and makes it less strict in GPSR with continuous error spaces. ϵ -lexicase selection with automatic threshold adaptation has been shown to promote population diversity and make use of more fitness cases when selecting parents thus enhancing the performance of GPSR.

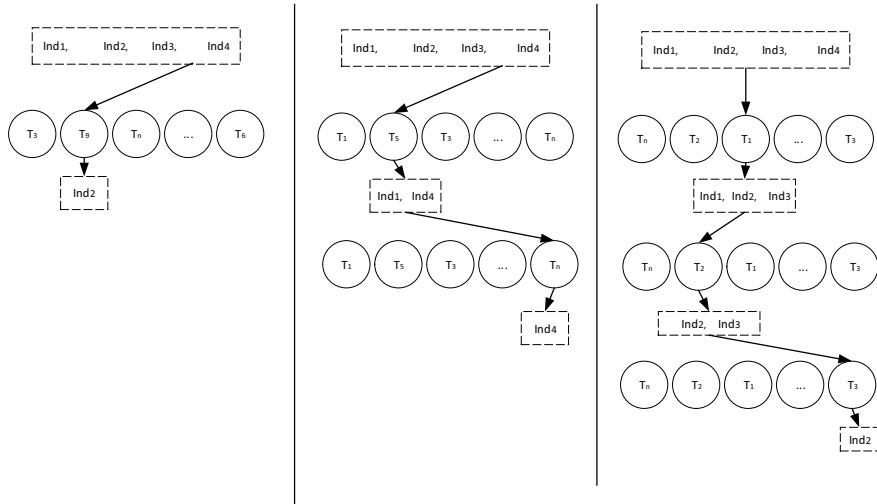


Fig. 5.5 Examples of lexicase selection

5.4 Semantic GP for Regression

The semantics of computer programs provides grounding for their syntax [45], more specifically, the semantics of programs determine their behaviours. But most GP algorithms manipulate programs only on the structure level without considering their semantics. Considering program/model structure alone allows GP to rely on simple search operators. However, this comes at a cost that the genotype-phenotype mapping in GP is complicated. It is usually hard to predict how the modification of the structure of a program will impact its output. A small change in the program structure can have a profound impact on its semantics and thus its fitness. This causes the fitness landscape of GP to be complex and limits its scalability [45].

In recent years, much research was devoted to investigating the importance of semantics in GP and forming a “new” research track of Semantic GP (SGP). Although it is still young, the obtained results of the research track so far are promising and encouraging.

5.4.1 Semantics in GP

Developing semantic GP methods first requires a formal definition of program semantics. Krawiec et al. [45] give a formal definition for the semantics of a GP program based on the given training instances which specify the desired behaviour of GP programs. They define the semantics of a GP program as a vector, each element of

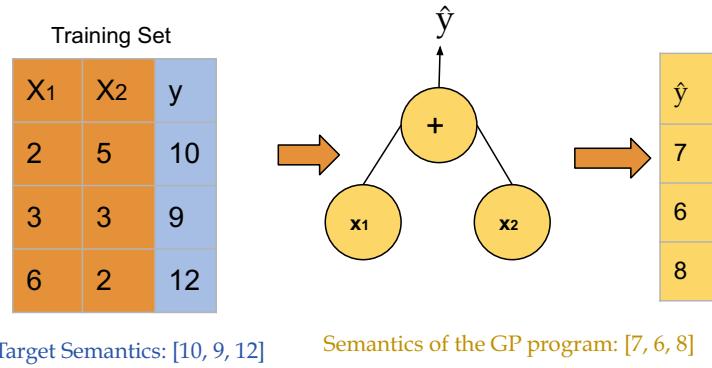


Fig. 5.6 An example of semantics in GPSR

which is the corresponding output of the program produced on each input/training instance. An example of semantics of GP based on this definition is shown in Fig. 5.6.

The semantics can then be viewed as a point in an N -dimensional space, which is referred to as *semantic space*, where N is the number of fitness cases, i.e. number of training instances. Since fitness cases in GPSR could not enumerate all the possible inputs, researchers also call these semantics-*sampling semantics* [76].

Many semantic GPSR methods have been proposed since 2009 [61, 76]. Basically, these semantic GP methods still act on the structure of programs either at the individual or sub-individual level, but rely on specific genetic operators or semantic survival criteria to achieve the desired semantic requirements indirectly.

Nguyen et al. [76] explore the effect of semantic guidance for crossover in GPSR. They proposed a new semantic aware crossover (SAC) that requires exchanging subtrees having equivalence on their approximate semantics. To determine the semantic equivalence of two expressions, they propose sampling semantics, which consists of the evaluation results of the expressions on a random set of points sampled from the domain. Two subtrees are designated as semantically equivalent if their output on the random sample set is close enough according to the semantic sensitivity. Nguyen et al. [76, 77] later extend and improve SAC by introducing semantic similarity-based crossover (SSC), which encourages exchanges of subtrees with not wildly different semantics. SSC forces a change in the semantics of GPSR programs in the population but keeps the change bounded. In this way, SSC helps to improve locality. However, it can also lead to the loss of diversity in some cases. Thus, a multi-population SSC approach [61] was developed to maintain population diversity. The experimental results showed that the multi-population method further improves the results obtained by SSC. [75] investigates the impact of incorporating semantic awareness into GP on its generalisation ability.

5.4.2 Geometric Semantic GP

Krawiec et al. [44] propose the approximately geometric crossover (AGC), where the semantic distance between the candidate child and the parents in the semantic space is used to guide the crossover operator. AGC generates a number of child programs. Among them, only the offspring whose semantics are most similar to those of their parents will survive. Another notable finding is that geometric crossover breeds offspring that are guaranteed to be not worse than the worst parent, while geometric mutation induces a unimodal fitness landscape. However, there is no clear or easy way to explore these properties due to the complex genotype-phenotype mapping in GP.

Moraglio et al. [56] propose the geometric semantic GP method (GSGP) introducing the precise genetic operators that search directly in the semantic space. In GSGP for SR, geometric semantic crossover on two parents P_1, P_2 generates the offspring $P_{XO} = P_1 \cdot P_R + (1 - P_R) \cdot P_2$, where P_R is a random real function with the output in the range of $[0, 1]$. Geometric semantic mutation of an individual P creates the individual $P_M = P + ms \cdot (P_{R1} - P_{R2})$, where P_{R1} and P_{R2} are two random real functions and ms —the mutation step is a constant to decide the perturbation caused by mutation. Figure 5.7 shows an example of geometric semantic operators under the Euclidean metric. The precise geometric semantic operators are promising as they are able to induce a unimodal fitness landscape. However, they have an obvious limitation of leading to rapid growth in the size of individuals. The precise geometric semantic operators generate offspring consisting of the complete structure of the parent(s), the random tree(s) and some additional operators. After a few generations, fitness evaluation becomes extremely expensive, thus the population is unmanageable. Moraglio et al. [56] suggest adding an automatic simplification step in each generation. Vanneschi et al. [78] propose an improved method to implement

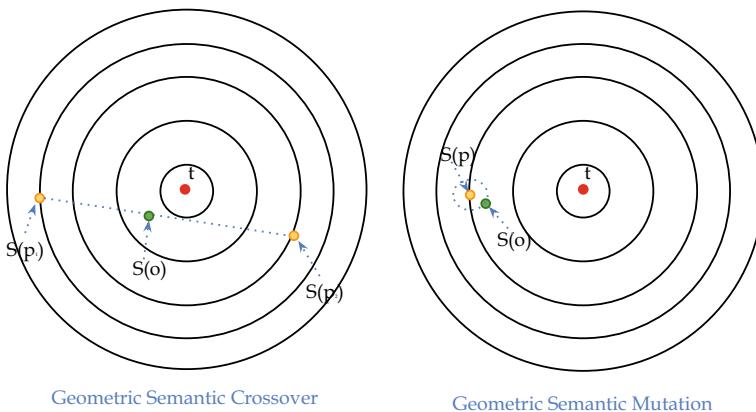


Fig. 5.7 An example of geometric semantic operators for generating a child program $s(o)$ under the Euclidean metric

the geometric semantic operators where they do not generate the programs explicitly during the evolutionary process. Instead, in their implementation, only the materials that are needed to reconstruct the programs are stored in memory. These include the individuals in the initial population, the involved random trees and a set of memory pointers to previous generations of individuals.

5.4.3 Angle-Aware Geometric Semantic Operators

GSGP presents a solid theoretical framework for designing precise geometric search operators. Utilising the geometry of the semantic space could be helpful for enhancing search in GP. However, it is difficult to realise efficient precise geometric search operators that search directly in the semantic space. Moreover, there are some potential limitations on the effectiveness of geometric semantic operators. Geometric semantic crossover can generate children outperforming both parents *only* when the target semantics are between the semantics of parents. In addition, geometric semantic crossover becomes less effective with a large number of data instances that lead to a high-dimensional semantic space. For geometric semantic mutation, the search performance is sensitive to the mutation step. It is always difficult to bound the variation of this mutation operator. Small mutation steps decrease the exploration ability of GP, thus leading to underfitting, while too big a mutation step could lead to overfitting, in turn leading GSGP to generalise poorly on unseen data.

Chen et al. [19] develop an angle-aware mating scheme for geometric semantic crossover. Given a set of candidate parents selected by tournament selection, with the angle-aware mating scheme, the geometric crossover will perform on two parents with the largest angle distance between their relative semantics to the target. The large angle distance benefits the effectiveness of crossover by reducing semantic duplication among the child programs and also their parents. Moreover, the mating scheme also helps foster the exploration ability of GP and leads to faster convergence to the target semantics. However, along with the evolutionary process in GPSR, the candidate parents become closer to each other in the semantic space, thus it is extremely difficult to find parents with a large angle distance at the final stage of the evolutionary process.

To further explore the geometric properties of the semantic space and obtain a greater generalisation gain for GPSR, Chen et al. [23] later propose angle-driven geometric semantic GP (ADGSGP). To further utilise the angle-awareness mating, an angle-driven selection (ADS) for geometric crossover is proposed. The new selection operator selects pairs of parents that have a large angle distance in the semantic space from the GP population directly without utilising the tournament selection. In addition, two new geometric semantic operators are also developed, perpendicular crossover (PC) and random segment mutation (RSM). PC further explores the geometric properties of the parent semantics anticipating to eliminate the ineffectiveness of the original geometric semantic crossover. While RSM aims to make it easy to determine the mutation step, it works by generating a child program highly correlated

to its parent and approximating the target semantics considering the right direction toward it. To fulfil the semantics of the new geometric operators, they develop a new algorithm named semantic context replacement (SCR). SCR addresses the potential drawbacks of a previous method, semantic backpropagation (SB), which is sensible but generally produces overly complex programs. They find that GSGP methods, including ADGSGP, usually generate complex models but generalise well. This conflicts with the common observation that complex programs usually overfit the training data. A possible reason for the generalisation gain of these complex models could be that they exhibit the same strength as ensembles. Ensemble learning tackles the generalisation problem by combining a number of accurate models to reduce the possibility of overfitting. The negative contribution of the overfitted models will be eliminated by models that generalise well in the final solution. For an effective ensemble learning, a necessary condition is that the ensemble has a mix of accurate and diverse models [89]. GSGP can be seen as an ensemble learning method, since geometric genetic operators always combine parent individuals and some randomly generated individuals to produce new individuals. These parents can be seen as the accurate models while the randomly generated individuals can be treated as providing the necessary diversity. The competitive generalisation of GSGP may derive a combination of these models, which can be treated as the ensembles.

5.5 GP for Regression with Incomplete Data

Data incompleteness is a pervasive problem in machine learning, of course including GPSR. There are three main categories of missing values according to how the missingness forms. They are missing completely at random (MCAR), missing not at random (MNAR) and missing at random (MAR) [35]. In MCAR, a random subset of the given data is missing for a completely random reason. The probability of a missing value is not related to any other values. For MCAR, most techniques for handling missingness give unbiased results since no gain can be guaranteed by analysing the available data for estimating missing values. For MNAR, the probability that a missing value is related to some non-observed information. There is no universal method to handle this type of missingness due to the unavailable of valuable information. Usually, missing values are neither MCAR nor MNAR. MAR refers to the probability of missingness depending on other existing values. Methods for handling missing values can be categorised into three different groups. They are:

- Delete incomplete instances and learn from the complete data portion only. This is a simple approach but may suffer from losing informative data;
- Impute incomplete data and learn using a complete data set after imputation and
- Directly deal with the original incomplete data without imputation.

In the rest of this section, we will mainly present the latter two groups.

5.5.1 Imputation Methods

Data imputation is a popular approach to handling data incompleteness. It estimates missing values based on existing data. The imputed data can then be used by any learning algorithm. There are two main types of imputation: single imputation and multiple imputation [35]. Single imputation directly replaces missing values with estimated ones based on observed data. The common problem with single imputation is that it ignores uncertainty and underestimates the variance. Multiple imputation addresses this problem by considering both within-imputation uncertainty and between-imputation uncertainty.

Data imputation approaches can also be categorised as data-driven approaches and regression-based approaches [35]. Data-driven imputation is typically based on instance similarity. A popular data-driven imputation method is K nearest neighbour (KNN)-based imputation. Another type of imputation method is based on feature predictability where regression algorithms are utilised to predict incomplete features. GPSR-based imputation is a typical example of this kind. Instance similarity-based imputation methods utilise the relationships between instances, while feature predictability-based imputation methods employ the relationships between features. However, both the instance similarity-based approach and the feature predictability-based approach undermine the other's factor. There is a desire for a new approach to combining the benefits from both of them.

Vladislavleva et al. [81] treat missing values in a certain range of the feature space as an unbalanced problem. They address the problem with an automatic instance weighting method according to the proximity, remoteness, surrounding and nonlinear deviation of the instances from their nearest neighbours. Al-Helali et al. [2] develop a hybrid imputation method called GP-KNN for GPSR with missing values. GP-KNN utilises the KNN-based instances proximity while employing the GP-based feature regression predictability. GPSR builds regression-based imputation models, which learn from similar instances selected by KNN. GP-KNN outperforms many popular imputation methods, e.g. KNN, random forest (RF), and classification and regression trees (CART). However, GP-KNN has a notable limitation of the need to build imputation models for imputing new incomplete instances, which can be computationally expensive. To address this limitation, Al-Helali et al. [10] extend GP-KNN by combining a weighted KNN (WKNN) with GP. The weighted KNN is used to increase the contributions of instances that are close to the incomplete instances when building the imputation models. GP imputation models learn from the instances retrieved by WKNN. To improve the efficiency, the regression models constructed in the training process are used on new unseen data with missing values, to avoid the expensive computational cost of building new imputation models.

5.5.2 Interval Functions

Another approach to dealing with incomplete data is to build learning algorithms that can directly work with missing values. Some attempts have been made for utilising *interval arithmetic* to enable GP to learn directly from incomplete data.

Interval arithmetic refers to dealing with intervals that are sets of real numbers defined by $[x^-, x^+] = \{X \in \mathbb{R} | x^- \leq x \leq x^+\}$. Four basic interval arithmetic operators are

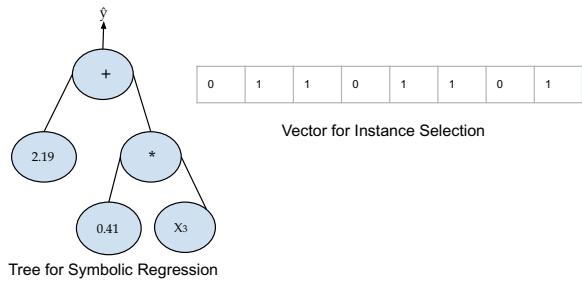
- $[x_1^-, x_1^+] + [x_2^-, x_2^+] = [x_1^- + x_2^-, x_1^+ + x_2^+]$;
- $[x_1^-, x_1^+] - [x_2^-, x_2^+] = [x_1^- - x_2^+, x_1^+ - x_2^-]$;
- $[x_1^-, x_1^+] * [x_2^-, x_2^+] = [\min(x_1^- x_2^-, x_1^- x_2^+, x_1^+ x_2^-, x_1^+ x_2^+), \max(x_1^- x_2^-, x_1^- x_2^+, x_1^+ x_2^-, x_1^+ x_2^+)]$;
- $[x_1^-, x_1^+]/[x_2^-, x_2^+] = [x_1^-, x_1^+] \cdot (1/[x_2^-, x_2^+])$.

The original idea of incorporating interval arithmetic into GP is from Keijzer [40], who presents the idea of replacing protected operators by using the known ranges of input features to compute the expected output interval of a GPSR model. Since the introduction of using interval arithmetic in GP, it has been used primarily for the evaluation of GPSR models. Later, Dick [32, 33] explores the potential of interval arithmetic for an effective GP search. The execution interval of the models based on the known interval of the input variables is obtained and checked. Only models with a valid execution interval could be evaluated normally. Otherwise, they will be assigned the lowest fitness. Tran et al. [73, 74] use GP with interval functions to construct features [74] and classifiers [73] for classification tasks with incomplete data. Al-Helali et al. [1] develop a feature selection method based on the interval function method for GPSR on incomplete regression data. In their method, missing values are replaced by an interval of the feature estimated based on its minimum and maximum of existing values, e.g. the original instances with missing values (?, 2.2, 1.3) and (?, ?, 3.3) where the interval of the three features corresponding to $[-1, 2]$, $[-2, 3]$ and $[-3, 5]$ would be converted to the interval-valued instances $([-1, 2], [2.2, 2.2], [1.3, 1.3])$ and $([-1, 2], [-2, 3], [3.3, 3.3])$. With interval arithmetic operators, the proposed GP for feature selection method works on intervals instead of numerical values and is able to select informative features and predictors for imputation. The selected incomplete features are then imputed and fed into the GPSR learning process.

5.5.3 Instance Selection and Feature Selection

Instance selection is a process to find a set of useful data instances to improve learning performance. It is typically very useful for discarding noisy and outlier instances, thus improving the effectiveness and efficiency of learning algorithms. However,

Fig. 5.8 An example of GP with hybrid tree-vector representation to perform symbolic regression and instance selection simultaneously on incomplete data (Adapted from [8])



instance selection methods usually use algorithms that work only on complete data such that very few of them are applicable to incomplete data.

Al-Helali et al. [4, 8] integrate instance selection into data imputation for GPSR on incomplete data. To this aim, they investigate using KNN for instance selection for imputation [4], and later propose a tree-vector mixed representation for GP to perform instance selection and symbolic regression simultaneously on incomplete data. Figure 5.8 shows an example of the tree-vector mixed representation for GP. Using this representation, each individual has two components, i.e. an expression tree and a bit vector. The tree component constructs GPSR models, while the bit vector represents selected instances that are used in the imputation method. The proposed method enhances the imputation performance of the WKNN method in both effectiveness and efficiency.

High dimensionality data is a challenge for regression, which is even more difficult on incomplete data. To address this problem in GPSR, Al-Helali et al. [3, 5] explore incorporating feature selection pressure into the fitness function of GP in the form of the ratio of selected features [3] and the model complexity [5]. They develop a regularised GP method [5], where the complexity measure based on the Hessian matrix of the phenotype of GP models is utilised as a regulariser for feature/predictor selection for imputation.

5.5.4 Transfer Learning

Transfer learning, which reuses knowledge learned from the source domain(s) to improve the learning in the target domain(s), has been considered a key solution for the problem of learning when lacking knowledge in the target domain(s). Incompleteness is one of the main causes of knowledge shortage in many machine learning tasks. However, transfer learning has seldom been used to address this issue. Al-Helali et al. [6] propose a multi-tree GP-based feature transformation method for transferring data from a complete source domain to a different, incomplete target domain. This method is enhanced by reducing the source-target domain distribution dissimilarity in [7]. Later, the same authors further extend these methods [9] to

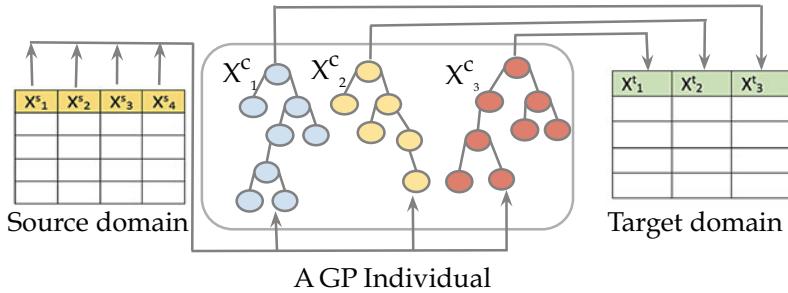


Fig. 5.9 An example of GP for feature construction to minimise the distribution mismatching between the source domain and the target domain with incomplete data

utilise the impact of the source features and instances on SR in the source domain to improve the transformation, as shown in Fig. 5.9.

5.6 Interpretable GP/Evolving Interpretable Models for Regression

Interpretable regression models, which describe everything explicitly in a mathematical form, could provide a good insight into the decision-making process. Due to its symbolic nature, GP has inherent interpretability. Typically, with a tree-based representation, a GP program is explicitly represented as an expression tree, which is often consistent with how humans understand a structure of a computer program in a tree representation. Many other GP variants, either with a linear representation or with a graph representation, are also interpretable, e.g. linear GP represents a program as a sequence of instructions, which is essentially equivalent to human-written programs (e.g. in C++, Java and Python). An interpretable regression model should be accurate at predicting the target variable, while at the same time being as simple as possible to facilitate the interpretation of the decision-making process and the discovery of new knowledge from the data.

5.6.1 Representation

França et al. [30] present a new representation named Interaction-Transformation (IT) for GPSR, which constrains the search space. A regression model with IT representation approximates the target model in a form of $f(x) = \sum_i w_i \cdot g_i(x)$ where w_i s are the coefficients, and the composition function $g(\cdot) = t(\cdot) \cdot p(\cdot)$, $t(\cdot)$ is a one-dimensional transformation function, and $p(\cdot)$ is a d-dimensional interaction function. This representation does not comprehend the description of a more com-

plicated set of mathematical expressions thus preventing GPSR from finding overly complex regression models that are difficult to interpret. Raymond et al. [63] propose a new representation for GP named adaptive weighted splines, which uses a fixed model structure. With adaptive weighted splines, each GP model is composed of a number of feature splines. Each feature in the input space is represented by a smoothing spline with two coefficients determining its influence on the final prediction of the model. This semi-structured GP method no longer fully presents the symbolic capability of GP, but the models developed can be regulated far more easily and also more interpretable.

5.6.2 Offline and Online Program Simplification

GPSR models are prone to contain many redundant branches. Although redundant branches can help protect the truly useful branches from being destroyed during evolution [71], they inevitably decrease the interpretability of GPSR models. Thus, it is desirable to remove the redundant branches to simplify GPSR models and make them more interpretable without affecting their performance. Following the principle of Occam's Razor, simplification typically generates smaller-sized variants of complex GP solutions. It can work in an offline mode on the final GP solution only or in an online mode that may alter the GP search process and lead to different sets of final solutions.

Naoki et al. [57] propose an offline simplification method, Equivalent Decision Simplification, for GPSR. The simplification method works by replacing subtrees with some predefined simpler subtrees if they are equivalent. They determine equivalence between specific simple trees and a subtree based on whether they yield numerically equivalent values over a range of inputs. Zhang et al. [87] develop an online simplification method based on algebraic simplification rules, algebraic equivalence and prime number techniques to simplify GP programs. When applying to an appropriate proportion of individuals at a certain frequency, the method has been shown to significantly reduce the size while enhancing the performance of GPSR. Kinzett et al. [41] develop a numerical simplification algorithm, which calculates the numerical contribution of a node to its parents. The subtree rooted at the node will be deleted if its contribution is below a user-defined threshold. In addition, a node will be replaced by a constant if its variance is less than a threshold. Rockett [65] develops a statistical permutation test-based pruning method for GPSR. They use a constant to replace the redundant subtrees, the removal of which does not statistically significantly change the output of the parent tree. They have observed around a 20% deduction in the median tree size of GPSR models.

5.6.3 Feature Selection

In explainable AI, a good explanation is often *selected*, i.e. instead of a complete set of causes of an event, humans are more interested in a subset of the more relevant causes [55]. This explanation can be achieved by finding the most informative features via *feature selection*. GP has a built-in feature selection ability that has been widely used in various tasks.

Feature selection can be achieved by feature ranking methods that generates a ranking of all features in terms of how important they are in modelling the output variable, then choosing the top features. Most existing feature ranking methods ignore interactions between features, leading to redundancy in the obtained feature subset. GP implicitly considers interactions between features when performing feature ranking. The importance of features can be simply measured based on the frequency of a feature appearing in good GP individuals [83]. Vladislavleva et al. [83] define two feature importance measures, which are the presence-weighted importance and the fitness-weighted importance. The presence-weighted method analyses the feature presence rate in a subset of selected models from the archive of Pareto GP, the fitness-weighted method obtains the feature importance by summing up the relative fitness of features in all the models. Chen et al. [20] develop a multi-stage GPSR process where at the first stage distinct features are collected from the top solutions at each generation and at the later stage only top solutions and the candidate features collected from the first stage are accepted for further features selection and modelling for SR. Smits et al. [67] measure the importance of features that appear in the best GP solutions in a multi-objective GP method. The presented features get their importance as the equally distributed fitness of the best solutions.

However, the average distribution of fitness values has the limitation that features presented in the inactive chunks of code can still obtain some credit as being a part of the best solutions. To avoid this limitation, the permutation-based feature importance measures have been proposed for GPSR [26, 34]. Permuting the values of features refers to rearranging the values of features randomly within the dataset, e.g. given the values of a feature as {7,6,9}, the permutation of the feature can take a random form among {6,9,7}, {6,7,9}, {7,6,9}, {7,9,6}, {9,7,6}, {9,6,7}. The underlying assumption of permutation importance is that important features should contribute more on model fitness, thus, permuting a more important feature will lead to a worse regression performance. Aldeia et al. [11] employ the partial effect/partial derivative, which measures the importance of the feature by how a discrete change of its value affects the target when its co-variables are with fixed values, for measuring the feature importance in GPSR.

5.7 LCSs for Symbolic Modelling

Symbolic representations have the notable benefit of the expressive power to represent complex relationships between the inputs and the prediction. Naqvi et al. [58] further extend XCSRCA to develop XCS-SR which is the very first LCS method for symbolic regression. Instead of using a binary tree representing the code fragment for actions, the classifier in XCS-SR uses a GP-style tree to represent a symbolic function. XCS-SR has been shown to outperform a semantic GP method on a number of benchmark SR tasks. They own the advantages of XCS-SR to the flexibility of multiple solutions each of which is the collective nature of the multiple rules in LCS and the limited tree depth which generates the bloat-free classifiers.

5.8 Summary

This chapter presented an introduction and an overview of a number of EC methods for regression and modelling including coefficient learning in traditional regression and symbolic regression (SR). We have discussed a broad range of research works in this field from the classic methods to the very recent research efforts on EC for regression/symbolic regression. We have highlighted the advancements of these population-based methods over traditional least-squared-based methods for regression. Meanwhile, we also identified the main challenges and some potential solutions in these research areas. It is worth highlighting, among many cutting-edge data-driven modelling methods, great attraction could be with EC (mostly GP) for SR, particularly with limited or even no available domain knowledge. EC for SR provides analytic/symbolic models/equations, thereby supporting interpretable decision-making and knowledge discovery, and delivering scientific insights.

However, there are also a number of open issues in EC for SR that deserve future research and further investigation including but not limited to the efficiency and the scalability of evolutionary search, generalisation ability of evolved SR model and constant/coefficient optimisation for evolved SR models. Several representations have been proposed for evolutionary algorithms to effectively search entire solution space for regression. However, future research should investigate the new representations that could lead to effective search in EC for SR. In addition, research on modularity in EC may be a promising direction to improve the generalisation and scalability of EC for SR when tackling complex regression problems. Moreover, gradient-based coefficient optimisation approaches could be efficient and discover accurate values for coefficients of a regression model learned during evolutionary search.

References

1. Al-Helali, B.: Genetic programming for symbolic regression on incomplete data. PhD thesis, Open Access Te Herenga Waka-Victoria University of Wellington (2021)
2. Al-Helali, B., Chen, Q., Xue, B., Zhang, M.: A hybrid GP-KNN imputation for symbolic regression with missing values. In: Australasian Joint Conference on Artificial Intelligence, pp. 345–357. Springer (2018)
3. Al-Helali, B., Chen, Q., Xue, B., Zhang, M.: Genetic programming-based simultaneous feature selection and imputation for symbolic regression with incomplete data. In: Asian Conference on Pattern Recognition, pp. 566–579. Springer (2019)
4. Al-Helali, B., Chen, Q., Xue, B., Zhang, M.: GP-based feature selection and weighted KNN-based instance selection for symbolic regression with incomplete data. In: 2020 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 905–912 (2020)
5. Al-Helali, B., Chen, Q., Xue, B., Zhang, M.: Hessian complexity measure for genetic programming-based imputation predictor selection in symbolic regression with incomplete data. In: European Conference on Genetic Programming (EuroGP), pp. 1–17. Springer (2020)
6. Al-Helali, B., Chen, Q., Xue, B., Zhang, M.: Multi-tree genetic programming-based transformation for transfer learning in symbolic regression with highly incomplete data. In: 2020 IEEE Congress on Evolutionary Computation (CEC), pp. 1–8. IEEE (2020)
7. Al-Helali, B., Chen, Q., Xue, B., Zhang, M.: Multi-tree genetic programming for feature construction-based domain adaptation in symbolic regression with incomplete data. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference, pp. 913–921 (2020)
8. Al-Helali, B., Chen, Q., Xue, B., Zhang, M.: GP with a hybrid tree-vector representation for instance selection and symbolic regression on incomplete data. In: 2021 IEEE Congress on Evolutionary Computation (CEC), pp. 604–611. IEEE (2021)
9. Al-Helali, B., Chen, Q., Xue, B., Zhang, M.: Multitree genetic programming with new operators for transfer learning in symbolic regression with incomplete data. *IEEE Trans. Evol. Comput.* **25**(6), 1049–1063 (2021)
10. Al-Helali, B., Chen, Q., Xue, B., Zhang, M.: A new imputation method based on genetic programming and weighted KNN for symbolic regression with incomplete data. *Soft. Comput.* **25**(8), 5993–6012 (2021)
11. Aldeia, G.S.I., de França, F.O.: Measuring feature importance of symbolic regression models using partial effects. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 750–758 (2021)
12. Amari, S.-I., Wu, S.: Improving support vector machine classifiers by modifying kernel functions. *Neural Netw.* **12**(6), 783–789 (1999)
13. Arnaldo, I., Krawiec, K., O'Reilly, U.-M.: Multiple regression genetic programming. In: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation (GECCO), pp. 879–886 (2014)
14. Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.D.: *Genetic Programming—An Introduction: On the Automatic Evolution of Computer Programs and Its Applications*. dpunkt-Verlag and Morgan Kaufmann Publishers Inc, San Francisco, California (1998)
15. Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.D.: *Genetic Programming: An Introduction: On the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann Publishers Inc., San Francisco (1998)
16. Brameier, M., Banzhaf, W.: A comparison of linear genetic programming and neural networks in medical data mining. *IEEE Trans. Evol. Comput.* **5**(1), 17–26 (2001)
17. Butz, M.V.: Kernel-based, ellipsoidal conditions in the real-valued xcs classifier system. In: Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation, pp. 1835–1842 (2005)
18. Chen, Q., Xue, B.: Generalisation in genetic programming for symbolic regression: challenges and future directions. In: *Women in Computational Intelligence: Key Advances and Perspectives on Emerging Topics*, pp. 281–302. Springer (2022)

19. Chen, Q., Xue, B., Mei, Y., Zhang, M.: Geometric semantic crossover with an angle-aware mating scheme in genetic programming for symbolic regression. In: European Conference on Genetic Programming (EuroGP), pp. 229–245. Springer (2017)
20. Chen, Q., Xue, B., Niu, B., Zhang, M.: Improving generalisation of genetic programming for high-dimensional symbolic regression with feature selection. In: 2016 IEEE Congress on Evolutionary Computation (CEC), pp. 3793–3800 (2016)
21. Chen, Q., Xue, B., Shang, L., Zhang, M.: Improving generalisation of genetic programming for symbolic regression with structural risk minimisation. In: Proceedings of the 18th Annual Conference on Genetic and Evolutionary Computation (GECCO), pp. 709–716. ACM (2016)
22. Q. Chen, B. Xue, and M. Zhang. Generalisation and domain adaptation in GP with gradient descent for symbolic regression. In: 2015 IEEE Congress on Evolutionary Computation (CEC), pp. 1137–1144, May 2015
23. Chen, Q., Xue, B., Zhang, M.: Improving generalization of genetic programming for symbolic regression with angle-driven geometric semantic operators. *IEEE Trans. Evol. Comput.* **23**(3), 488–502 (2018)
24. Chen, Q., Xue, B., Zhang, M.: Improving symbolic regression based on correlation between residuals and variables. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference, pp. 922–930 (2020)
25. Chen, Q., Xue, B., Zhang, M.: Rademacher complexity for enhancing the generalization of genetic programming for symbolic regression. *IEEE Trans. Cybern.* (2020). <https://doi.org/10.1109/TCYB.2020.3004361>
26. Chen, Q., Zhang, M., Xue, B.: Feature selection to improve generalization of genetic programming for high-dimensional symbolic regression. *IEEE Trans. Evol. Comput.* **21**(5), 792–806 (2017)
27. Chen, Q., Zhang, M., Xue, B.: Structural risk minimization-driven genetic programming for enhancing generalization in symbolic regression. *IEEE Trans. Evol. Comput.* **23**(4), 703–717 (2019)
28. Costelloe, D., Ryan, C.: On improving generalisation in genetic programming. In: Genetic Programming: 12th European Conference, EuroGP 2009 Tübingen, Germany, April 15–17, 2009 Proceedings 12, pp. 61–72. Springer (2009)
29. dal Piccol Sotto, L.F., de Melo, V.V.: Studying bloat control and maintenance of effective code in linear genetic programming for symbolic regression. *Neurocomputing* **180**, 79–93 (2016)
30. de França, F.O.: A greedy search tree heuristic for symbolic regression. *Inf. Sci.* **442**, 18–32 (2018)
31. de França, F.O., Aldeia, G.S.I.: Interaction-transformation evolutionary algorithm for symbolic regression. *Evol. Comput.* **29**(3), 367–390 (2021)
32. Dick, G.: Interval arithmetic and interval-aware operators for genetic programming (2017). arXiv preprint [arXiv:1704.04998](https://arxiv.org/abs/1704.04998)
33. Dick, G.: Revisiting interval arithmetic for regression problems in genetic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, pp. 129–130 (2017)
34. Dick, G.: Sensitivity-like analysis for feature selection in genetic programming. In: Proceedings of the 19th Annual Conference on Genetic and Evolutionary Computation (GECCO), pp. 401–408 (2017)
35. Donders, A.R.T., Van Der Heijden, G.J., Stijnen, T., Moons, K.G.: A gentle introduction to imputation of missing values. *J. Clin. Epidemiol.* **59**(10), 1087–1091 (2006)
36. Friedman, J., Hastie, T., Tibshirani, R.: The Elements of Statistical Learning, vol. 1. Springer Series in Statistics, New York (2001)
37. Gulsen, M., Smith, A., Tate, D.: A genetic algorithm approach to curve fitting. *Int. J. Prod. Res.* **33**(7), 1911–1923 (1995)
38. Helmuth, T., Spector, L., Matheson, J.: Solving uncompromising problems with lexicase selection. *IEEE Trans. Evol. Comput.* **19**(5), 630–643 (2014)
39. Jervase, J.A., Bourdoucen, H., Al-Lawati, A.: Solar cell parameter extraction using genetic algorithms. *Meas. Sci. Technol.* **12**(11), 1922 (2001)

40. Keijzer, M.: Improving symbolic regression with interval arithmetic and linear scaling. In: Proceedings of the European Conference on Genetic Programming (EuroGP), pp. 70–82. Springer (2003)
41. Kinzett, D., Zhang, M., Johnston, M.: Using numerical simplification to control bloat in genetic programming. In: Asia-Pacific Conference on Simulated Evolution and Learning, pp. 493–502. Springer (2008)
42. Kommenda, M., Kronberger, G., Affenzeller, M., Winkler, S.M., Burlacu, B.: Evolving simple symbolic regression models by multi-objective genetic programming. In: Genetic Programming Theory and Practice XIII, pp. 1–19. Springer (2016)
43. Koza, J.R.: Genetic Programming II. Automatic Discovery of Reusable Subprograms. MIT Press, Cambridge (1992)
44. Krawiec, K., Lichocki, P.: Approximating geometric crossover in semantic space. In: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation (GECCO), pp. 987–994 (2009)
45. Krawiec, K., Pawlak, T.: Locally geometric semantic crossover: a study on the roles of semantics and homology in recombination operators. *Genet. Program Evolvable Mach.* **14**(1), 31–63 (2013)
46. La Cava, W., Moore, J.H.: Learning feature spaces for regression with genetic programming. *Genet. Program Evolvable Mach.* **21**(3), 433–467 (2020)
47. La Cava, W., Spector, L., Dana, K.: Epsilon-lexicase selection for regression. In: Proceedings of the Genetic and Evolutionary Computation Conference, vol. 2016, 741–748 (2016)
48. Lanzi, P.L., Loiacono, D., Wilson, S.W., Goldberg, D.E.: Extending xcsf beyond linear approximation. In: Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation, pp. 1827–1834 (2005)
49. Le, N., Xuan, H.N., Brabazon, A., Thi, T.P.: Complexity measures in genetic programming learning: a brief review. In: Proceedings of the 2016 IEEE Congress on Evolutionary Computation (CEC), pp. 2409–2416. IEEE (2016)
50. Li, L.-L., Wang, L., Liu, L.-H.: An effective hybrid psosa strategy for optimization and its application to parameter estimation. *Appl. Math. Comput.* **179**(1), 135–146 (2006)
51. Li, S., Gu, Q., Gong, W., Ning, B.: An enhanced adaptive differential evolution algorithm for parameter extraction of photovoltaic models. *Energy Convers. Manage.* **205**, 112443 (2020)
52. Loiacono, D., Marelli, A., Lanzi, P.L.: Support vector regression for classifier prediction. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, pp. 1806–1813 (2007)
53. McKay, R.I., Hoai, N.X., Whigham, P.A., Shan, Y., O'Neill, M.: Grammar-based genetic programming: a survey. *Genet. Program. Evolv. Mach.* **11**(3), 365–396 (2010)
54. Miller, J.F.: Cartesian genetic programming: its status and future. *Genet. Program. Evolv. Mach.* **21**(1), 129–168 (2020)
55. Miller, T.: Explanation in artificial intelligence: insights from the social sciences. *Artif. Intell.* **267**, 1–38 (2019)
56. Moraglio, A., Krawiec, K., Johnson, C.G.: Geometric semantic genetic programming. In: International Conference on Parallel Problem Solving from Nature, pp. 21–31. Springer (2012)
57. Mori, N., McKay, B., Hoai, N.X., Essam, D., Takeuchi, S.: A new method for simplifying algebraic expressions in genetic programming called equivalent decision simplification. In: International Work-Conference on Artificial Neural Networks, pp. 171–178. Springer (2009)
58. Naqvi, S.S., Browne, W.N.: Adapting learning classifier systems to symbolic regression. In: 2016 IEEE Congress on Evolutionary Computation (CEC), pp. 2209–2216. IEEE (2016)
59. Ni, J., Drieberg, R.H., Rockett, P.I.: The use of an analytic quotient operator in genetic programming. *IEEE Trans. Evol. Comput.* **17**(1), 146–152 (2012)
60. Nicolau, M., Agapitos, A.: Choosing function sets with better generalisation performance for symbolic regression models. *Genet. Program. Evolvable Mach.* **22**(1), 73–100 (2021)
61. Pham, T.A., Nguyen, Q.U., Nguyen, X.H., O'Neill, M.: Examining the diversity property of semantic similarity based crossover. In: European Conference on Genetic Programming (EuroGP), pp. 265–276. Springer (2013)

62. Raymond, C., Chen, Q., Xue, B., Zhang, M.: Genetic programming with rademacher complexity for symbolic regression. In: 2019 IEEE Congress on Evolutionary Computation (CEC), pp. 2657–2664. IEEE (2019)
63. Raymond, C., Chen, Q., Xue, B., Zhang, M.: Adaptive weighted splines: A new representation to genetic programming for symbolic regression. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference, pp. 1003–1011 (2020)
64. Robinson, D., Chen, Q., Xue, B., Wagner, I., Price, M., Hume, P., Chen, K., Hodgkiss, J., Zhang, M.: Particle swarm optimisation for analysing time-dependent photoluminescence data. In: 2021 IEEE Congress on Evolutionary Computation (CEC), pp. 1735–1742 (2021)
65. Rockett, P.: Pruning of genetic programming trees using permutation tests. *Evol. Intel.* **13**(4), 649–661 (2020)
66. Sigaud, O., Wilson, S.W.: Learning classifier systems: a survey. *Soft. Comput.* **11**(11), 1065–1078 (2007)
67. Smits, G., Kordon, A., Vladislavleva, K., Jordaan, E., Kotanchek, M.: Variable selection in industrial datasets using pareto genetic programming. In: Genetic Programming Theory and Practice III, pp. 79–92. Springer (2006)
68. Smits, G.F., Kotanchek, M.: Pareto-front exploitation in symbolic regression. In: Genetic Programming Theory and Practice II, pp. 283–299. Springer (2005)
69. Stein, A., Eymüller, C., Rauh, D., Tomforde, S., Hähner, J.: Interpolation-based classifier generation in xcsf. In: 2016 IEEE Congress on Evolutionary Computation (CEC), pp. 3990–3998. IEEE (2016)
70. Stein, A., Menssen, S., Hähner, J.: What about interpolation? a radial basis function approach to classifier prediction modeling in xcsf. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 537–544 (2018)
71. Streeter, M.J.: The root causes of code growth in genetic programming. In: European Conference on Genetic Programming (EuroGP), pp. 443–454 (2003)
72. Topchy, A., Punch, W.F., et al.: Faster genetic programming based on local gradient search of numeric leaf values. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), vol. 155162. Morgan Kaufmann, San Francisco (2001)
73. Tran, C.T., Zhang, M., Andreae, P.: Directly evolving classifiers for missing data using genetic programming. In: 2016 IEEE Congress on Evolutionary Computation (CEC), pp. 5278–5285. IEEE (2016)
74. Tran, C.T., Zhang, M., Andreae, P., Xue, B.: Directly constructing multiple features for classification with missing data using genetic programming with interval functions. In: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion, pp. 69–70 (2016)
75. Uy, N.Q., Hien, N.T., Hoai, N.X., O'Neill, M.: Improving the generalisation ability of genetic programming with semantic similarity based crossover. In: European Conference on Genetic Programming (EuroGP), pp. 184–195. Springer (2010)
76. Uy, N.Q., Hoai, N.X., O'Neill, M., McKay, R.I., Galván-López, E.: Semantically-based crossover in genetic programming: application to real-valued symbolic regression. *Genet. Program Evolvable Mach.* **12**(2), 91–119 (2011)
77. Uy, N.Q., O'Neill, M., Hoai, N.X., McKay, B., Galván-López, E.: Semantic similarity based crossover in gp: the case for real-valued function regression. In: International Conference on Artificial Evolution (Evolution Artificielle), pp. 170–181. Springer (2009)
78. Vanneschi, L., Castelli, M., Manzoni, L., Silva, S.: A new implementation of geometric semantic GP and its application to problems in pharmacokinetics. In: European Conference on Genetic Programming (EuroGP), pp. 205–216. Springer (2013)
79. Vanneschi, L., Castelli, M., Silva, S.: Measuring bloat, overfitting and functional complexity in genetic programming. In: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation (GECCO), pp. 877–884 (2010)
80. Vapnik, V.: Estimation of Dependencies Based on Empirical Data. Springer Science & Business Media (2006)
81. Vladislavleva, E., Smits, G., Den Hertog, D.: On the importance of data balancing for symbolic regression. *IEEE Trans. Evol. Comput.* **14**(2), 252–277 (2009)

82. Vladislavleva, E.J., Smits, G.F., Den Hertog, D.: Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. *IEEE Trans. Evol. Comput.* **13**(2), 333–349 (2008)
83. Vladislavleva, K., Veeramachaneni, K., Burland, M., Parcon, J., O'Reilly, U.M.: Knowledge mining with genetic programming methods for variable selection in flavor design. In: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation (GECCO), pp. 941–948 (2010)
84. Wilson, S.W.: Classifier fitness based on accuracy. *Evol. Comput.* **3**(2), 149–175 (1995)
85. Wilson, S.W.: Function approximation with a classifier system. In: Proceedings of the 2001 Annual Conference on Genetic and Evolutionary Computation (GECCO), pp. 974–981 (2001)
86. Zhang, M., Smart, W.: Genetic programming with gradient descent search for multiclass object classification. In: European Conference on Genetic Programming (EuroGP), pp. 399–408. Springer (2004)
87. Zhang, M., Wong, P., Qian, D.: Online program simplification in genetic programming. In: Asia-Pacific Conference on Simulated Evolution and Learning, pp. 592–600. Springer (2006)
88. Zhong, J., Feng, L., Ong, Y.-S.: Gene expression programming: A survey. *IEEE Comput. Intell. Mag.* **12**(3), 54–72 (2017)
89. Zhou, Z.-H.: Ensemble learning. In: Machine Learning, pp. 181–210. Springer Singapore, Singapore (202)

Chapter 6

Evolutionary Clustering and Community Detection



Julia Handl, Mario Garza-Fabre, and Adán José-García

Abstract This chapter provides a formal definition of the problem of cluster analysis, and the related problem of community detection in graphs. Building on the mathematical definition of these problems, we motivate the use of evolutionary computation in this setting. We then review previous work on this topic, highlighting key approaches regarding the choice of representation and objective functions, as well as regarding the final process of model selection. Finally, we discuss successful applications of evolutionary clustering and the steps we consider necessary to encourage the uptake of these techniques in mainstream machine learning.

6.1 Introduction

Unsupervised learning is concerned with the identification of patterns in data in scenarios where information on the outcome of interest is not available directly. In other words, while supervised learning is concerned with the mapping from an input space X to an output (target) space Y , unsupervised learning is strictly limited to the analysis of X and the discovery of patterns inherent to that space.

The most commonly encountered question in unsupervised learning relates to the presence of natural groups within the data, i.e., subsets of entities that are inherently similar or related to each other and, at the same time, inherently dissimilar or unrelated to other entities within a data set. Where data is available in the form

J. Handl (✉)

University of Manchester, Manchester, UK

e-mail: julia.handl@manchester.ac.uk

M. Garza-Fabre

Cinvestav, Campus Tamaulipas, Km. 5.5 carretera Cd. Victoria-Soto La Marina, Cd. Victoria 87130, Tamaulipas, Mexico

e-mail: mario.garza@cinvestav.mx

A. José-García

Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France

e-mail: adan.josegarcia@univ-lille.fr

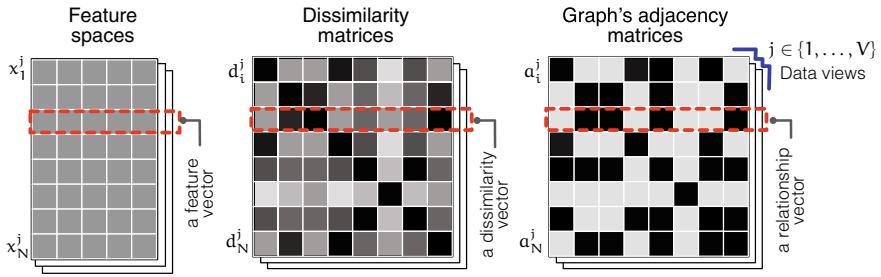


Fig. 6.1 Unsupervised learning scenarios relevant to the problems of cluster analysis and community detection. The different ways in which entities are characterized are highlighted in red: \mathbf{x}_i^j is a feature vector, \mathbf{d}_i^j is a dissimilarity vector, and \mathbf{a}_i^j is a row of the adjacency matrix, such that $j \in \{1, \dots, V\}$, where V represents the number of distinct data views available in the form of feature spaces, dissimilarity matrices, or relational spaces

of features, directly characterizing each entity, or in the form of (dis)similarities, capturing the pairwise relationships between all samples, the above problem is referred to as *clustering* or *cluster analysis* [33]. Otherwise, when the data is characterized primarily through a partial set of relations between the entities, information which is commonly represented as a graph, the resulting problem is known as *community detection* [12]. Figure 6.1 provides a side-by-side comparison of these three key scenarios, which may overlap in practice. Due to their close relationship, this chapter aims to provide a holistic overview of evolutionary approaches designed for all three problem settings.

6.2 Unsupervised Learning Scenarios

We are concerned with unsupervised learning on a set of entities X . Here, X is made up of individual entities $x_i \in X$, with $i \in \{1, \dots, N\}$ and N being the cardinality of X (i.e., N is the total number of entities in the data set).

Depending on the particular scenario of unsupervised learning, further information about entities may be available as follows:

1. In many instances of unsupervised learning, each entity x_i is directly represented through a feature vector:

$$\mathbf{x}_i^j = (x_{i1}^j, \dots, x_{iD^j}^j) .$$

Here, D^j represents the dimensionality of the j -th feature space, F^j . Some applications are characterized by the availability of multiple feature spaces (data views), so $j \in \{1, \dots, V\}$, where V represents the number of distinct feature spaces available.

2. In certain applications of unsupervised learning, a feature representation of individual entities is not appropriate or available. Instead, entities may be characterized indirectly, through their dissimilarity or similarity relationships to all other entities within the data set, X . Concretely, in such scenarios, each entity x_i is represented through a dissimilarity vector:

$$\mathbf{d}_i^j = (d_{i1}^j, \dots, d_{iN}^j) .$$

In vector \mathbf{d}_i^j , d_{it}^j captures the dissimilarity between entity i and entity t , whereas N corresponds to the total number of entities within the data set. As above, some applications are characterized by the availability of multiple dissimilarity values, so $j \in \{1, \dots, V\}$, where V represents the number of distinct dissimilarity matrices (data views) available. Without loss of generality, this definition focuses on dissimilarities only, as relational information presented as a similarity matrix can be mapped to a dissimilarity matrix through a suitable mathematical transformation.

3. Finally, unsupervised learning scenarios can be characterized by the availability of relational information between subsets of entities only. This information can be naturally captured in the form of a graph. Concretely, in such scenarios, each entity x_i is modeled as a node of a graph, and the available relational information, for that node, is given by a single row of the graph's adjacency matrix:

$$\mathbf{a}_i^j = (a_{i1}^j, \dots, a_{iN}^j) .$$

In this case, a_{it}^j captures the strength of the relationship between the i -th and t -th data entities, with a value of $a_{it}^j = 0$ indicating the absence of relational information (and, therefore, absence of an edge between the associated nodes). As before, N refers to the total number of entities within the data set. In the simplest case, the entries of \mathbf{a}_i^j are binary, i.e., $a_{it}^j = 0$ or $a_{it}^j = 1$, representing the absence or presence of a relation, respectively. Where present, the relation is represented as an unweighted edge in the graph. Alternatively, edges within the graph may be weighted and/or directed, reflecting the strength, and potential directionality, of the relationship between connected entities. Again, some applications may be characterized by the availability of multiple sets of relations, so $j \in \{1, \dots, V\}$, where V represents the number of distinct relational spaces (data views) available.

Considering the above definitions, the touching points between the three scenarios are clear. First, given the choice of a suitable distance function, problem instances consistent with Scenario 1 can always be transformed into instances consistent with Scenario 2, by mapping each feature space to a dissimilarity matrix. Second, Scenario 3 presents a generalization of Scenario 2. Specifically, it relaxes two assumptions usually made in cluster analysis: (i) the assumption that relational information is available (or can be derived) for all pairs of data entities; and (ii) the assumption of symmetry made by standard, metric distance functions. In other words, unsupervised learning problems arising in the form of Scenarios 1 and 2 can equivalently be

modeled as learning problems on complete (i.e., fully connected), weighted, and undirected graphs, with edge weights capturing the similarity between entities.

It is also evident that we may encounter instances that combine aspects of different problem scenarios. In providing the above definitions, we have ensured to cater for *multi-view* settings, where data entities can be characterized from a number of incommensurable perspectives. In a multi-view setting, however, each individual data view may arise in a form consistent with a different scenario, so the full problem instance may in fact involve multiple scenarios. For example, a learning problem may involve two views, one available as a feature space and one available in the form of a dissimilarity matrix. Similarly, information provided in the form of a graph may be accompanied by node features, i.e., by a feature vector associated with each of the nodes (data entities) described by the graph. In this case, the learning problem can be thought of as a multi-view problem involving two views: a set of relational information captured by the graph's edges, and a feature space directly characterizing each entity.

6.3 Cluster Analysis and Community Detection

The previous section has defined different scenarios we may encounter in unsupervised learning, with the aim of highlighting the commonalities and potential interplay between those settings. We will now set out to provide a formal definition of the problems of cluster analysis and community detection. While these learning tasks have typically been studied in separate threads in the academic literature, below we will focus on a joint definition. The motivation behind this is the close relationship between them, as highlighted in the scenario definitions above.

The problems of cluster analysis and community detection aim to partition a given set of entities X into sub-groups. A generic definition of these partitioning problems is as follows:

$$\operatorname{argmin}_{K, \Pi} (f^h(\Pi(X))) .$$

Here, K is the decision variable indicating the number of groups (clusters or communities) in a partition. Π defines a partition of X into subsets $\{X_1, \dots, X_K\}$, such that $\forall x_i \in X : \exists X_k : x_i \in X_k$. Commonly, definitions of these problems further assume Π to induce a *crisp partition*, i.e., $\forall l \neq m : X_l \cap X_m = \emptyset$.

In the above definition, f^h represents an objective function that captures the quality of a partition and, without loss of generality, is to be minimized. Note that $h \in \{1, \dots, H\}$ and, with $H > 1$, i.e., where multiple objective functions are to be used, this formulation results in a *multi-objective optimization problem*. The use of multiple objective functions can arise for a variety of reasons. It can help capture multiple different aspects of the quality of a partition, such as relationships within groups and across groups, or it can capture the quality of a partition with regard to multiple views that may be available for the entities considered.

In terms of the problem definition, key differences between cluster analysis and community detection relate only to the specific choices of objective functions, and the information these functions are based on. Fundamentally, this is where clustering algorithms rely on a direct feature representation of each entity, or the full matrix of pairwise dissimilarities between entities. In contrast, objective functions for community detection are expressed in terms of a graph's edges, and the presence or level of relationship (similarity) they convey.

In practice, specialized algorithms for both problems exist, which are not always transferable across problem boundaries. In particular, some of the best-known clustering algorithms assume the presence of a feature space, and do therefore not directly generalize to community detection settings.

6.4 Practical Challenges and Opportunities for Evolutionary Computation

There is a long history of research on cluster analysis and community detection, and this is evident from the large variety of algorithms and objective functions available for both problems [2, 51, 56]. There is a clear trade-off between the flexibility of a given algorithm and its scalability to large problems. Some of the most established algorithms are those that introduce strong assumptions about the data and/or rely on a greedy or local search heuristic. A prominent example of this is the k -means algorithm [39]: it relies on the external definition of the number of clusters, and lends itself to the (local) optimization of a specific objective function (within-cluster variance) only. In return, it achieves a run time complexity that is linear in terms of the number of entities in a data set, and remains one of the most commonly used algorithms in practice [31, 33]. Given steadily increasing demands regarding the scale of data and the speed with which it requires processing, the identification of fast, specialized heuristics (and associated objective functions) remains an important active area of research.

On the other hand, there are applications in which the narrow assumptions made by such heuristics are constraining. Their use may prevent problem formulations that are sufficiently comprehensive to cover all relevant problem aspects, including the full range of data views available. Meta-heuristics such as evolutionary algorithms can provide a powerful alternative in this setting. One of the specific advantages of meta-heuristic algorithms is their flexibility to adapt to different problem formulations, i.e., the opportunity they provide to exchange or experiment with additional objective functions, constraints, and data views, without a complete overhaul of the underlying optimization engine. The ability to perform a wider, global exploration of the search space and escape more easily from local optima is another fundamental advantage of meta-heuristic methods [53, 59].

In the following, we highlight some of the key developments made by evolutionary computation researchers in designing approaches to clustering and community detection. Our discussion focuses on the crucial design challenges associated with the

adoption of existing meta-heuristic optimizers for the unsupervised learning domain, including issues around problem representation and operator design, the interaction of these choices with problem formulation (in terms of objective functions), and the strategies for addressing final model selection. It is our experience that the largest performance differences in unsupervised learning arise from decisions with respect to the problem formulation, rather than the specifics (or parameterization) of the optimizer used. Hence, our discussion will focus on the choice of meta-heuristic optimizer only where this choice is essential to support aspects of the problem formulation.

6.4.1 Solution Representation

Representation plays a crucial role in the adaption of meta-heuristic optimizers to a given problem [53, 59]. Jointly with the variation operators, they define the phenotypic neighborhoods that are accessible from a given candidate solution during optimization, thereby impacting on problem difficulty and the effectiveness of the overall search process. In cluster analysis, this aspect is further amplified by the fact that the choice of representation may predetermine the type of partition that can be induced, so it directly affects the formulation of the optimization problem and the set of phenotypes that can be reached [34, 45].

6.4.1.1 Direct Representations

The simplest representation of a partition relies on the use of a separate decision variable to indicate the cluster or community assignment for every data entity. A direct problem representation can thus be designed as a string $\mathbf{r} = (r_1, \dots, r_N)$, where $r_i \in \{1, \dots, K\}$ represents the cluster or community entity $x_i \in X$ is assigned to, as shown in Fig. 6.2a. Alternatively, a binary representation of these integer values could be adopted, resulting in a binary string of length $N \times \lceil \log_2(K) \rceil$.

The advantage of these direct representations is their generality. Other than a maximum number of clusters K_{\max} , they introduce no assumptions regarding the properties of partitions, and they can therefore be deployed in optimizing partitions with respect to any objective function or constraint. Nevertheless, both representations share the same core issue, which is rooted in the lack of a direct interpretation of the specific cluster or community labels: the labels only serve to induce the co-assignment (and separation) of entities, but the final phenotype (partition) is agnostic to the specific label used. Consider, for example, genotype $(1, 2, 2, 1, 2, 2, 1, 1, 2, 2)$ illustrated in Fig. 6.2a (for a problem with $N = 10$ and $K = 2$). If we exchange cluster labels, so that entities originally assigned to Cluster 1 are now assigned to Cluster 2 and vice versa, we will obtain a completely different new genotype, namely $(2, 1, 1, 2, 1, 1, 2, 2, 1, 1)$. Note, however, that both of these genotypes induce the same two sets of data entities. When it comes to evolutionary algorithms, in particu-

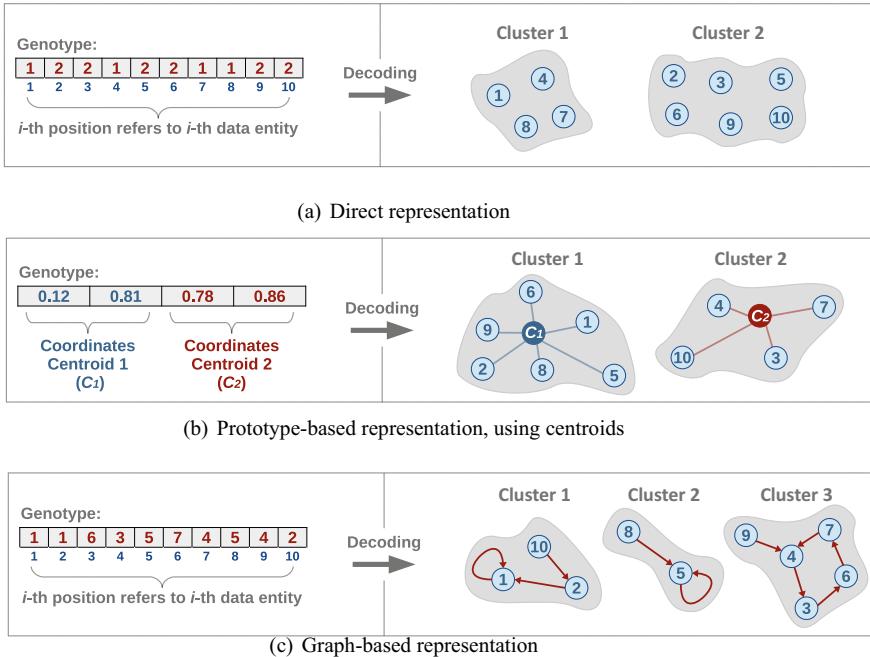


Fig. 6.2 Illustration of the (a) direct, (b) prototype-based, and (c) graph-based representations used in evolutionary clustering (equivalent choices are available for community detection). An example problem instance with $N = 10$ data entities is considered in all cases

lar, the resulting redundancy makes it difficult to derive effective crossover operators for this representation. Adjustments designed to address this issue for grouping problems [11] have shown limited success, especially for the problem scales typically considered during cluster analysis [24]. Consequently, the use of direct representations in the clustering and community detection literature is sparse.

6.4.1.2 Prototype-Based Representations

In meta-heuristic approaches to cluster analysis, prototype-based representations have been extensively used to overcome issues with scalability. Rather than specifying cluster assignments directly, prototype-based approaches specify a set of representatives from which a partition can be induced: each entity is assigned to the cluster associated with the representative it is closest to—in the following we refer to this as the *cluster assignment step*. These representatives may adopt the form of cluster centroids or cluster medoids, depending on the clustering scenario at hand and, specifically, on the availability of a numerical feature space, which is a prerequisite for the calculation of cluster centroids. Hence, the representation can take one of the following forms:

1. A string of $K \times D$ real values, $\mathbf{r} = (r_{11}, \dots, r_{1D}, \dots, r_{K1}, \dots, r_{KD})$, where (r_{k1}, \dots, r_{kD}) represents the centroid vector of the k -th cluster. Figure 6.2b illustrates this representation based on centroids, for $K = 2$ and $D = 2$.
2. A string of K integer values, $\mathbf{r} = (r_1, \dots, r_K)$, where $r_k \in \{1, \dots, N\}$ identifies the specific entity $x_{r_k} \in X$ serving as the cluster medoid for cluster X_k . Alternatively, a binary representation of each medoid could be used.

Either representation can induce a partition on X by assigning each entity to its closest cluster representative, using a designated distance function (in the case of cluster centroids) or dissimilarity matrix (in the case of cluster medoids). Standard prototype-based approaches assume that the number of groups K in the partitions is known in advance, as this defines the representation length. This assumption can be relaxed to assume knowledge of the maximum number of groups K_{\max} only, e.g., through the introduction of placeholder values [4]. The definition of a centroid or medoid does not extend directly to a graph-based scenario, and therefore community detection. However, alternative definitions of node representativeness, within graphs, have been derived and used in this context [64], allowing the adoption of a representation equivalent to the medoid-based approach described above.

A clear advantage of the prototype-based approach is its scalability: it eliminates the linear increase of representation length with the number of entities, N . Instead, representation length becomes a function of the number of desired groups, K , and dimensionality, D (for centroid-based representations). For medoid-based representations, the number of available allele values for each encoding position increases with N , so there remains a dependency of the search space on the number of entities. This becomes explicit in a binary representation: where this is used to represent medoids, the representation length is $K \times \lceil \log_2(N) \rceil$. A separate advantage of prototype-based representations is the fact that, for those assuming a fixed K , routine (generic) variation operators can potentially be used, in the case of real, integer, and binary encodings.

Despite these distinct strengths, key limitations of the prototype-based representations result from the cluster assignment step. The first of these limitations relates to the cluster shapes that can be obtained. In particular, when the standard Euclidean distance is used to support cluster assignment, this restricts the search to clusters with a hyper-spherical shape (i.e., clusters that are compact). This restriction at the representational level will override decisions made during the choice of the objective functions. Consequently, the use of a prototype-based representation can significantly limit an algorithm's ability to discover data partitions that violate assumptions of compactness but provide promising characteristics from other points of view, e.g., clusters that are elongated but spatially well-separated. Recent work has highlighted potential avenues for reducing this limitation [36]. As illustrated in Fig. 6.3, transformations of the original dissimilarity space (choices of more complex distance functions) can potentially address this issue, enabling the use of prototype-based representations for the identification of non-compact clusters.

A second difficulty with prototype-based representations arises in clustering settings involving multiple data views. As the cluster assignment step requires the calcu-

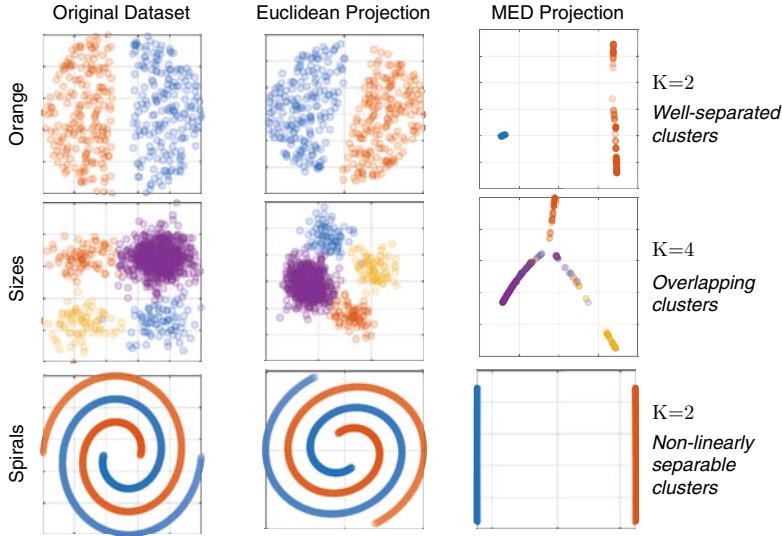


Fig. 6.3 The impact of the distance function on how an algorithm “perceives” the relationships between a given set of entities, which will directly determine cluster assignments in a prototype-based representation: original data (left); embedding of the associated Euclidean distances [6] (center); and embedding of the associated MED distances [6] (right). This shows that the MED distance favors the clear spatial separation of elongated cluster structures (e.g., on the *Spirals* data) but is less capable than the Euclidean distance in separating overlapping clusters (e.g., on the *Sizes* data). This underlines that the choice of distance function can play a significant role in determining the types of clusters that can be identified, irrespective of the choice of objective function

lation or use of dissimilarity information, it becomes a potential source of bias when multiple incommensurable sources of dissimilarity information exist. In particular, reliance on a single source, or on a fixed weighting between the available sources, will override decisions made during the choice of the objective functions. Recent work has highlighted how the introduction of such bias can be avoided in the context of a many-objective optimizer [36]. Specifically, the use of a decomposition-based approach [32, 63] has been exploited as a mechanism to directly align reference vectors deployed within the optimizer with the weights used during cluster assignment (see Fig. 6.4), avoiding the introduction of unintended bias during the search process.

6.4.1.3 Graph-Based Representations

Finally, graph-based representations have enjoyed notable success in identifying partitions, for both clustering and the problem of community detection [30, 51]. In community detection, the rationale for a graph-based representation is straightforward, as it aligns very closely with the learning problem at hand. In cluster analysis, this particular choice of representation is less intuitive. However, in the context

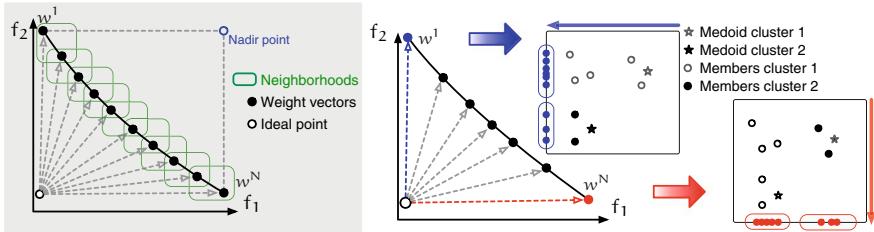


Fig. 6.4 Example of a many-objective optimizer considering many views at the cluster assignment step: This is possible due to explicit knowledge of the weights associated with each reference vector, which are exploited during the decoding (and cluster assignment) as well as the evaluation step of the algorithm

of multi-objective clustering in particular, graph-based representations have been shown to provide an effective mechanism to explore a diverse range of candidate partitions [24].

The most straightforward version of the graph-based representation captures the set of all edges within a graph (or all pairwise relations within a data set), and introduces a binary decision variable to allow for the inclusion or removal of each of them. In other words, the representation of a partition is given as a string of E binary values, $\mathbf{r} = (r_1, \dots, r_E)$, where E denotes the number of edges in the original graph and $r_e \in \{0, 1\}$ indicates the absence or presence of a given link in the candidate graph. To interpret each candidate graph as a partition, an additional decoding step determines the set of connected components defined by the links; each such component is interpreted as a separate cluster or community. This representation can be suitable for small problem instances, and introduces no additional bias, but lacks scalability in the case of cluster analysis and densely connected graphs, as the number of edges will then grow as $\mathcal{O}(N^2)$.

Alternatively, and more commonly, a candidate partition can be represented by explicitly defining connections between data entities or graph nodes [49, 50]. Here, the representation of a partition is given as $\mathbf{r} = (r_1, \dots, r_N)$, a string of N integer values, where N denotes the number of data entities or nodes in the graph. In a clustering context, the associated interpretation is for $r_i \in \{1, \dots, N\}$ to represent the index of one other data entity which x_i is connected to, as illustrated in Fig. 6.2c. In a community detection setting (or a clustering setting with additional assumptions), a more compressed form of this representation becomes available by exploiting the concept of an adjacency list: here $r_i \in \{1, \dots, L_i\}$ refers to an index into the original graph's adjacency list for node x_i , of length L_i . Although an integer encoding is used in both cases, in the compressed form the possible alleles (range of integer values) may vary across genotype positions.

As before, an additional decoding step is required to interpret such an integer string as a partition. First, a candidate graph is constructed, containing the set of N nodes and edges between entities x_i and x_j , iff $r_i = x_j$ or, in the compressed encoding, if x_j corresponds to the r_i -th entry in the adjacency list of x_i (or vice versa). Subse-

quently, the connected components of this graph are determined, and each of them is interpreted as a separate cluster or community. Similar to the binary representation discussed above, this approach can be highly effective in small problem instances, but can suffer from poor scalability in the case of cluster analysis and densely connected graphs. While the integer string is restricted to length N , the number of the allele choices can still grow as a function of N , leading to an exponential increase in the size of the solution space, which is highly redundant.

In cluster analysis, the scalability issues of the locus-based adjacency representation have been addressed in different ways [17, 18, 24, 65]. Firstly, by limiting the adjacency list for each node to a reduced number of possibilities, e.g., representing the nearest neighbors of each data point. Secondly, by designing specialized initialization schemes, exploiting minimum-spanning trees, in order to bias the search process toward the most promising edges of the graph. Lastly, by pre-processing the data set in advance, so that only the most relevant decisions become the focus of the optimization process [17]. In community detection, node and edge centrality may be used as additional sources of heuristic bias, during the initialization and variation stages of the search [27].

6.4.2 Objective Functions

Assuming the use of any competent optimizer (as well as the absence of harmful bias at the representation level), the choice of objective function is arguably the most important decision determining the outcome of an unsupervised learning task. Building on the general problem formulation derived in Sect. 6.3, this section aims to highlight the different types of criteria we may choose to integrate as objectives for a given application, and the rationale behind this.

One of the most challenging aspects of cluster analysis and community detection is the lack of a clearly defined quality criterion. As a consequence, the literature contains a wide variety of objective functions, each prioritizing different (and sometimes combinations of) quality aspects of a partition [2, 35]. Furthermore, while there are conceptual similarities between the objective functions used in cluster analysis and community detection, the two fields generally use different approaches. For example, both fields feature measures that consider the strength of inter-group relations, but clustering objectives typically do so by assessing the maximum or average dissimilarity within a cluster (e.g., diameter [28] or within-cluster variance [38]), while the field of community detection relies on variants of modularity, which assess within-cluster edge densities relatively to a statistical null model [46].

The diversity of potential objectives arises due to the range of complementary, yet conflicting aspects that we attribute to a good partition of a graph or data set, and the difficulty in defining such aspects mathematically. Generally, there is an agreement that partitions are derived with the aim of identifying groups that are homogeneous and mutually distinct. However, there are very different ways in which homogeneity and distinctiveness can be described mathematically. For example, within-group

homogeneity may be translated into a requirement to resemble all other entities in the same group, or it may suffice to resemble a cluster representative, or even just another subset of entities in the group; such different assumptions will result in drastically different groups in many data sets.

A direct implication of this ambiguity is that the available objective functions typically capture some but not all desirable aspects of a clustering or community detection problem. Thus, there is a need to carefully align objectives with the modeling intentions in a given application. Furthermore, when desirable partition attributes are not clear in advance, the simultaneous consideration of multiple objectives can provide a more comprehensive problem formulation that avoids premature decisions on the importance of different quality facets.

Meta-heuristic algorithms are well-suited to assist with the above challenges. Unlike purpose-built heuristics, they are sufficiently flexible to allow for experimentation with different objective functions. Moreover, due to their population-based approach, multi-objective evolutionary algorithms (MOEAs) provide a particularly convenient approach to the simultaneous optimization of multiple objectives, and the exploration of trade-offs between them. Cluster analysis and community detection are often applied for exploratory data analysis, i.e., to assist with the understanding of a novel data set. MOEAs facilitate the exploration of a select range of candidate solutions, and the learning associated with this can add distinct value to the process.

Multiple objectives in clustering and community detection do not solely derive from the definition of multiple quality criteria. A further complicating factor in the definition of a partition is the decision on the number of groups to create, K . Algorithms like k -means [39] side-step this issue by making K a user-defined parameter. However, where algorithms are required to determine K as part of the optimization process, integration of objectives with opposing biases (with respect to K) has been found to be particularly useful [17, 24]. Elsewhere, the value of K has been explicitly employed as an additional optimization criterion [37, 48, 62].

As highlighted in Sect. 6.3, multiple objectives can also play the role of assessing partition quality with respect to multiple views of a data set. If a set of entities is characterized by multiple feature or dissimilarity spaces, we will usually wish to identify partitions that take into account, and are consistent with, all available information. Where the relative importance or reliability of these views are unknown, capturing them through individual objectives, and optimizing these using a Pareto optimization approach, can help with exploring a range of trade-offs and understanding both conflict and alignment between them [7, 18, 36, 55].

Finally, unsupervised learning settings may involve additional constraints or objectives that need to be considered in characterizing optimal partitions. These may include, for example, considerations related to group sizes [40], prior domain knowledge [9], or aspects of fairness [8]. Multi-objective approaches provide a convenient mechanism to integrate any of these into the search process.

6.4.3 Model Selection

The previous section has highlighted the potential benefits of problem formulations integrating a number of optimization criteria, either to capture desirable properties of a partition more comprehensively or to account for the availability of multiple views, and explore the trade-offs between them. However, one of the challenges this presents is the need for final model selection: typically, an unsupervised learning scenario requires the ultimate selection of a single solution.

Problems of model selection equally arise in traditional approaches to cluster analysis and community detection. For example, the use of k -means commonly involves running this algorithm for a range of possible cluster numbers, and using an additional cluster validity index [3, 35] to support the selection of a final solution from the set of alternatives obtained. Similar approaches can and have been suggested in an evolutionary clustering context, including scenarios involving multiple criteria [5, 15] and multiple data views [36]. A cluster validity index frequently used for model-selection purposes is the *Silhouette Width* [54], due to its prominence in the field of clustering. In general, measures for model selection may be at their most effective if they do not fully coincide with the objectives already considered during the optimization stage of the search process [26], i.e., when they provide complementary guidance.

More innovative approaches focus on analyzing the shape of the approximation set returned by multi-objective optimizers, using this to pinpoint the most promising solution alternatives. For example, one approach to solution selection focuses on the identification of *knee* solutions [23, 24, 42, 57], i.e., solutions that present particularly promising performance trade-offs between the objective functions considered. This builds on ongoing work in the evolutionary multi-objective optimization literature [10, 29, 58], but also shares similarities with existing approaches from the clustering literature, such as the Gap statistic [61] and the elbow method [60].

Some authors have explored the interpretation of the final approximation set as a clustering ensemble [21]. Drawing on previous work in ensemble learning [13], the aim is to derive a single consensus solution that best represents the information captured in the approximation front [25, 43, 44, 52, 65]. The rationale behind this approach is that all solutions in the approximation set contain useful information regarding the correct partition. However, potential disadvantages include: the focus on a majority consensus, which means that information from the extremes of the approximation front, in particular, may be insufficiently represented; and the assumption that all nondominated partitions are equally reliable, which is not necessarily the case and can affect the resulting consensus.

Model selection is a challenging task and remains an active area of research. As analyzed in [19], all of the above-discussed strategies have shown some success during empirical evaluations, but their limitations are rooted in assumptions that may not hold in every scenario. As an alternative, the potential of machine learning to capture the complexities of the task is showcased [19] reporting promising results. A supervised learning approach is explored, relying on the initial construction of

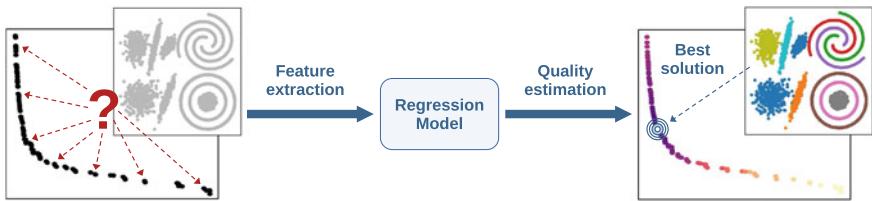


Fig. 6.5 Supporting model selection in multi-objective clustering through machine learning. Each individual solution in the approximation front is characterized (feature extraction). Then, the quality of solutions is estimated by a (previously trained) regression model. The solution with the highest estimated quality is identified and selected as the final result [19]

a regression model that is later exploited to estimate the quality of solutions in a given approximation front (see Fig. 6.5). Model construction depends on: (i) a collection of sample problems with known solutions; (ii) a set of training fronts for each sample problem, generated by some algorithm; (iii) the extraction of features for all solutions in the training fronts; (iv) the quality assessment of these solutions by direct comparison with the known correct solution using an *external validity index*¹; and (v) the use of a machine learning method to build the regression model, using the extracted features as explanatory variables and the solution-quality measurements as the response variable. The current approach to feature extraction seeks to characterize nondominated points based on individual properties of the partitions they encode, their relation to other solutions in the approximation front, as well as global aspects of the entire front and the particular clustering problem being solved. Feature extraction is a key issue in this methodology and deserves further investigation.

6.4.4 Choice of Optimizer

A variety of evolutionary algorithms and alternative meta-heuristics have been applied to the tasks of clustering and community detection [1, 20, 34, 47]. As discussed earlier, the choice of representation (with associated variation operators) and the choice of optimization criteria are arguably the most important design aspects in determining the performance of these approaches. Furthermore, these aspects ultimately predetermine the general class of algorithms that should be used.

For example, the continuous encoding implicit to standard prototype-based representations allows for the use of optimization methods designed for optimization over continuous spaces. In contrast, the use of a graph-based representation necessitates the use of integer or binary optimization approaches. Similarly, choices about the number and nature of objectives have driven decisions toward the adoption of multi-objective and many-objective optimization approaches. Given identical choices of

¹ Cluster validity indices can be external or internal, depending on whether or not they depend on knowledge of the correct partition (ground truth) to determine solution quality.

representation, variation operators, and objective(s), there are bound to remain performance differences between distinct choices of optimizers. There has been limited investigation of this aspect in the literature [41, 45], largely (we speculate) due to prioritization: in terms of recovery of the ground truth, the associated performance differences are likely to be outweighed by those reflecting fundamental changes in problem formulation. Nevertheless, one way of addressing this shortfall, going forward, may be the generation and inclusion of benchmarks derived from clustering problems into benchmark suites routinely used for the development of general-purpose meta-heuristics.

6.5 Final Perspectives

As highlighted above, evolutionary approaches have an important role to play in helping to explore novel, and potentially more comprehensive, formulations of clustering and community detection. In this chapter, we have aimed to highlight those design issues that we find to be of particular relevance in defining the capabilities of such approaches. However, in addition to the points highlighted so far, there are other areas requiring further development.

Whether it refers to the size of the problem (data set) and its dimensionality, or to the number of optimization criteria, scalability remains a key challenge for the use of evolutionary algorithms (and other meta-heuristics) in unsupervised learning applications. Regarding problem size, recent work has highlighted some progress and core mechanisms that can be exploited, including the use of stratification [14] and changes to problem resolution which impact on the granularity of the search [17]. Increases in the number of optimization criteria, either to handle multiple performance aspects or data views, can benefit from current developments in the area of many-objective optimization [36].

Finally, the field needs to address issues around the uptake of the algorithms by practitioners and the mainstream machine learning literature. While promising applications of evolutionary clustering approaches have been reported [1], and several different implementations are now available through GitHub, uptake of the techniques outside of the evolutionary computation community remains limited. Future work needs to prioritize the dissemination of these methods within the machine-learning and practitioner community, and the translation of algorithms into standard software packages, with a view of encouraging increased adoption of this research into practical applications. It is crucial that such messaging is transparent in terms of the relative strengths and weaknesses of current approaches. As set out in our chapter, evolutionary computation offers advantages when it comes to comprehensive, flexible problem formulations, and the literature has highlighted this, e.g., by demonstrating how changes in problem formulation can drive robustness toward a wider range of data properties than traditional algorithms [16, 22, 24]. However, it is clear that these benefits also come at significant computational cost, creating barriers for applications requiring fast throughput or handling of very large data sets. Helping

practitioners understand the resulting trade-offs relies on routinely including comparisons to state-of-the-art non-evolutionary machine learning techniques, as well as considerations of time complexity [30], into published work. The best choice of algorithm for a given unsupervised learning setting will always depend on a range of factors, and we need to ensure that the place of evolutionary-based approaches is better understood.

References

1. Aljarah, I., Faris, H., Mirjalili, S.: Evolutionary data clustering: Algorithms and applications. Springer, Berlin (2021)
2. Aljarah, I., Habib, M., Nujoom, R., Faris, H., Mirjalili, S.: A comprehensive review of evaluation and fitness measures for evolutionary data clustering. In: Aljarah, I., Faris, H., Mirjalili, S. (eds.) Evolutionary Data Clustering: Algorithms and Applications, pp. 23–71. Springer Singapore, Singapore (2021)
3. Arbelaitz, O., Gurrutxaga, I., Muguerza, J., Pérez, J.M., Perona, I.: An extensive comparative study of cluster validity indices. *Pattern Recogn.* **46**(1), 243–256 (2013)
4. Bandyopadhyay, S., Maulik, U.: Genetic clustering for automatic evolution of clusters and application to image classification. *Pattern Recogn.* **35**(6), 1197–1208 (2002)
5. Bandyopadhyay, S., Mukhopadhyay, A., Maulik, U.: An improved algorithm for clustering gene expression data. *Bioinformatics* **23**(21), 2859 (2007)
6. Bayá, A.E., Granitto, P.M.: How many clusters: a validation index for arbitrary-shaped clusters. *IEEE/ACM Trans. Comput. Biol. Bioinf.* **10**(2), 401–414 (2013)
7. Caballero, R., Laguna, M., Martí, R., Molina, J.: Scatter Tabu search for multiobjective clustering problems. *J. Oper. Res. Soc.* **62**(11), 2034–2046 (2011)
8. Chhabra, A., Masalkovaité, K., Mohapatra, P.: An overview of fairness in clustering. *IEEE Access* **9**, 130698–130720 (2021)
9. Davidson, I., Ravi, S.S.: Clustering with constraints: Feasibility issues and the k-means algorithm. In: Proceedings of the 2005 SIAM International Conference on Data Mining, pp. 138–149. SIAM (2005)
10. Deb, K., Gupta, S.: Understanding knee points in bicriteria problems and their implications as preferred solution principles. *Eng. Optim.* **43**(11), 1175–1204 (2011)
11. Falkenauer, E.: Genetic Algorithms and Grouping Problems. Wiley Ltd, Chichester (1998)
12. Fortunato, S.: Community detection in graphs. *Phys. Rep.* **486**(3), 75–174 (2010)
13. Fred, A.L.N., Jain, A.K.: Combining multiple clusterings using evidence accumulation. *IEEE Trans. Pattern Anal. Mach. Intell.* **27**(6), 835–850 (2005)
14. Garcia-Piquer, A., Bacardit, J., Fornells, A., Golobardes, E.: Scaling-up multiobjective evolutionary clustering algorithms using stratification. *Pattern Recogn. Lett.* **93**, 69–77 (2017)
15. Garcia-Piquer, A., Sancho-Asensio, A., Fornells, A., Golobardes, E., Corral, G., Teixidó-Navarro, F.: Toward high performance solution retrieval in multiobjective clustering. *Inf. Sci.* **320**, 12–25 (2015)
16. Garza-Fabre, M., Handl, J., Knowles, J.: A new reduced-length genetic representation for evolutionary multiobjective clustering. In: Trautmann, H., Rudolph, G., Klamroth, K., Schütze, O., Wiecek, M., Jin, Y., Grimme, C. (eds.) Evolutionary Multi-Criterion Optimization, pp. 236–251. Springer International Publishing, Münster (2017)
17. Garza-Fabre, M., Handl, J., Knowles, J.: An improved and more scalable evolutionary approach to multiobjective clustering. *IEEE Trans. Evol. Comput.* **22**(4), 515–535 (2018)
18. Garza-Fabre, M., Handl, J., José-García, A.: Evolutionary multi-objective clustering over multiple conflicting data views. *IEEE Trans. Evolut. Comput.* (2022)

19. Garza-Fabre, M., Sánchez-Martínez, A.L., Aldana-Bobadilla, E., Landa, R.: Decision making in evolutionary multiobjective clustering: a machine learning challenge. *IEEE Access* 1–22 (2022)
20. Gharehchopogh, F.S., Abdollahzadeh, B., Khodadadi, N., Mirjalili, S.: Chapter 20 - metaheuristics for clustering problems. In: Mirjalili, S., Gandomi, A.H. (eds.), *Comprehensive Metaheuristics*, pp. 379–392. Academic (2023)
21. Golalipour, K., Akbari, E., Hamidi, S.S., Lee, M., Enayatifar, R.: From clustering to clustering ensemble selection: a review. *Eng. Appl. Artif. Intell.* **104**, 104388 (2021)
22. Gong, C., Chen, H., He, W., Zhang, Z.: Improved multi-objective clustering algorithm using particle swarm optimization. *PLoS ONE* **12**(12), e0188815 (2017)
23. Gupta, A., Datta, S., Das, S.: Fuzzy clustering to identify clusters at different levels of fuzziness: an evolutionary multiobjective optimization approach. *IEEE Trans. Cybern.* **51**, 2601–2611 (2021)
24. Handl, J., Knowles, J.: An evolutionary approach to multiobjective clustering. *IEEE Trans. Evol. Comput.* **11**(1), 56–76 (2007)
25. Handl, J., Knowles, J.: Evidence accumulation in multiobjective data clustering. In: International Conference on Evolutionary Multi-Criterion Optimization, pp. 543–557. Springer (2013)
26. Handl, J., Knowles, J., Kell, D.B.: Computational cluster validation in post-genomic data analysis. *Bioinformatics* **21**(15), 3201–3212 (2005)
27. Handl, J., Ospina-Forero, L., Cann, T.: Multi-objective community detection for bipartite graphs. Under Submission (2022)
28. Hansen, P., Jaumard, B.: Minimum sum of diameters clustering. *J. Classif.* **4**(2), 215–226 (1987)
29. He, Z., Yen, G.G., Ding, J.: Knee-based decision making and visualization in many-objective optimization. *IEEE Trans. Evol. Comput.* **25**(2), 292–306 (2021)
30. Hruschka, E.R., Campello, R.J.G.B., Freitas, A.A., Ponce, A.C., de Carvalho, L.F.: A survey of evolutionary algorithms for clustering. *IEEE Trans. Syst., Man, Cybern., Part C (Appl. Rev.)* **39**(2), 133–155 (2009)
31. Ikotun, A.M., Ezugwu, A.E., Abualigah, L., Abuhamaja, B., Heming, J.: K-means clustering algorithms: a comprehensive review, variants analysis, and advances in the era of big data. *Inf. Sci.* **622**, 178–210 (2023)
32. Ishibuchi, H., Tsukamoto, N., Nojima, Y.: Evolutionary many-objective optimization: a short review. In: 2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence), pp. 2419–2426. IEEE (2008)
33. Jain, A.K.: Data clustering: 50 years beyond K-means. *Pattern Recogn. Lett.* **31**(8), 651–666 (2010)
34. José-García, A., Gómez-Flores, W.: Automatic clustering using nature-inspired metaheuristics: a survey. *Appl. Soft Comput.* **41**, 192–213 (2016)
35. José-García, A., Gómez-Flores, W.: A survey of cluster validity indices for automatic data clustering using differential evolution. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '21, pp. 314–322. ACM, New York (2021)
36. José-García, A., Handl, J., Gómez-Flores, W., Garza-Fabre, M.: An evolutionary many-objective approach to multiview clustering using feature and relational data. *Appl. Soft Comput.* **108**, 107425 (2021)
37. Liu, Y., Özyer, T., Alhajj, R., Barker, K.: Integrating multi-objective genetic algorithm and validity analysis for locating and ranking alternative clustering. *Informatica* **29**, 33–40 (2005)
38. MacQueen, J.: Classification and analysis of multivariate observations. In: 5th Berkeley Symposium Mathematics Statistics Probability, pp. 281–297 (1967)
39. MacQueen, J.: Some methods for classification and analysis of multivariate observations. In: Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, pp. 281–297. University of California Press (1967)
40. Malinen, M.I., Fränti, P.: Balanced k-means for clustering. In: Joint IAPR International Workshops on Statistical Techniques in Pattern Recognition (SPR) and Structural and Syntactic Pattern Recognition (SSPR), pp. 32–41. Springer (2014)

41. Martínez-PeñaLoza, M.-G., Mezura-Montes, E., Cruz-Ramírez, N., Acosta-Mesa, H.-G., Ríos-Figueroa, H.-V.: Improved multi-objective clustering with automatic determination of the number of clusters. *Neural Comput. Appl.* **28**, 2255–2275 (2017)
42. Matake, N., Hiroyasu, T., Miki, M., Senda, T.: Multiobjective clustering with automatic K-determination for large-scale data. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, GECCO '07, pp. 861–868. ACM, London (2007)
43. Mukhopadhyay, A., Maulik, U., Bandyopadhyay, S.: Multiobjective genetic algorithm-based fuzzy clustering of categorical attributes. *IEEE Trans. Evol. Comput.* **13**(5), 991–1005 (2009)
44. Mukhopadhyay, A., Maulik, U., Bandyopadhyay, S.: Multiobjective genetic clustering with ensemble among pareto front solutions: application to MRI brain image segmentation. In: 2009 Seventh International Conference on Advances in Pattern Recognition, pp. 236–239 (2009)
45. Mukhopadhyay, A., Maulik, U., Bandyopadhyay, S.: A survey of multiobjective evolutionary clustering. *ACM Comput. Surv.* **47**(4) (2015)
46. Newman, M.E.J.: Modularity and community structure in networks. *Proc. Natl. Acad. Sci.* **103**(23), 8577–8582 (2006)
47. Osaba, E., Del Ser, J., Camacho, D., Bilbao, M.N., Yang, X.S.: Community detection in networks using bio-inspired optimization: latest developments, new results and perspectives with a selection of recent meta-heuristics. *Appl. Soft Comput.* **87**, 106010 (2020)
48. Özyer, T., Liu, Y., Alhajj, R., Barker, K.: Multi-objective genetic algorithm based clustering approach and its application to gene expression data. In: Yakhno, T. (ed.) *Advances in Information Systems*, pp. 451–461. Springer, Berlin (2005)
49. Park, Y.J., Song, M.S.: A genetic algorithm for clustering problems. In: *Genetic Programming*, pp. 568–575. Morgan Kaufmann, Madison (1998)
50. Pizzuti, C.: Ga-net: a genetic algorithm for community detection in social networks. In: *International Conference on Parallel Problem Solving from Nature*, pp. 1081–1090. Springer (2008)
51. Pizzuti, C.: Evolutionary computation for community detection in networks: a review. *IEEE Trans. Evol. Comput.* **22**(3), 464–483 (2018)
52. Qian, X., Zhang, X., Jiao, L., Ma, W.: Unsupervised texture image segmentation using multiobjective evolutionary clustering ensemble algorithm. In: *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pp. 3561–3567 (2008)
53. Rothlauf, F.: *Representations for genetic and evolutionary algorithms*, 2nd edn. Springer, Berlin (2006)
54. Rousseeuw, P.J.: Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.* **20**, 53–65 (1987)
55. Saha, S., Mitra, S., Kramer, S.: Exploring multiobjective optimization for multiview clustering. *ACM Trans. Knowl. Discov. Data* **12**(4), 44:1–44:30 (2018)
56. Saxena, A., Prasad, M., Gupta, A., Bharill, N., Patel, O.P., Tiwari, A., Er, M.J., Ding, W., Lin, C.T.: A review of clustering techniques and developments. *Neurocomputing* **267**, 664–681 (2017)
57. Shirakawa, S., Nagao, T.: Evolutionary image segmentation based on multiobjective clustering. In: *IEEE Congress on Evolutionary Computation*, pp. 2466–2473 (2009)
58. Shukla, P.K., Braun, M.A., Schmeck, H.: Theory and algorithms for finding knees. In: *International Conference on Evolutionary Multi-Criterion Optimization*, pp. 156–170. Springer (2013)
59. Talbi, E.G.: *Metaheuristics from design to implementation*. Wiley (2009)
60. Thorndike, R.L.: Who belongs in the family. In: *Psychometrika*. Citeseer (1953)
61. Tibshirani, R., Walther, G., Hastie, T.: Estimating the number of clusters in a data set via the gap statistic. *J. R. Stat. Soc.: Ser. B (Stat. Methodol.)* **63**(2), 411–423 (2001)
62. Wahid, A., Gao, X., Andreae, P.: Multi-view clustering of web documents using multi-objective genetic algorithm. In: *2014 IEEE Congress on Evolutionary Computation (CEC)*, pp. 2625–2632. IEEE, Beijing (2014)
63. Zhang, Q., Li, H.: MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Trans. Evol. Comput.* **11**(6), 712–731 (2007)

64. Zhou, K., Martin, A., Pan, Q.: A similarity-based community detection method with multiple prototype representation. *Phys. A* **438**, 519–531 (2015)
65. Zhu, S., Lihong, X., Goodman, E.D.: Evolutionary multi-objective automatic clustering enhanced with quality metrics and ensemble strategy. *Knowl.-Based Syst.* **188**, 105018 (2020)

Chapter 7

Evolutionary Classification



Bach Nguyen, Bing Xue, Will Browne, and Mengjie Zhang

Abstract Classification is a supervised machine learning process that categories an instance based on a number of features. The process of classification involves several stages, including data preprocessing (such as feature selection and feature construction), model training and evaluation. Evolutionary computation has been widely applied to all these stages to improve the performance and explainability of the built classification models, where term for this research area is *Evolutionary Classification*. This chapter introduces the fundamental concepts of evolutionary classification, followed by the key ideas using evolutionary computation techniques to address existing classification challenges such as multi-class classification, unbalanced data, explainable/interpretable classifiers and transfer learning.

7.1 Introduction

Classification is one of the most important tasks in machine learning as it addresses many real-world problems [39, 120]. The task of classification is to assign a known categorical value (also known as a *class label*) to an instance based on the instance's properties (also known as *features*). An example of classification is email spam

B. Nguyen · B. Xue · M. Zhang (✉)

Centre of Data Science and Artificial Intelligence & School of Engineering and Computer Science, Victoria University of Wellington,
Wellington, New Zealand

e-mail: MengjieZhang@ecs.vuw.ac.nz

B. Nguyen

e-mail: Hoai.Bach.Nguyen@ecs.vuw.ac.nz

B. Xue

e-mail: Bing.Xue@ecs.vuw.ac.nz

W. Browne

School of Electrical Engineering and Robotics, Queensland University of Technology,
Brisbane, Australia
e-mail: Will.Browne@qut.edu.au

detection where the class labels are *spam* or *non-spam* and the features are the word frequencies in an email [22].

The label assignment is achieved by building a classifier model that takes features as inputs and outputs the class label. The classifier is built based on a set of labelled instances, called a *training set*. The quality of the built classifier is then examined by another set of instances, called a *test set*. Classification problems can be divided into two categories: binary classification and multi-class classification. A binary classification problem has only two class labels, while a multi-class classification problem has more than two class labels.

7.2 Evolutionary Computation (EC) for Classification: Fundamentals

Generally, the process of building a classifier can be considered a learning process that searches for an optimal classifier model based on a given training set. Since evolutionary computation (EC) has a potential global search ability, it has been widely applied to build classifiers for many real-world applications, such as text classification [48], medical diagnosis [89] and image classification [2, 3, 29].

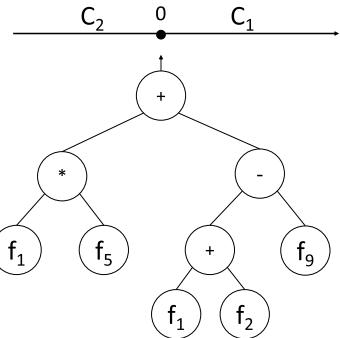
The classification performance depends significantly on the quality of the dataset. However, many datasets contain irrelevant or redundant features and noisy instances, which severely deteriorate the classification performance. To address these issues, EC has been applied to preprocess the dataset and/or remove associated negative artefacts during the training process, which can significantly improve the classification performance.

In this section, we will firstly discuss how two established EC approaches: Genetic Programming (GP) and Learning Classifier System (LCSs) can be used to perform classification directly. We will next show how EC can optimise parameters/hyperparameters of standard classification algorithms, and thus improving their classification performance. Finally, we will introduce typical evolutionary data reduction approaches, which are commonly used as preprocessing steps for classification.

7.2.1 GP for Classification: Binary Versus Multi-class Classification

GP has a great capacity for classification thanks to its flexible representation which can construct the necessary decision boundaries. GP can be utilised to construct different classifiers such as decision trees [14, 52, 99], classification rules [27, 92, 107] and discriminant functions [119, 129, 132]. Among these classifiers, evolving discriminant functions is the most common approach so will be the main focus of this chapter.

Fig. 7.1 GP for binary classification. For an instance, if the output of the tree (discriminant function) is greater than 0, the instance is assigned to class C_1 . Otherwise, the instance is assigned to class C_2 . Note that the tree uses only 4 features $\{f_1, f_2, f_5, f_9\}$ out of the 9 original features



Discrimination functions are simply a mathematical function whose variables are the features. To classify an instance, the instance's feature values are passed to the function that then outputs a single output value from the root node (assuming tree-based GP). The output value is compared with a threshold value (or values) to determine the class label of the instance. In binary classification, there are only two class labels, thus a single threshold is sufficient. Particularly, if the output value is greater than a given threshold, the instance belongs to one class, otherwise the instance belongs to the other class. Usually, the threshold is set to zero. Figure 7.1 presents a discriminant function evolved by GP for a binary classification problem. Notice that the evolved discriminant function does not use all features. In other words, GP can select only discriminative features to address a classification task, which is known as the “embedded” feature selection ability of GP.

For multi-class classification problems, GP needs to cope with more than two class labels, which is more challenging than binary classification. There are two main approaches extending GP for multi-class classification problems. The first approach is to treat an n -class classification problem (n is the number of class labels) as n binary classification problems [51]. Each binary classification problem is to separate one class from the remaining $(n - 1)$ classes, which is handled by a discriminant function with a threshold [60, 129, 132]. Thus, for an n -class classification problem, n discriminant functions are evolved. The second approach is to evolve a single discriminant function with $(n - 1)$ threshold values. The $(n - 1)$ threshold values define n intervals, and each interval represents a class. Therefore, the class of an instance is determined by the corresponding interval that the output value belongs to. More details on GP for multi-class classification will be presented in Sect. 7.3.

7.2.2 Learning Classifier Systems for Classification

Learning classifier systems (LCSs) are one of the few EC algorithms that can perform classification directly. LCSs represent a rule-based agent that typically consists of two main components: *a discovery component* (for example genetic algorithms) and

Instance	Features : Class	Features : Class	Features : Class
	1 0 1 1 0 0 : 1	1 0 1 1 0 0 : 1	1 0 1 1 0 0 : 1
Rule	Condition : Action	Condition : Action	Condition : Action
	1 0 # 1 # 0 : 1	1 0 # # # # : 0	0 1 # 1 # # : 1
Matching rule (Overspecific)		Matching rule (Overgeneral) (Incorrect Class)	Non-matching rule (Overspecific)

Fig. 7.2 Examples of LCSs rules presented in a classification problem with 6 features. Each rule has two parts: *condition* corresponding to the feature values and *action* corresponding to the class label [108]

a learning component (e.g., reinforcement learning) [108]. Rules are fundamental knowledge blocks in LCSs, which can be presented in many ways. In this section, we will describe the basic representation for data containing binary features, i.e., the feature value is either 0 or 1. Figure 7.2 shows some examples of LCSs rules, which have two main parts: *condition* of the feature values and *action* corresponding to the class label. The condition is a vector in which each vector element corresponds to a feature in the data. The element value has three possible values: 0, 1, or “don’t care” (denoted as #). The “don’t care” value can match any value of the corresponding feature, i.e., the corresponding feature will be considered to match both 0 and 1. If all the entries in a rule’s condition match all the corresponding feature values of an instance, the instance is considered to match the rule. Note that, matching does not require the class label of the instance and the action of the rule to be the same, i.e., LCSs can form a complete map that includes incorrect classifiers. Occasionally, it is easier to identify when an input state does not belong to a class than identify the class it does belong to. Figure 7.2 gives three examples of LCSs rules, where the first two rules match the instance while the last rule does not match the instance due to the difference in the second feature. It should be noted that LCSs rules can be interpreted as the following expression: “IF *condition* THEN *action*”. For example, the left most rule in Fig. 7.2: {1 0 # 1 # 0: 1} can be interpreted as:

IF {first bit is 1 AND second bit is 0 AND fourth bit is 1 AND sixth bit is 0}
THEN {the class label is 1}

Generalisation refers to the ability of a classifier in covering input samples, which assists to classify unseen/future sample. LCSs, unlike most machine learning algorithms, have an explicit generalisation measure, which is the number of “#” values in the condition. In Fig. 7.2, the left most rule is *overspecific* since too many features are specified (i.e., too few “#” values) while the middle rule is *overgeneral* since too few features are specified (i.e., too many “#” values). The overgeneral rule may match many instances, but the class labels of at least one instance is predicted wrongly due to the incorrectly removes feature information.

Since an LCS matches its rules with data instances, LCSs need to discover a set of collaborative rules to well cover the instance space. The number of rules usually

depends on several factors including the problem complexity and the representation. Noting multiple alphabets exist for classifying different types of real-world problems from bioinformatics to sleep patterns [109]. The training process of LCSs consists of a number of iterations. For each iteration, a single training instance is selected from the training set and the parameters of the matching rules will be adjusted to reflect the knowledge gained and a new rule will be added if no matching rules pre-exist. Once all the training instances have been considered, the training process returns to the first instance and selects from the training set again. Once the training process is finished, the last set of rules is returned as the set of classification rules.

To classify an unseen instance, the instance will be compared against all the rules in the set of classification rules. Each matched rule predicts a class label for the instance. A voting mechanism combines all the predicted classes and outputs the final class for the instance. A key advantage of LCSs is its learning technique with the highly desired benefit of transparency where its rules are human-interpretable, which is essential to eXplainable AI.

7.2.3 Vector-Based EC Algorithms for Classification

This subsection shows how to use EC algorithms with vector-based representations, such as Genetic Algorithms (GAs), Particle Swarm Optimisation (PSO) and Differential Evolution (DE) to tune the parameters of classification algorithms including Support Vector Machines, K-nearest Neighbour and Artificial Neural Networks.

7.2.3.1 Evolutionary Support Vector Machines (SVMs)

The main idea of SVMs is to build hyperplanes (decision boundaries in linear SVMs) that separate instances from different classes. The hyperplane is usually represented by a weight vector w , in which each vector element corresponds to an original feature. Therefore, the size of the weight vector is equal to the number of features. In case the instances are not linearly separable, SVMs will map the original data to high-dimensional data with an expectation that the instances are linearly separable in the high-dimensional space [40, 64]. Such a mapping is usually embedded in a kernel function. One of the most common kernel functions is the following Radial Basis Functions (RBFs):

$$K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right) \quad (7.1)$$

where x_i and x_j are two training instances and σ is the kernel width of the RBFs kernel.

The optimal weight vector w can be obtained by optimising the following objective function:

$$\begin{aligned}
 & \text{minimise} \quad \frac{1}{2} ||w||^2 + C \sum_{i=1}^k \xi_i \\
 & \text{subject to } \forall i : y_i \times (w_i x_i + b) > 1 - \xi_i \text{ and } \xi_i \geq 0
 \end{aligned} \tag{7.2}$$

where w is the weight vector to be optimised, C is an essential parameter that controls the trade-off between the training performance and the generalisation, and ξ_i is the allowable margin boundary for the i th class.

Vector-based EC algorithms, such as PSO and DE, can be applied to optimise C and σ . For multi-class classification, SVMs learn multiple hyperplanes and each hyperplane separates one class from all the other classes. If there are n classes, n pairs: $(C_1, \sigma_1), (C_2, \sigma_2), \dots, (C_n, \sigma_n)$ can be optimised. Thus, a candidate solution can be represented as a $2 \times n$ -vector consisting of $2 \times n$ parameters. The fitness value of the candidate solution is the classification performance of the SVMs trained with the $2 \times n$ parameters [31, 33, 44, 64]. This method has been applied to many real-world applications such as cloud computing [134], analog circuits design [125], electricity price forecasting [94] and healthcare [126].

7.2.3.2 Evolutionary K-Nearest Neighbour

K-nearest Neighbour (KNN) is a simple but effective classification algorithm [20]. KNN is known as a lazy classification algorithm since it does not build any model. The performance of KNN relies on three important factors: the distance function, the number of nearest neighbours k and the importance of the neighbours, features and classes [13]. Usually, the importance is represented by a weight vector where the larger weights indicate more important neighbours/features/classes. GAs, DE and PSO have been widely applied to tune these factors since they do not require a well-formulated objective function unlike mathematical optimisation algorithms.

A typical example of evolutionary KNN is proposed by Biswas et al. [13], where class-specific feature weight vectors are optimised by DE. The rationale is that different classes have different feature distributions, and thus their feature importances (i.e., feature weight vectors) are also different. Therefore, each class has its own feature weight vector, also known as a class-specific feature weight vector. Suppose there are c classes and d features, each candidate solution of DE is a vector consisting of $(c \times d + 1)$ elements. The last element is to optimise the number of neighbours k . Figure 7.3 illustrates the representation of a candidate solution in DE. The fitness value of each candidate solution is obtained by calculating the accuracy of KNN (with the parameters represented by the candidate solution) on the training set. It has been shown that the performance of KNN can be significantly improved by optimising the weight vectors and the number of neighbours k simultaneously [5, 13, 24].

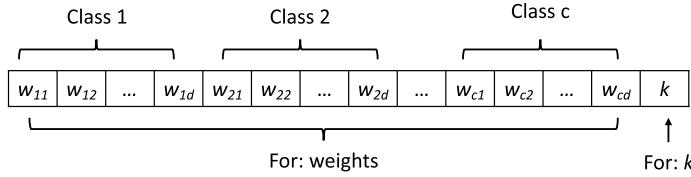


Fig. 7.3 Representation to optimise parameters for KNN: the first part consisting of $c \times d$ weights and the second part is to optimise the number of nearest neighbours k

7.2.3.3 Evolutionary Neural Networks

Artificial neural networks (ANNs) are inspired by simplified learning models of the human brain. An ANN has an input layer that contains a number of input nodes and each input node corresponds to a feature. When applied to classification, an ANN also has an output layer consisting of multiple output nodes and each output node typically shows the likelihood of an instance belonging to a class. Between the input layer and the output layer, there are usually several hidden layers. The nodes or “neurons” in the ANN are connected by connection weights. The main focus of training ANNs is to find the optimal set of connection weights plus a bias term. The classification performance of an ANN depends on two aspects: its architecture and its connection weights (i.e., the number of hidden layers and the number of neurons in each layer) [63].

The connection weights of ANNs are often optimised by a back-propagation learning process. Firstly, a loss function is used to measure the difference between an ANN’s output and the desired one. The difference (i.e., error) is propagated back by a gradient-based algorithm to adjust the connection weights. The training process is repeated for a pre-defined number of iterations. A main disadvantage of the back-propagation process is the potential of being trapped at local optima due to its gradient-based nature [21, 45, 46]. EC algorithms have a well-known global search ability, and have been applied to optimise an ANN’s connection weights. Each candidate solution is a vector consisting of the connection weights. The fitness of a candidate solution is the ANN’s classification performance. The results show that vector-based EC algorithms such as GAs, PSO and DE can outperform the back-propagation learning [32, 35, 71]. However, when the networks are larger, especially for deep neural networks (DNNs), the number of weight vectors is significantly increased, which is a challenging task for evolutionary algorithms to address. Recent works have reduced the search space by considering only the biases instead of all the weights [34] or converting large weight vectors set to a small set of orthogonal basis vectors [97]. Besides optimising connection weights, EC algorithms have also been also applied to automatically design or search for the architecture(s) of an ANN, especially DNNs. More details will be given in Sect. 7.7.3.

7.2.4 Evolutionary Data Reduction for Classification

In the era of big data, many classification problems have a large number of features and/or instances. Unfortunately, the quantity improvement does not usually lead to a quality improvement [59]. Among a large number of features, many of them do not provide any useful information to distinguish different classes, which are known as irrelevant features. Some features provide the same information as other features, so are known as redundant features. On the instance level, there might be outliers/noisy instances. Any classification algorithm, which overfits such instances, will not generalise well to unseen data, and thus the classification performance will be deteriorated significantly.

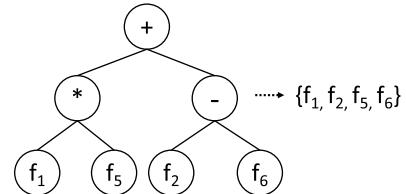
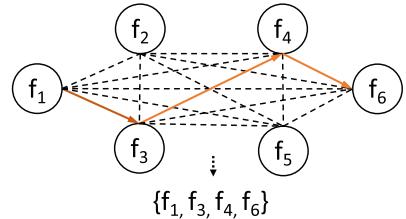
The above issues can be addressed by three main data reduction approaches: feature selection, feature construction and instance selection. Feature selection aims to select a small subset of relevant features from the original feature set [77]. Feature construction builds new high-level features based on the original low-level features. The number of new features is typically smaller than the number of original features, so feature construction can be considered a feature reduction approach. Instance selection, on the other hand, aims to remove noisy instances [83]. The key challenges of the three data reduction tasks are the large, discrete and rough search spaces, where EC algorithms usually cope well. The literature has shown that evolutionary data reduction approaches have achieved great classification performance while significantly reducing the amount of data [122]. The following subsections briefly explain how EC algorithms can be applied to achieve feature selection, feature construction and instance selection.

7.2.4.1 Evolutionary Feature Selection

Suppose there are d original features, the total number of possible feature subsets is 2^d , which exponentially increases with respect to the number of features. Each feature subset is a candidate solution and the key question to enable EC to address these tasks is how to represent a feature subset. Generally, there are three main feature selection representations: vector-based representations, tree-based representations and graph-based representations [77].

Among the three representations, the vector-based representation is the most common one. Figure 7.4 shows an example of a vector-based representation, where each vector element corresponds to an original feature. The element's value shows whether the corresponding feature is selected or not. Vector-based representations can be further divided into two main sub-categories: binary vectors and continuous vectors. In the binary one, each vector element is a binary value, where “1” indicates that the corresponding feature is selected while “0” indicates that the corresponding feature is discarded. In the continuous one, each vector element is a continuous value in the range $[0, 1]$. If the vector element is greater than a threshold value (set to 0.6 in Fig. 7.4), the corresponding feature is selected. Otherwise, the corresponding

	f_1	f_2	f_3	f_4	f_5	f_6	
Binary vector	1	0	1	1	0	0	$\rightarrow \{f_1, f_3, f_4\}$
Continuous vector	0.7	0.2	0.1	0.3	0.4	0.9	$> 0.6 ? \rightarrow \{f_1, f_3, f_4\}$

Fig. 7.4 Vector-based representation for feature selection**Fig. 7.5** Tree-based representation for feature selection**Fig. 7.6** Graph-based representation for feature selection

feature is discarded. While the binary one has been widely used by GAs [4, 123], the continuous one has been widely used by PSO [23, 75, 78] and DE [116].

The tree-based representation is mostly used in GP. The idea is that if a feature appears in a GP-based classifier (Sect. 7.2.1), it indicates that the feature can provide relevant information to distinguish different classes. Thus, on top of explicit feature construction, feature selection can be achieved by selecting all or some of the features in a tree-based classifier as shown in Fig. 7.5.

In graph-based representations, each feature is a node in the graph. If a node is visited, the corresponding feature is selected. Figure 7.6 shows an example of graph-based representation where the solid arrow path connects the selected features. Graph-based representations are mostly used by ant colony optimisation (ACO) [66, 85].

Figure 7.7 illustrates the overall feature selection system consisting of two main components: subset discovery and subset evaluation. The subset discovery component is responsible for generating new feature subsets. Depending on the representation, the subset discovery can be performed by a suitable evolutionary algorithm. For example, PSO, DE and GAs are usually used to generate candidate solutions represented by a vector-based representation. The subset evaluation component then measures the goodness of the generated subsets. Based on the feedback from subset

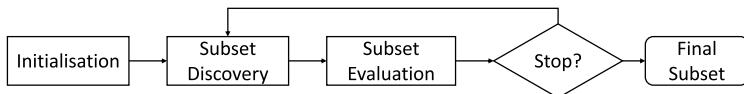


Fig. 7.7 Overall feature selection system

evaluation, subset discovery is expected to generate more promising feature subsets. The “generating-evaluation” process is repeated until a stopping criterion is met, and then the best feature subset is the final solution. Two common stopping criteria are the maximum number of generations and the maximum number of evaluations.

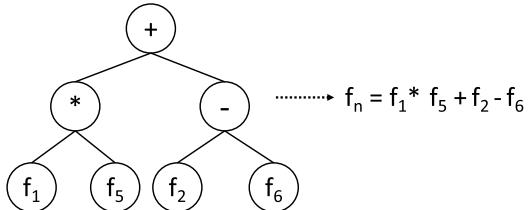
Based on the evaluation criterion, feature selection methods can be divided into three categories: wrapper, filter and embedded approaches. The wrapper approach trains a classification algorithm multiple times to evaluate all the candidate subsets, which considers the interactions between the classification algorithm and the feature subsets. In contrast, the embedded approach trains an embedded classification algorithm and selects features based on the trained classifier. For example, we can train a GP-based classifier and then select features appearing in the final tree. Instead of using classification algorithms, the filter approach uses statistical or distance measures to evaluate feature subsets. Some common filter measures are Relief [110], mutual information [80] and correlation measures [38]. Among the three approaches, the wrapper one usually achieves the highest classification performance but requires intensive computational resource. The filter one is the most efficient one but usually achieves the lowest classification performance. The embedded one comes between the wrapper and the filter in terms of the classification performance and the computational cost.

7.2.4.2 Evolutionary Feature Construction

The goal of feature construction is to build new high-level features that are functions of the original features, but leads to simpler decision boundaries. The main difference between feature selection and feature construction is that feature construction builds new features while feature selection only selects a subset of original features (i.e., no new features). Feature construction is particularly useful when the original features are not informative enough. Among EC algorithms, GP has been widely applied to achieve feature construction since its flexible representation can represent functions naturally [102]. GP does not require any pre-defined model structure for a new feature and any operators can be included in a GP program. Figure 7.8 shows a new feature (f_n) represented by a GP tree.

The overall system of feature construction is very similar to the system of feature selection in Fig. 7.7. The only difference is that the discovery component generates new high-level features instead of feature subsets as in feature selection. When the number of classes is small (i.e., binary classification), building a single high-level feature could be sufficient [73, 112]. However, when the number of classes is large,

Fig. 7.8 Tree-based representation for feature construction



we usually need to build multiple high-level features, which can be achieved by multi-tree GP. Particularly, each individual in multi-tree GP consists of a set of trees. A common approach is to associate each tree to a class. Each tree differentiates the instances from the tree's corresponding class to the instances from the other classes. The results show that such association results in much better performance than single-tree GP or multi-tree GP without any association [102]. Nguyen et al. [81] use a hybrid vector-tree representation to perform feature selection and feature construction simultaneously, which achieves better performance than performing each of the two tasks separately.

7.2.4.3 Evolutionary Instance Selection

The goal of instance selection is to select a subset of instances that are not noisy or redundant, which is similar to feature selection. We can use the same representations of feature selection to achieve instance selection. The only difference is that each vector entry (in the vector-based representation) or node (in the tree/graph-based representation) corresponds to an original instance rather than a feature. EC algorithms have been successfully used for instance selection since they do not require any assumption regarding data structure or data distribution [19, 55].

7.3 GP for Multi-class Classification

Among the EC algorithms, GP is one of the few algorithms that can perform classification directly (without relying on any external classification algorithm). For binary classification problems, GP can use a single threshold (usually 0) to distinguish between the two classes. However, real-world applications usually have more than two classes, also known as multi-class classification problems, which are more challenging for GP to address. This section discusses the basic ideas used to extend GP for multi-class classification.

7.3.1 Program Class Translation Based on Single Evolved Tree-Based Programs

Since each individual in standard GP is typically represented by a single tree, such representation with a single threshold is not readily applied to classify more than two classes. This subsection introduces two basic but powerful approaches that allow a single tree-based program to cope with multi-class classification. The first approach is to have multiple thresholds, also known as *boundary translations*. The second approach is to let each tree program represent a set of class distributions, also known as *probability/statistical GP*.

7.3.1.1 Static Versus Dynamic Boundary Translations

One way to extend GP for n -class classification is to use $(n - 1)$ threshold values, i.e., $\{T_1, T_2, \dots, T_{n-1}\}$, to form n intervals corresponding to n classes. An example is given in Fig. 7.9, where the interval $(-\infty, T_1]$ corresponds to *Class 1*, the interval $(T_1, T_2]$ corresponds to *Class 2* and so on. In other words, the threshold values are the boundaries between different classes. These threshold values can be either *statically* pre-defined or *dynamically* updated during the evolutionary process [65].

In the static approach, the set of $(n - 1)$ thresholds are manually pre-defined. However, it is hard to pre-define the optimal threshold values. Furthermore, the static approach enforces a class order, for example the distance between *Class 1* and *Class 2* is much farther than the distance between *Class 1* and *Class 3*, which might not be true in the dataset. An early version of this static method was to use an “equal” interval with a single parameter/threshold value to define different classes, which was used for multi-class object detection [128].

An alternative approach is to allow the thresholds to be updated during the evolutionary process, so the class boundaries are adjusted to better suit a certain classification problem. Zhang and Smart [130] proposed two methods for dynamic threshold values based on the GP population. The first method is regarded as centred dynamic class boundary determination with an example given in Fig. 7.10. The first step is to calculate the output values of all the GP programs on all the training instances. Then, for each class, its class centre is obtained by averaging the output values of

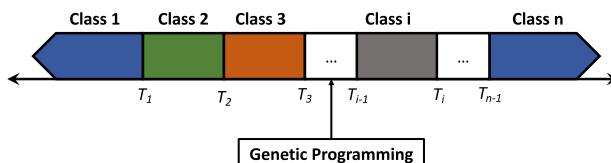


Fig. 7.9 Static boundaries of n classes

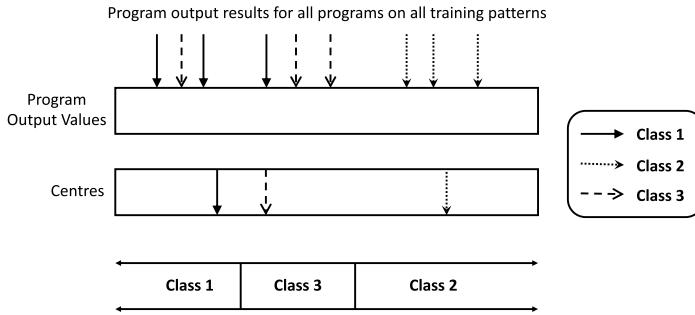


Fig. 7.10 Centred dynamic class boundaries

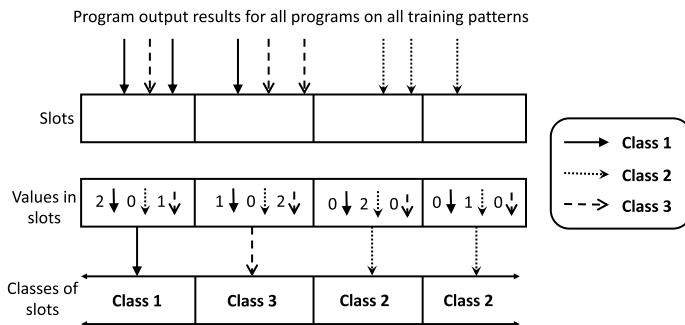


Fig. 7.11 Slotted dynamic class boundaries

all the instances belonging to the target class. The threshold value or class boundary between every two classes is the middle point of the two adjacent class centres.

The second method is regarded as slotted dynamic class boundary determination, with an example given in Fig. 7.11. The first step is to split a closed range into a certain number of slots and the task is to determine which class a slot belongs to. It is suggested that the range $[-25, 25]$ with 200 slots would be sufficient if the input features are scaled into $[-1, 1]$. After splitting, the output values are obtained by applying all the GP programs to the training instances, where each output value falls in a certain slot. For each slot, the number of training instances in each class is recorded, and the majority class (i.e., the class with the largest number of instances in the slot) is assigned to the slot. If there is no training instance in a slot, the slot will be assigned to the class of the nearest neighbouring slot.

The results indicate that the static approach can perform well on relatively easy classification problems where the class labels are ordinal. The dynamic approach achieves much better performance when the class labels are nominal (unordered). Among the two dynamic methods, the slotted dynamic class boundary determination is recommended for a complicated classification problem [130].

7.3.1.2 Probability/Statistical GP

Although the class boundary approaches are intuitive, they usually require certain prior domain knowledge. To address the issue, Zhang and Smart [131] integrate the probability theory and GP to build classification rules. Instead of determining class boundaries, the idea is to assume that the distribution of each class follows a Gaussian distribution, which is defined by a mean and a standard deviation. The two values can be obtained by calculating the mean and standard deviations of the output values when applying GP programs to the training instances from a target class.

In probability GP, each GP program can form a set of class distributions for all the classes and the class distribution can be presented as a normal curve. Figure 7.12 shows two bell curves for two classes. In the ideal case, the GP program should classify all the training instances correctly, i.e., the two curves should be clearly separated as in Fig. 7.12c. Thus, the area of the overlapping region can measure the fitness of a GP program. The smaller the area, the better the program. For multi-class classification, the fitness measure is the sum of the overlapping region between every two classes.

7.3.2 Strongly Typed GP for Multi-class Classification

Standard GP usually works on one type of data while complex classification problems may involve multiple types of data. To address this issue, Montana [70] proposed an enhanced GP version called Strongly Typed GP (STGP). For terminal nodes, STGP specifies their data types such as *Float*, *Integer* or *Array*. For internal (function) nodes, STGP also specifies their *input* and *output* data types. Accordingly, the input data type of an internal node has to match the output data types of its children nodes. STGP has been applied to complex classification problems that require different function types, such as cancer diagnosis [56], credit control [14], software quality classification [49, 50] and image classification [10, 12, 58].

To provide intuitive explanation, we take the image classification task as an illustrative example. An image content is represented by its pixel values, i.e., low-level features. However, it has been known that using more informative high-level features, which are functions of low-level features, usually results in a better classifi-

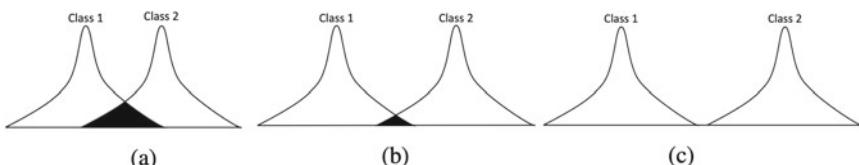


Fig. 7.12 Examples of the overlap between class distributions. **a** Worse fitness with large overlap; **b** Better fitness with small overlap; **c** Best fitness with no overlap

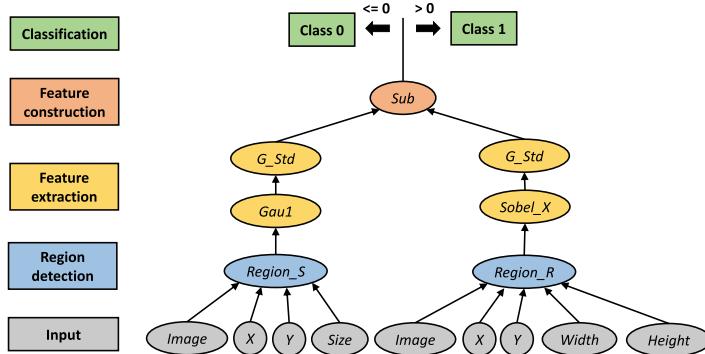


Fig. 7.13 Strongly Typed GP (STGP) for image classification with multiple layers [9, 11]. Different layers have different functionalities and are shown in different colours

cation performance. The process of extracting high-level features from an image is called feature extraction. Traditional image classification approaches perform feature extraction and classification as two separate steps, which “ignores” the complement between the extracted features and the classification algorithm. In contrast, STGP can perform feature extraction and classification simultaneously within in a single evolved GP program. A typical work was proposed by Bi et al. [9, 11] where the idea is to have various layers, i.e., multi-layer GP (MLGP), with different functionalities. In this work, a GP program has five main layers as showed in Fig. 7.13. The first (bottom) layer is an input layer with four possible types of each terminal: *Image* represents an input image, (X, Y) are the top left of the region, *Size* is the size of a square region, and $(Width, Height)$ are the width and height of a rectangle region. The second layer is to perform region detection with two possible operators: *Region_S* for a square region and *Region_R* for a rectangle region. While *Region_S* requires a *Size* input, *Region_R* requires $(Width, Height)$. The third layer is to perform feature extraction which uses different traditional feature extraction methods such as *Sobel*, *LBP*, *G_Std* and *Gau1* to extract features from the detected regions. The fourth layer is to perform feature construction, which builds high-level features based on the features extracted from the feature extraction layer. The fifth (top) layer performs classification based on the output of the fourth layer.

The results show that STGP not only achieves better classification than non-GP approaches but also generates interpretable features. Figure 7.14 shows an evolved GP program achieving 100% accuracy on a face classification problem. The left side of the program extracts features from a rectangle area between the two eyes of a face. The right side of the program extracts features from a rectangle region covering the lower left side of the face including the left mouth part of a face. Particularly, the surprised faces have wider mouth and bigger (whiter) eyes than the happy faces.

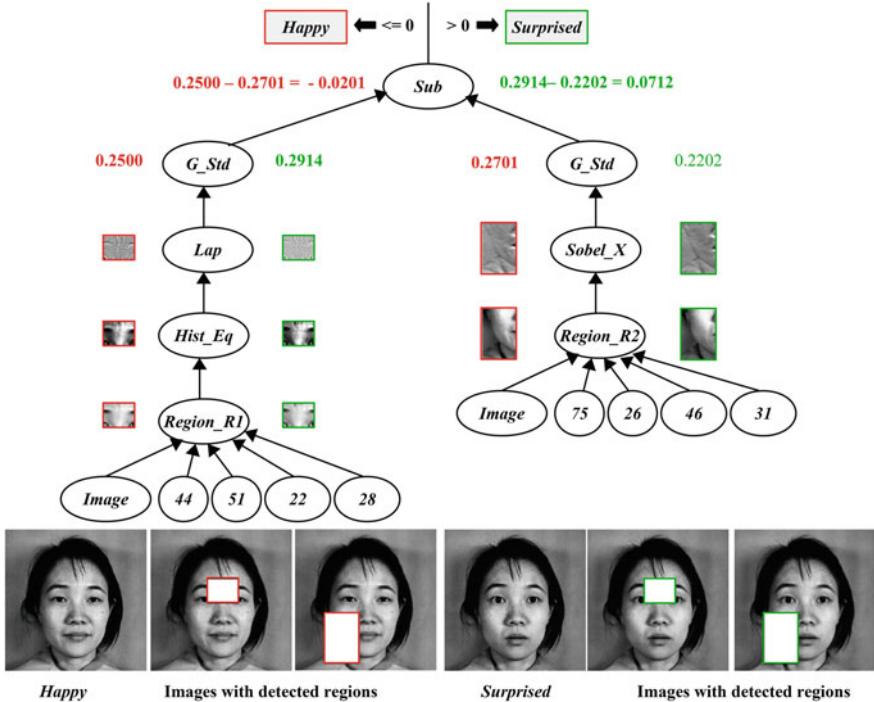


Fig. 7.14 An explainable tree evolved by MLGP for the JAFFE dataset [9]

7.3.3 Multi-tree GP for Multi-class Classification

Besides extending single-tree GP for multi-class classification, one can also apply multi-tree GP in which each GP individual is a set of trees [72, 118]. Particularly, for a n -class classification problem, each individual consists of n trees: $\{T_1, T_2, \dots, T_i, \dots, T_n\}$. The question is how to determine the instance's class based on n outputs of the n trees. The most straightforward approach is to directly compare the n outputs, then assign the class with the largest output to the instance, also known as the “winner-takes-all” approach [29].

Another approach is to use the output of T_i to predict whether an instance belongs to the i th class (C_i) or not, which is a binary classification problem. Formally, for an instance x :

$$\text{If } T_i(x) \geq 0 \text{ and } T_j(x) < 0 \text{ for all } j \neq i \text{ then } x \text{ belongs to class } C_i$$

Thus, multi-tree GP decomposes a n -class classification problem into n binary classification problems which can be solved simultaneously by n trees with a single threshold value of 0. Figure 7.15 represents an example of multi-tree GP for multi-class classification.

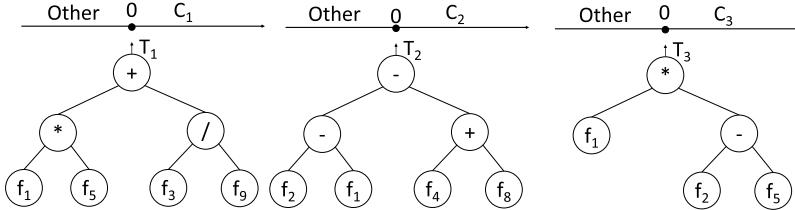


Fig. 7.15 Multi-tree GP for a classification problem containing 3 classes

Although the multi-tree representation fits multi-class classification naturally, the standard classification accuracy cannot be applied as in single-tree representation. For example, in a 3-class classification problem, the output of an example in the *first* class should be [1, 0, 0]. However, a 3-tree individual may output [1, 0.9, 0] which can then use the “winner-takes-all” approach to classification. Thus, we need to count how many instances are correctly classified by each *tree* in a GP individual. In general, multi-tree representation usually achieves better classification performance than single-tree representation. However, multi-tree representation requires a careful design of evolutionary operators, especially when there are a large number of classes [102].

7.3.4 Linear GP for Multi-class Classification

Linear Genetic Programming (LGP) is a variant of GP, where each individual is a linear sequence of instructions [15]. The instructions are operated on a number of variables, called *registers*. Particularly, LGP uses original features as *input registers* to calculate one or more *output registers*. When applies to an n -class classification problem, LGP can build n output registers $\{r_1, r_2, \dots, r_n\}$, where the r_i register corresponds to the i th class. Given an instance, an LGP program outputs n values from the n output registers. The class with the highest value is assigned to the instance [25, 30]. An example of LGP for multi-class classification is given in Fig. 7.16. The example shows that LGP can co-evolve n sub-programs (or registers) for n classes. Furthermore, a register can be re-used to define other registers, which can reduce the number of original features in comparison with using the tree-based representation [30]. Besides defining the set of registers, users also define an *instruction set* for LGP, which consists of two main instruction types: *operations* (such as arithmetic/Boolean operations) and *conditional branches* (i.e., *if* statements).

It can be seen that LGP (phenotype, graphs) and Artificial Neural Networks (ANNs) have quite similar structure. In fact, they can be seen as alternatives and perhaps competitive techniques to address classification problems. Brameier and Banzhaf [15] compared the two approaches by conducting experiments on medical datasets. The results showed that LGP could achieve comparative classification

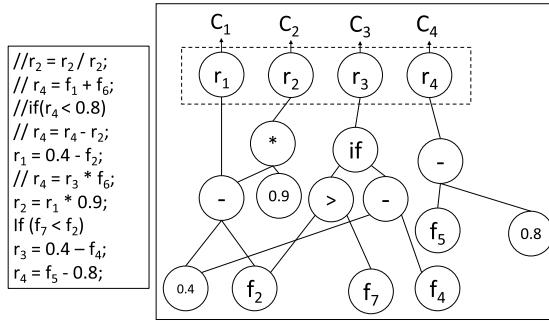


Fig. 7.16 An example of applying LGP to a classification problem having 4 classes and 7 original features (f_1, f_2, \dots, f_7). The four registers r_1, r_2, r_3 and r_4 correspond to the 4 class labels C_1, C_2, C_3 and C_4 . The left sub-figure is the program evolved by LGP. The right sub-figure is the directed acyclic graph representation of the evolved program [30]

performance even though it was not tuned for each dataset unlike ANNs. In terms of computation time, LGP was much more efficient than ANNs. In contrast to ANNs, LGP was shown to build more interpretable models from which the extracted knowledge can provide more insight into the medical diagnosis.

7.4 GP for Classification with Missing Data

In classification, missing data is a common issue where many feature values are unknown due to issues like mechanical failures or respondents' refusal during the data collection process [61]. Most existing classification algorithms require complete data, which makes them unusable for incomplete or missing data. Note, LCSs treat missing values as a "don't care" so can simply match the remaining input to continue classification. Generally, there are two main approaches dealing with missing values. The first approach is to impute missing values according to available values. The imputed values are often considered functions of the available values. After the imputation step, standard classification algorithms can be applied to the obtained (complete) data. The second approach is to build a classification algorithm that can directly deal with missing values. The previous sections show that GP has a great potential in building functions and performing classification, which makes GP an excellent choice to address missing data [104, 106]. The two following subsections show how GP can be used as a data imputation approach and a direct classification algorithm for incomplete data classification, respectively.

7.4.1 GP-Based Imputation

In a classification problem, there is typically a certain kind of relationship between features, and thus it is expected that a feature can be represented as a function of other features. The idea of GP-based imputation is to build functions that can map from complete features to incomplete features. The built functions can be used to fill all the missing values. Given a training set X , the overall steps of imputing the d th feature are as follows.

- Split the training set X into two subsets: X_{dC} contains all the instances that the d th feature has complete values and X_{dM} contains all the instances that the d th feature value is missing.
- Use GP on X_{dC} to build a mapping function from other features to the d th feature. This process is very similar to the process of applying GP for feature construction (or symbolic regression).
- Apply the mapping function to impute the missing values of the d th feature in X_{dM} .
- Form a complete training set X_C by combine the subset X_{dC} and the subset X_{dM} with imputed values.

Once all the missing values are imputed, standard classification algorithms can be trained on the completed data. The results show that GP-based imputation can achieve better classification performance than traditional imputation methods [103]. Since the mapping function of each incomplete feature is built based on the complete data X_{dC} , it is possible that some features in the mapping function may have incomplete values in X_{dM} . In addition, different instances in X_{dM} may have different incomplete features. To address the above situation, multiple mapping functions are built for each incomplete feature. The goal is to provide a wide range of functions among which the most suitable mapping function can be selected for each instance in X_{dM} . The multiple mapping functions can be obtained by running GP multiple times for each feature [103, 105].

7.4.2 GP-Based Classifier

Standard GP cannot be applied to classify the incomplete data since its evolved function requires all the feature values to calculate the function's output. Tran et al. [104] propose Interval GP (IGP) that can handle the missing data directly. The idea is instead of using a specific feature value of each feature, IGP uses the value interval of each feature as its input. The main consideration is that if a feature value is missing, GP can use the feature's interval *directly* estimated from the training set. The use of the feature intervals eliminates the need of imputing missing values that is usually more time-consuming.

Given two features x and y , their corresponding intervals $[x_l, x_u]$ and $[y_l, y_u]$, IGP can combine the two features by using the following operators:

$$\begin{aligned}
x + y &= \begin{cases} lower : x_l + y_l \\ upper : x_u + y_u \end{cases} \\
x - y &= \begin{cases} lower : x_l - y_u \\ upper : x_u - y_l \end{cases} \\
x \times y &= \begin{cases} lower : \min(x_l \times y_l, x_l \times y_u, x_u \times y_l, x_u \times y_u) \\ upper : \max(x_l \times y_l, x_l \times y_u, x_u \times y_l, x_u \times y_u) \end{cases} \\
x/y &= \begin{cases} lower : \min(x_l/y_l, x_l/y_u, x_u/y_l, x_u/y_u) \\ upper : \max(x_l/y_l, x_l/y_u, x_u/y_l, x_u/y_u) \end{cases}
\end{aligned}$$

Since the input of IGP is feature intervals, its output is also an interval $[out_l, out_u]$. The middle point of the interval $mid = (out_l + out_u)/2$ is used as the final output of IGP. The results show that IGP can achieve better classification performance than classifiers that can directly handle missing data such as CART and C4.5 [133].

7.5 GP for Classification with Unbalanced Data

Most classification algorithms assume an even data distribution between different classes. However, such assumption does not hold in many real-world applications which often have unbalanced data. For example, in cancer diagnosis, most patients are non-cancerous (called the *majority* class), while there are very few cancerous patients (called the *minority* class). If all the instances are treated equally, the trained classifier will be bias to the majority class, i.e., instances from the majority class are more likely to be classified correctly than instances from the minority class. However, in many classification problems, correctly classifying the minority class is essential. For instance, wrongly classifying non-cancerous patients (false positives) may lead to a few additional clinical tests, while wrongly classifying cancerous patients (false negatives) may cost their lives. Therefore, building classifiers with high accuracy on all classes is an important task in unbalanced data. This section shows how GP can be used to address the unbalanced data problem.

7.5.1 Fitness Measures

A common fitness measure for GP-based classification algorithms is the overall classification accuracy which can be calculated by Eq. (7.3).

$$Acc = \frac{TP + TN}{TP + FP + TN + FN} \quad (7.3)$$

where the minority class is the *positive* class, TP , TN , FP and FN are the number of true positives, true negatives, false positives and false negatives, respectively.

Clearly, the overall classification accuracy ignores the fact that the positive class has a much smaller number of instances. The classification algorithm will focus more on classifying the negative instances (from the majority class) correctly, which easily yields a higher overall classification accuracy. Thus, Eq. (7.3) considers the classification performance on the minority class and the classification performance on the majority class equally important. To address the issue, we can use *weighted average classification accuracy* which uses a weighting factor w to control the trade-off between the performance of the two classes [86]. The weighted average classification accuracy is shown in Eq. (7.4).

$$AveAcc = w \times \frac{TP}{TP + FN} + (1 - w) \times \frac{TN}{TN + FP} \quad (7.4)$$

where $w \in [0, 1]$. If $w = 0.5$, the two classes are equally important. If $w > 0.5$, the weighted average accuracy focuses more on the minority class. The key question is how to select w , which is typically problem-dependent.

Note that the weighted average accuracy cannot differentiate two different GP classifiers that have the same TP and TN rates. The weighted average accuracy can be further improved by considering the GP output in its fitness measure. Firstly, two target outputs for the majority class and the minority class are defined. The smaller distance between the classifier's outputs and the desired outputs, the better the classifier [7]. The fitness function follows the idea of mean square error and its formula is shown in Eq. (7.5).

$$AveMse = 1 - \frac{1}{2} \left(\frac{\sum_{i=1}^{N_p} (sig(P_{pi}) - T_p)^2}{2 * N_p} + \frac{\sum_{i=1}^{N_n} (sig(P_{ci}) - T_n)^2}{2 * N_n} \right) \quad (7.5)$$

where

$$sig(x) = \frac{2}{1 + e^{-x}} - 1$$

N_p and N_n are the numbers of instances in the positive (minority) class and the negative (majority) class. P_{pi} and P_{ci} are the outputs of a GP classifier being evaluated on the i th instance of the positive class and negative class. T_p and T_n are the two desired outputs of the two classes. It has been shown that *AveMse* achieves significantly better performance than *AveAcc*.

AUC (Area Under the Curve) is another measure that can be used with GP to handle unbalanced data. However, AUC is computationally intensive since it requires to calculate TP and FP many times to form an accurate rendition of the curve. GP can use a direct estimator of the AUC metric [124], which can be seen in Eq. (7.6).

$$AucEst = \frac{\sum_{i=1}^{N_p} \sum_{j=1}^{N_n} I_w(P_i, P_j)}{N_p \times N_n} \quad (7.6)$$

where

$$I_w(P_i, P_h) = \begin{cases} 1, & P_i > P_j \text{ and } P_i \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

P_i and P_j are two output values of the classifier P when the positive instance i and the negative instance j are used as the classifier's inputs. In general, AUC-based fitness measures usually result in a good performance on unbalanced data but also cause the long training times.

Geometric mean (G_Mean) is another measure for unbalanced datasets which can be calculated by Eq. (7.7).

$$G_Mean = \sqrt{\frac{TP}{TP + FN} \times \frac{TN}{TN + FP}} \quad (7.7)$$

The advantage of G_Mean is that it does not require to specify weights to combine the minority accuracy as the majority accuracy as in Eq. (7.4).

7.5.2 Cost-Sensitive Methods

Cost-sensitive learning [135] is another approach to handling unbalanced data. The idea is to treat different misclassification output differently. For example, the cost of predicting a cancerous patient as a non-cancerous patient is much more than the cost of predicting a non-cancerous patient as a cancerous patient. Thus, cost-sensitive learning can enhance the classification performance on the minority class. In cost-sensitive learning, the cost information is presented by a cost matrix as in Table 7.1. C_{00} and C_{11} are the cost of correct predictions, so they are set to 0. C_{01} is the cost of wrongly predicting an instance from the majority class, and it is usually set to 1. C_{10} is the cost of wrongly predicting an instance from the minority class, and it is usually greater than 1, so GP can focus more on avoiding wrong prediction in the minority class. If prior knowledge is not known, C_{10} is usually set to the ratio between the number of instances in the majority class and the number of instances

Table 7.1 A cost matrix for unbalanced data

	Minority class (Class 0)	Majority class (Class 1)
Predicted minority class (Class 0)	$C_{00} = 0$	$C_{01} = 1$
Predicted majority class (Class 1)	$C_{10} > 1$	$C_{11} = 0$

in the minority class. Such a ratio is guaranteed to be greater than 1. GP-based classification algorithms are then trained to minimise the total cost of all the training instances. It has been shown that GP with cost-sensitive learning achieves better performance than GP with the overall classification accuracy [88]. Recently, GP has also been used to learn the cost matrix, which achieves better performance than the manually defined cost matrix [87].

7.5.3 Ensemble Learning

The goal of classification is to achieve good classification performance on both the minority and majority classes. However, Bhowan et al. [8] showed that the minority accuracy and the majority accuracy are usually in conflict, i.e., a classifier with a high minority accuracy usually has a low majority accuracy and vice versa. Thus, building a single classifier might not be sufficient, which requires to build a set or ensemble of classifiers. However, the set of classifiers needs to be diverse. Bhowan et al. [8] proposed to use Multi-objective GP (MOGP) to build the set of classifiers. In MOGP, a classifier P_1 dominates the other classifier P_2 if P_1 is better than P_2 on both classes. The goal of MOGP is to find all the classifiers that are not dominated by any other classifiers, also known as Pareto-optimal classifiers. During its evolutionary process, MOGP maintains an archive set containing all the non-dominated classifiers discovered so far. Once the stopping criterion is met, MOGP returns its archive set as the ensemble of classifiers, which is shown to achieve good performance on the unbalanced dataset. An advantage of MOGP is its ability to build a diverse ensemble of classifiers in a single run.

7.6 Genetic Programming for Explainable/Interpretable Classifiers

Explainability is an essential criterion in building classification models, especially in critical applications such as self-driving cars, laws and healthcare. The focus of explainable classifiers is to improve the transparency and trustworthiness of classification models. Given its symbolic representation, GP has a great potential to build understandable classifiers. In general, GP-based methods can be divided into two main categories: *model-based interpretability* methods which aim to build interpretable models during the training process and *post-hoc interpretability* methods which aim to build a more interpretable model to approximate a trained model. A recent comprehensive survey on GP for learning interpretable models can be seen from [67].

7.6.1 Model-Based GP for Classification

Although GP offers potential explainability, most existing GP-based classifiers are still large and complex due to the depth and breadth of the learnt GP-tree, which make them less interpretable. One way to improve the interpretability of GP is to reduce its tree size. An example is to dynamically control the tree depth limit [95] based on the best tree in the population. During the evolutionary process, any offspring, which has a larger depth than the limit, is removed. The only exception is when the offspring is better than the current best tree. The tree size can also be controlled by adding penalty terms to the GP's fitness function that penalises individuals with large trees [26, 43, 127]. Another alternative is to develop genetic operators that generate small offspring. For example, crossover is allowed to swap sub-trees with similar sizes only, which prevents crossover from generating offspring that is much larger than its parents [53, 90]. Similarly, mutation is allowed to replace a sub-tree with another sub-tree that is not much larger than the replaced one [54]. The model size can also be embedded into the selection process where the small trees are given higher priority. For instance, Wang et al. [117] maintain an archive set of small and high-performing trees. Selecting parents from the archive set will generate small offspring. Zhang and Wong [132] proposed to simplify the GP programs during the evolutionary process, which is known as online program simplification. The idea is to reduce a program's size by removing the redundancy in the program, for example, a sub-tree ($a\text{-}0$) is replaced by a single node a .

Besides model size, there are several other complexity measures for GP such as *structural complexity* and *behavioural/functional complexity*. Most structural complexity measures consider the shape of a tree. Given two trees with the same number of nodes, a balance tree with fewer nested nodes is considered simpler than a skew and deep tree [47, 68]. On the other hand, the behavioural/functional complexity focuses on the complexity of the evolved functions/programs, especially the non-linear functions that are much less interpretable. An example is the degree of the polynomial functions [113]. A more complex one is the VC dimension which illustrates the ability to learn from any given data [17, 18].

Another way to improve the explainability of GP-based classifier is to reduce the number of features, which can be achieved by *feature selection* and *feature construction*. GP has a built-in feature selection ability where the features, which appear frequently in good tree, are considered good features and should be selected. Tran et al. [105] show that GP can select only 1% of the original features and still improve the classification performance over using all features. Given its flexible representation, GP has also been widely applied to achieve feature construction. The main goal of GP-based feature construction methods is to build informative high-level features from low-level features. It has been shown that GP can significantly improve the classification performance over the original features [105]. Depending on the difficulty of a learning problem, GP can build a single high-level feature [73, 112] or multiple high-level features [1, 101]. An advantage of GP-based feature

construction is its explicit high-level features, which can *explain* how the original features interact to achieve high learning performance [57].

7.6.2 Post-hoc Interpretability GP for Classification

Post-hoc interpretability GP approximates a (complex) pre-trained classifier by a GP-based classifier that is more explainable than the original classifier. Firstly, the complex classifier is used to generate a dataset where the output is the prediction of the classifier. An interpretable GP-based classifier is then trained on the generated dataset. For example, Evans et al. [28] approximate a 200-layer neural network model or an ensemble of 500 decision trees by a 4-layer decision tree, which is much more interpretable while still maintaining the learning performance. However, in comparison with the model-based methods, there is still very limited work on post-hoc interpretability GP.

7.7 Advanced Topics

7.7.1 Evolutionary Transfer Learning for Classification

Transfer learning aims to address a target classification problem by applying knowledge learned from other related classification problems [84]. Since EC is a population-based family and does not require any assumption of the objective function types, EC has been widely applied to achieve transfer learning, also known as evolutionary transfer learning. In general, evolutionary transfer learning can be divided into three main categories: *evolutionary sequential transfer learning*, *evolutionary multi-tasking* and *evolutionary multiform learning/optimisation* [37].

Evolutionary sequential transfer learning tackles the target classification problem L_k when all other related classification (source) problems L_1, L_2, \dots, L_{k-1} have been addressed and the knowledge from the source problems has been extracted. For example, Nguyen et al. [74, 76, 79] apply EC to build a latent feature space on which the projected source data and projected target data follow the same distribution, and thus the projected source data can improve the classification performance on the target classification task.

In contrast to sequential transfer learning, evolutionary multi-tasking addresses all the classification problems L_1, L_2, \dots, L_k simultaneously. Each task is typically addressed by a sub-population. During the training process, the sub-populations continuously share their knowledge. A well-known framework of evolutionary multi-tasking is MFEA (Multi-Factorial Evolutionary Algorithm) [36], in which the assortative mating allows parents of different classification problems to exchange their knowledge, but with low probability to avoid negative transfer learning.

While the two former ones deal with multiple different classification problems, evolutionary multiform learning/optimisation deal with a single classification problem, which is viewed under different objective formulations. Each objective formulation forms one learning task and MFEA can be applied to address the set of obtained learning tasks. For example, Jiao et al. [41] view multi-objective feature selection (for classification) as a multiform learning problem where different objective formulations use different weight vectors to combine the classification performance and the number of selected features. The proposed algorithm can evolve better feature subsets than other evolutionary multi-objective algorithms.

7.7.2 Evolutionary Surrogate Models

To address a problem, EC usually requires a large number of evaluations to obtain sufficiently good solutions. However, in many real-world problems, the evaluation can be computationally intensive. For example, each evaluation in designing vehicle shape can take several hours [111]. Surrogate models are proposed to reduce the computation cost of EC on computationally intensive problems, which is particularly useful for intensive classification on large datasets. The key idea of a surrogate model is to simulate the behaviour of the true fitness function while requiring much less computational resource. Thus, using the surrogate model to evaluate the candidate solutions can significantly reduce the computation cost. In general, designing a surrogate model has three main steps:

- *Model construction*: This step builds a surrogate model based on a set of candidate solutions and their fitness values. The solution set can be obtained after a number of generations or by applying some sample strategies. Usually, building a surrogate model is a classification or regression problem, and the solution set is considered a (surrogate) training set. The classification-based model aims to rank the candidate solutions, which is known as a *relative fitness model* [78]. In contrast, the regression-based model aims to predict the fitness values of the candidate solutions, which is known as an *absolute model* [100].
- *Interaction*: This step regards how to use the surrogate model during the evolutionary process. In general, there are two main ways: *pre-selection* and *estimation*. The *pre-selection* way generates a large number of candidate solutions which are evaluated by the surrogate model [98]. From the large set of solutions, a small number of promising solutions are selected, which are then re-evaluated by the true fitness function. The *estimation* way directly evaluates offspring by the surrogate model.
- *Re-evaluation*: This step regards when to re-evaluate candidate solutions using the true fitness function, which avoids the convergence to a false optima. The two most common strategies are: individual-based and generation-based. The individual-based strategy re-evaluates a part of the population in every generation [62, 96]. In contrast, the generation-based strategy re-evaluates the whole population in some generations [6, 82].

Evolutionary surrogate models have been successfully applied to many expensive real-world classification problems such as health system [115].

7.7.3 *Evolutionary Deep Structures*

Deep neural networks (DNNs) have had great success in many real-world classification applications. However, the performance of DNNs relies heavily on their network architectures which usually requires rich expertise. Recently, EC has been widely applied to automatically search for the optimal DNNs architecture. The main reason is that EC algorithms can deal with various challenges of neural architecture search such as complex constraints, discrete search space and multiple conflicting criteria [63]. In general, each individual of an evolutionary population represents a network architecture. The connection weights of the architecture are optimised by a gradient-based optimisation technique. The performance of the obtained network (constructed by the combination of the architecture and the connection weights) is used as the individual's fitness. However, the architecture of DNNs is usually large and complex which requires an effective representation for EC. An effective representation should consider three aspects: the hyperparameters of each layer, the depth of the architecture and the connections between layers [91, 121]. A typical encoding scheme uses a set of integer numbers to denote all possible options of the above aspects. These integer numbers are converted to binary values, and the task is to optimise a vector of binary values. Therefore, vector-based evolutionary algorithms have been widely applied to neural architecture design [16, 42, 69, 93, 114]. The results show that the evolved architectures are not only effective but also much simpler than existing manual designs [63]. More discussions on evolutionary neural architecture search can be seen from later chapters of this handbook.

7.8 Conclusions

Given the potential global search ability, EC has been widely applied to many classification problems including challenging characteristics such as high-dimensional data, irrelevant/redundant data, missing data and unbalanced data. Evolutionary classification applications range from optimising parameters for standard classification algorithms to performing classification directly. Evolutionary classification can also improve the quality of the training data by various preprocessing steps such as feature selection, feature construction and instance selection, which can significantly increase the classification performance. Furthermore, EC builds interpretable classifiers such as classification rules/trees/graphs by GP and LCSs, which is particularly important in eXplainable AI.

Recently, in evolutionary classification, several emerging directions have been investigated such as evolutionary cost-sensitive learning, evolutionary transfer learning, evolutionary surrogate models and evolutionary deep structures. We expect to see more work to consolidate these directions. In addition, the lack of theoretical analysis and intensive computation cost are major concerns in evolutionary classification, which need to be addressed in the future.

References

1. Ain, Q.U., Al-Sahaf, H., Xue, B., Zhang, M.: Generating knowledge-guided discriminative features using genetic programming for melanoma detection. *IEEE Trans. Emerg. Top. Comput. Intell.* **5**(4), 554–569 (2020)
2. Ain, Q.U., Xue, B., Al-Sahaf, H., Zhang, M.: Genetic programming for skin cancer detection in dermoscopic images. In: 2017 IEEE Congress on Evolutionary Computation (CEC), pp. 2420–2427. IEEE (2017)
3. Al-Sahaf, H., Song, A., Neshatian, K., Zhang, M.: Two-tier genetic programming: towards raw pixel-based image classification. *Expert Syst. Appl.* **39**(16), 12291–12301 (2012)
4. Albuquerque, I.M.R., Nguyen, B.H., Xue, B., Zhang, M.: A novel genetic algorithm approach to simultaneous feature selection and instance selection. In: IEEE Symposium Series on Computational Intelligence (SSCI), pp. 616–623 (2020)
5. AlSukker, A., Khushaba, R., Al-Ani, A.: Optimizing the k-nn metric weights using differential evolution. In: International Conference on Multimedia Computing and Information Technology (MCIT), pp. 89–92. IEEE (2010)
6. Bajer, L., Pitra, Z., Repický, J., Holeňa, M.: Gaussian process surrogate models for the cma evolution strategy. *Evol. Comput.* **27**(4), 665–697 (2019)
7. Bhowan, U., Johnston, M., Zhang, M.: Developing new fitness functions in genetic programming for classification with unbalanced data. *IEEE Trans. Syst. Man, Cybernet. Part B (Cybernetics)* **42**(2), 406–421 (2011)
8. Bhowan, U., Johnston, M., Zhang, M.: Ensemble learning and pruning in multi-objective genetic programming for classification with unbalanced data. In: Australasian Joint Conference on Artificial Intelligence, pp. 192–202. Springer (2011)
9. Bi, Y., Xue, B., Zhang, M.: An automatic feature extraction approach to image classification using genetic programming. In: International Conference on the Applications of Evolutionary Computation, pp. 421–438. Springer (2018)
10. Bi, Y., Xue, B., Zhang, M.: An automated ensemble learning framework using genetic programming for image classification. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 365–373 (2019)
11. Bi, Y., Xue, B., Zhang, M.: Genetic programming with image-related operators and a flexible program structure for feature learning in image classification. *IEEE Trans. Evol. Comput.* **25**(1), 87–101 (2020)
12. Bi, Y., Xue, B., Zhang, M.: Genetic programming with a new representation to automatically learn features and evolve ensembles for image classification. *IEEE Trans. Cybernet.* **51**(4), 1769–1783 (2021)
13. Biswas, N., Chakraborty, S., Mullick, S.S., Das, S.: A parameter independent fuzzy weighted k-nearest neighbor classifier. *Pattern Recogn. Lett.* **101**, 80–87 (2018)
14. Bot, M.C., Langdon, W.B.: Application of genetic programming to induction of linear classification trees. In: European Conference on Genetic Programming, pp. 247–258. Springer (2000)
15. Brameier, M., Banzhaf, W.: Linear genetic programming, vol. 1. Springer (2007)

16. Byla, E., Pang, W.: Deepswarm: Optimising convolutional neural networks using swarm intelligence. In: UK Workshop on Computational Intelligence, pp. 119–130. Springer (2019)
17. Chen, Q., Xue, B., Shang, L., Zhang, M.: Improving generalisation of genetic programming for symbolic regression with structural risk minimisation. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 709–716 (2016)
18. Chen, Q., Xue, B., Zhang, M.: Improving generalization of genetic programming for symbolic regression with angle-driven geometric semantic operators. *IEEE Trans. Evol. Comput.* **23**(3), 488–502 (2018)
19. Cheng, F., Chu, F., Zhang, L.: A multi-objective evolutionary algorithm based on length reduction for large-scale instance selection. *Inf. Sci.* **576**, 105–121 (2021)
20. Cover, T., Hart, P.: Nearest neighbor pattern classification. *IEEE Trans. Inf. Theory* **13**(1), 21–27 (1967)
21. Cui, X., Zhang, W., Tüske, Z., Picheny, M.: Evolutionary stochastic gradient descent for optimization of deep neural networks. *Adv. Neural Inf. Process. Syst.* **31** (2018)
22. Dada, E.G., Bassi, J.S., Chiroma, H., Abdulhamid, S.M., Adetunmbi, A.O., Ajibawa, O.E.: Machine learning for email spam filtering: review, approaches and open research problems. *Heliyon* **5**(6), e01802 (2019)
23. Demir, K., Nguyen, B.H., Xue, B., Zhang, M.: Particle swarm optimisation for sparsity-based feature selection in multi-label classification. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, pp. 232–235 (2022)
24. Derrac, J., Chiclana, F., García, S., Herrera, F.: Evolutionary fuzzy k-nearest neighbors algorithm using interval-valued fuzzy sets. *Inf. Sci.* **329**, 144–163 (2016)
25. Downey, C., Zhang, M., Liu, J.: Parallel linear genetic programming for multi-class classification. *Genet. Program Evolvable Mach.* **13**(3), 275–304 (2012)
26. Ekart, A., Nemeth, S.Z.: Selection based on the pareto nondomination criterion for controlling code growth in genetic programming. *Genet. Program Evolvable Mach.* **2**(1), 61–73 (2001)
27. Espejo, P.G., Romero, C., Ventura, S., Hervás, C.: Induction of classification rules with grammar-based genetic programming. In: Conference on Machine Intelligence, pp. 596–601 (2005)
28. Evans, B.P., Xue, B., Zhang, M.: What's inside the black-box? a genetic programming method for interpreting complex machine learning models. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1012–1020 (2019)
29. Fan, Q., Bi, Y., Xue, B., Zhang, M.: Genetic programming for image classification: A new program representation with flexible feature reuse. *IEEE Trans. Evolut. Comput.* 1–1 (2022). <https://doi.org/10.1109/TEVC.2022.3169490>
30. Fogelberg, C., Zhang, M.: Linear genetic programming for multi-class object classification. In: Australasian Joint Conference on Artificial Intelligence, pp. 369–379. Springer (2005)
31. Friedrichs, F., Igel, C.: Evolutionary tuning of multiple SVM parameters. *Neurocomputing* **64**, 107–117 (2005)
32. Giri, R., Chowdhury, A., Ghosh, A., Das, S., Abraham, A., Snasel, V.: A modified invasive weed optimization algorithm for training of feed-forward neural networks. In: IEEE International Conference on Systems, Man and Cybernetics, pp. 3166–3173 (2010)
33. Gomes, T.A., Prudêncio, R.B., Soares, C., Rossi, A.L., Carvalho, A.: Combining meta-learning and search techniques to select parameters for support vector machines. *Neurocomputing* **75**(1), 3–13 (2012)
34. Gong, M., Liu, J., Li, H., Cai, Q., Su, L.: A multiobjective sparse feature learning model for deep neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* **26**(12), 3263–3277 (2015)
35. Gudise, V., Venayagamoorthy, G.: Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks. In: IEEE Swarm Intelligence Symposium, pp. 110–117 (2003)
36. Gupta, A., Ong, Y.S., Feng, L.: Multifactorial evolution: toward evolutionary multitasking. *IEEE Trans. Evol. Comput.* **20**(3), 343–357 (2015)
37. Gupta, A., Ong, Y.S., Feng, L.: Insights on transfer optimization: because experience is the best teacher. *IEEE Trans. Emerg. Top. Comput. Intell.* **2**(1), 51–64 (2017)

38. Hall, M.A.: Correlation-based feature selection for discrete and numeric class machine learning. In: Proceedings of the Seventeenth International Conference on Machine Learning, p. p. 359–366 (2000)
39. Han, J., Pei, J., Kamber, M.: Data Mining: Concepts and Techniques. Elsevier (2011)
40. Huang, J., Hu, X., Yang, F.: Support vector machine with genetic algorithm for machinery fault diagnosis of high voltage circuit breaker. *Measurement* **44**(6), 1018–1027 (2011)
41. Jiao, R., Xue, B., Zhang, M.: Benefiting from single-objective feature selection to multiobjective feature selection: a multiform approach. *IEEE Trans. Cybernet.* 1–14 (2022). <https://doi.org/10.1109/TCYB.2022.3218345>
42. Junior, F.E.F., Yen, G.G.: Particle swarm optimization of deep neural networks architectures for image classification. *Swarm Evol. Comput.* **49**, 62–74 (2019)
43. Kalganova, T., Miller, J.: Evolving more efficient digital circuits by allowing circuit layout evolution and multi-objective fitness. In: Proceedings of the first NASA/DoD workshop on evolvable hardware, pp. 54–63. IEEE (1999)
44. Kalita, D.J., Singh, S.: SVM hyper-parameters optimization using quantized multi-PSO in dynamic environment. *Soft. Comput.* **24**(2), 1225–1241 (2020)
45. Karaboga, D., Akay, B., Ozturk, C.: Artificial bee colony (abc) optimization algorithm for training feed-forward neural networks. In: International Conference on Modeling Decisions for Artificial Intelligence, pp. 318–329. Springer (2007)
46. Kearney, J.K., Thompson, W.B., Boley, D.L.: Optical flow estimation: an error analysis of gradient-based methods with local optimization. *IEEE Trans. Pattern Anal. Mach. Intell.* **2**, 229–244 (1987)
47. Keijzer, M., Foster, J.: Crossover bias in genetic programming. In: European Conference on Genetic Programming, pp. 33–44. Springer (2007)
48. Khaleel, M.I., Hmeidi, I.I., Najadat, H.M.: An automatic text classification system based on genetic algorithm. In: Proceedings of the The 3rd Multidisciplinary International Social Networks Conference on Social Informatics 2016, Data Science 2016, MISNC, SI, DS 2016. Association for Computing Machinery, New York, NY, USA (2016)
49. Khoshgoftaar, T.M., Liu, Y.: A multi-objective software quality classification model using genetic programming. *IEEE Trans. Reliab.* **56**(2), 237–245 (2007)
50. Khoshgoftaar, T.M., Seliya, N., Liu, Y.: Genetic programming-based decision trees for software quality classification. In: IEEE International Conference on Tools with Artificial Intelligence, pp. 374–383 (2003)
51. Kishore, J.K., Patnaik, L.M., Mani, V., Agrawal, V.: Application of genetic programming for multicategory pattern classification. *IEEE Trans. Evol. Comput.* **4**(3), 242–258 (2000)
52. Kuo, C.S., Hong, T.P., Chen, C.L.: Applying genetic programming technique in classification trees. *Soft. Comput.* **11**(12), 1165–1172 (2007)
53. Langdon, W.B.: Size fair and homologous tree genetic programming crossovers. *Genet. Program Evolvable Mach.* **1**(1/2), 95–119 (2000)
54. Langdon, W.B., Soule, T., Poli, R., Foster, J.A.: The evolution of size and shape. *Adv. Genetic Program.* **3**, 163–190 (1999)
55. Le, H.L., Neri, F., Landa-Silva, D., Triguero, I.: Accelerated pattern search with variable solution size for simultaneous instance selection and generation. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, pp. 256–259 (2022)
56. Le, T.T., Fu, W., Moore, J.H.: Scaling tree-based automated machine learning to biomedical big data with a feature set selector. *Bioinformatics* **36**(1), 250–256 (2020)
57. Lensen, A.: Mining feature relationships in data. In: European Conference on Genetic Programming (Part of EvoStar), pp. 247–262. Springer (2021)
58. Lensen, A., Al-Sahaf, H., Zhang, M., Xue, B.: Genetic programming for region detection, feature extraction, feature construction and classification in image data. In: European Conference on Genetic Programming, pp. 51–67. Springer (2016)
59. Li, J., Liu, H.: Challenges of feature selection for big data analytics. *IEEE Intell. Syst.* **32**(2), 9–15 (2017)

60. Li, Y.M., Wang, M., Cui, L.J., Huang, D.M.: A new classification arithmetic for multi-image classification in genetic programming. In: IEEE International Conference on Machine Learning and Cybernetics, vol. 3, pp. 1683–1687 (2007)
61. Little, R.J., Rubin, D.B.: Statistical Analysis with Missing Data, vol. 793. Wiley (2019)
62. Liu, B., Zhang, Q., Gielen, G.G.: A gaussian process surrogate model assisted evolutionary algorithm for medium scale expensive optimization problems. *IEEE Trans. Evol. Comput.* **18**(2), 180–192 (2013)
63. Liu, Y., Sun, Y., Xue, B., Zhang, M., Yen, G.G., Tan, K.C.: A survey on evolutionary neural architecture search. *IEEE Trans. Neural Netw. Learn. Syst.* 1–21 (2021). <https://doi.org/10.1109/TNNLS.2021.3100554>
64. Lorena, A.C., De Carvalho, A.C.: Evolutionary tuning of SVM parameter values in multiclass problems. *Neurocomputing* **71**(16–18), 3326–3334 (2008)
65. Loveard, T., Ciesielski, V.: Representing classification problems in genetic programming. In: IEEE Congress on Evolutionary Computation, vol. 2, pp. 1070–1077 vol. 2 (2001)
66. Ma, W., Zhou, X., Zhu, H., Li, L., Jiao, L.: A two-stage hybrid ant colony optimization for high-dimensional feature selection. *Pattern Recogn.* **116**, 107933 (2021)
67. Mei, Y., Chen, Q., Lenssen, A., Xue, B., Zhang, M.: Explainable artificial intelligence by genetic programming: a survey. *IEEE Trans. Evolut. Comput.* 1–1 (2022). <https://doi.org/10.1109/TEVC.2022.3225509>
68. Mei, Y., Zhang, M., Nyugen, S.: Feature selection in evolving job shop dispatching rules with genetic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 365–372 (2016)
69. Mitschke, N., Heizmann, M., Noffz, K.H., Wittmann, R.: Gradient based evolution to optimize the structure of convolutional neural networks. In: IEEE International Conference on Image Processing (ICIP), pp. 3438–3442 (2018). <https://doi.org/10.1109/ICIP.2018.8451394>
70. Montana, D.J.: Strongly typed genetic programming. *Evol. Comput.* **3**(2), 199–230 (1995)
71. Montana, D.J., Davis, L.: Training feedforward neural networks using genetic algorithms. In: Proceedings of the 11th International Joint Conference on Artificial Intelligence - Volume 1, p. 762–767. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1989)
72. Muni, D., Pal, N., Das, J.: A novel approach to design classifiers using genetic programming. *IEEE Trans. Evol. Comput.* **8**(2), 183–196 (2004)
73. Nag, K., Pal, N.R.: Feature extraction and selection for parsimonious classifiers with multi-objective genetic programming. *IEEE Trans. Evol. Comput.* **24**(3), 454–466 (2019)
74. Nguyen, B.H., Xue, B., Andreea, P.: A particle swarm optimization based feature selection approach to transfer learning in classification. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 37–44 (2018)
75. Nguyen, B.H., Xue, B., Andreea, P., Ishibuchi, H., Zhang, M.: Multiple reference points-based decomposition for multiobjective feature selection in classification: Static and dynamic mechanisms. *IEEE Trans. Evol. Comput.* **24**(1), 170–184 (2020)
76. Nguyen, B.H., Xue, B., Andreea, P., Zhang, M.: A hybrid evolutionary computation approach to inducing transfer classifiers for domain adaptation. *IEEE Trans. Cybernet.* **51**(12), 6319–6332 (2021). <https://doi.org/10.1109/TCYB.2020.2980815>
77. Nguyen, B.H., Xue, B., Zhang, M.: A survey on swarm intelligence approaches to feature selection in data mining. *Swarm Evol. Comput.* **54**, 100663 (2020)
78. Nguyen, B.H., Xue, B., Zhang, M.: A constrained competitive swarm optimiser with an SVM-based surrogate model for feature selection. *IEEE Trans. Evolut. Comput.* 1–1 (2022). <https://doi.org/10.1109/TEVC.2022.3197427>
79. Nguyen, B.H., Xue, B., Zhang, M., Andreea, P.: Population-based ensemble classifier induction for domain adaptation. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 437–445 (2019)
80. Nguyen, H.B., Xue, B., Andreea, P.: Mutual information for feature selection: estimation or counting? *Evol. Intel.* **9**(3), 95–110 (2016)
81. Nguyen, H.B., Xue, B., Andreea, P.: A hybrid GA-GP method for feature reduction in classification. In: Asia-Pacific Conference on Simulated Evolution and Learning, pp. 591–604. Springer (2017)

82. Nguyen, H.B., Xue, B., Andreae, P.: PSO with surrogate models for feature selection: static and dynamic clustering-based methods. *Memetic Comput.* **10**(3), 291–300 (2018)
83. Olvera-López, J.A., Carrasco-Ochoa, J.A., Martínez-Trinidad, J., Kittler, J.: A review of instance selection methods. *Artif. Intell. Rev.* **34**(2), 133–143 (2010)
84. Pan, S.J., Yang, Q.: A survey on transfer learning. *IEEE Trans. Knowl. Data Eng.* **22**(10), 1345–1359 (2009)
85. Paniri, M., Dowlatshahi, M.B., Nezamabadi-pour, H.: Ant-td: ant colony optimization plus temporal difference reinforcement learning for multi-label feature selection. *Swarm Evol. Comput.* **64**, 100892 (2021)
86. Patterson, G., Zhang, M.: Fitness functions in genetic programming for classification with unbalanced data. In: Australasian Joint Conference on Artificial Intelligence, pp. 769–775. Springer (2007)
87. Pei, W., Xue, B., Shang, L., Zhang, M.: Developing interval-based cost-sensitive classifiers by genetic programming for binary high-dimensional unbalanced classification [research frontier]. *IEEE Comput. Intell. Mag.* **16**(1), 84–98 (2021)
88. Pei, W., Xue, B., Zhang, M., Shang, L.: A cost-sensitive genetic programming approach for high-dimensional unbalanced classification. In: IEEE Symposium Series on Computational Intelligence (SSCI), pp. 1770–1777 (2019)
89. Pena-Reyes, C.A., Sipper, M.: Evolutionary computation in medicine: an overview. *Artif. Intell. Med.* **19**(1), 1–23 (2000)
90. Platel, M.D., Clergue, M., Collard, P.: Maximum homologous crossover for linear genetic programming. In: European Conference on Genetic Programming, pp. 194–203. Springer (2003)
91. Real, E., Aggarwal, A., Huang, Y., Le, Q.V.: Regularized evolution for image classifier architecture search. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 33, pp. 4780–4789 (2019)
92. Sakprasat, S., Sinclair, M.C.: Classification rule mining for automatic credit approval using genetic programming. In: IEEE Congress on Evolutionary Computation, pp. 548–555 (2007)
93. Sapra, D., Pimentel, A.D.: An evolutionary optimization algorithm for gradually saturating objective functions. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 886–893 (2020)
94. Shrivastava, N.A., Khosravi, A., Panigrahi, B.K.: Prediction interval estimation of electricity prices using PSO-tuned support vector machines. *IEEE Trans. Industr. Inf.* **11**(2), 322–331 (2015). <https://doi.org/10.1109/TII.2015.2389625>
95. Silva, S., Costa, E.: Dynamic limits for bloat control in genetic programming and a review of past and current bloat theories. *Genet. Program Evolvable Mach.* **10**(2), 141–179 (2009)
96. Sun, C., Jin, Y., Zeng, J., Yu, Y.: A two-layer surrogate-assisted particle swarm optimization algorithm. *Soft. Comput.* **19**(6), 1461–1475 (2015)
97. Sun, Y., Yen, G.G., Yi, Z.: Evolving unsupervised deep neural networks for learning meaningful representations. *IEEE Trans. Evol. Comput.* **23**(1), 89–103 (2019)
98. Tackett, W.A.: Recombination, selection, and the genetic construction of computer programs. Ph.D. thesis, University of Southern California, Department of Electrical Engineering Systems, USA (1994)
99. Tanigawa, T., Zhao, Q.: A study on efficient generation of decision trees using genetic programming. In: Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation, GECCO'00, p. 1047-1052. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2000)
100. Tong, H., Huang, C., Minku, L.L., Yao, X.: Surrogate models in evolutionary single-objective optimization: A new taxonomy and experimental study. *Inf. Sci.* **562**, 414–437 (2021)
101. Tran, B., Xue, B., Zhang, M.: Class dependent multiple feature construction using genetic programming for high-dimensional data. In: Australasian Joint Conference on Artificial Intelligence, pp. 182–194. Springer (2017)
102. Tran, B., Xue, B., Zhang, M.: Genetic programming for multiple-feature construction on high-dimensional classification. *Pattern Recogn.* **93**, 404–417 (2019)

103. Tran, C.T., Zhang, M., Andreae, P.: Multiple imputation for missing data using genetic programming. In: Proceedings of the Annual Conference on Genetic and Evolutionary Computation, pp. 583–590 (2015)
104. Tran, C.T., Zhang, M., Andreae, P.: Directly evolving classifiers for missing data using genetic programming. In: IEEE Congress on Evolutionary Computation (CEC), pp. 5278–5285 (2016)
105. Tran, C.T., Zhang, M., Andreae, P.: A genetic programming-based imputation method for classification with missing data. In: European Conference on Genetic Programming, pp. 149–163. Springer (2016)
106. Tran, C.T., Zhang, M., Andreae, P., Xue, B.: Directly constructing multiple features for classification with missing data using genetic programming with interval functions. In: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion, pp. 69–70 (2016)
107. Tsakonas, A., Dounias, G., Jantzen, J., Axer, H., Bjerregaard, B., von Keyserlingk, D.G.: Evolving rule-based systems in two medical domains using genetic programming. *Artif. Intell. Med.* **32**(3), 195–216 (2004)
108. Urbanowicz, R.J., Browne, W.N.: Introduction to Learning Classifier Systems. Springer (2017)
109. Urbanowicz, R.J., Granizo-Mackenzie, A., Moore, J.H.: An analysis pipeline with statistical and visualization-guided knowledge discovery for Michigan-style learning classifier systems. *IEEE Comput. Intell. Mag.* **7**(4), 35–45 (2012)
110. Urbanowicz, R.J., Meeker, M., La Cava, W., Olson, R.S., Moore, J.H.: Relief-based feature selection: Introduction and review. *J. Biomed. Inform.* **85**, 189–203 (2018)
111. Urquhart, M., Ljungskog, E., Sebben, S.: Surrogate-based optimisation using adaptively scaled radial basis functions. *Appl. Soft Comput.* **88**, 106050 (2020)
112. Virgolin, M., Alderliesten, T., Bosman, P.A.: On explaining machine learning models by evolving crucial and compact features. *Swarm Evol. Comput.* **53**, 100640 (2020)
113. Vladislavleva, E.J., Smits, G.F., Den Hertog, D.: Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming. *IEEE Trans. Evol. Comput.* **13**(2), 333–349 (2008)
114. Wang, B., Sun, Y., Xue, B., Zhang, M.: A hybrid differential evolution approach to designing deep convolutional neural networks for image classification. In: Australasian Joint Conference on Artificial Intelligence, pp. 237–250. Springer (2018)
115. Wang, H., Jin, Y., Jansen, J.O.: Data-driven surrogate-assisted multiobjective evolutionary optimization of a trauma system. *IEEE Trans. Evol. Comput.* **20**(6), 939–952 (2016)
116. Wang, P., Xue, B., Liang, J., Zhang, M.: Differential evolution based feature selection: A niching-based multi-objective approach. *IEEE Trans. Evolut. Comput.* 1–1 (2022). <https://doi.org/10.1109/TEVC.2022.3168052>
117. Wang, S., Mei, Y., Zhang, M., Yao, X.: Genetic programming with niching for uncertain capacitated arc routing problem. *IEEE Trans. Evol. Comput.* **26**(1), 73–87 (2021)
118. Winkler, S., Affenzeller, M., Wagner, S.: Advanced genetic programming based machine learning. *J. Math. Model. Algorithms* **6**(3), 455–480 (2007)
119. Wongsee, W., Chaiyaratana, N., Vichittumaros, K., Winichagool, P., Fucharoen, S.: Thalassaemia classification by neural networks and genetic programming. *Inf. Sci.* **177**(3), 771–786 (2007)
120. Wood, J., Nguyen, B.H., Xue, B., Zhang, M., Killeen, D.: Automated fish classification using unprocessed fatty acid chromatographic data: a machine learning approach. In: Proceedings of the Australasian Joint Conference on Artificial Intelligence, pp. 516–529. Springer (2022)
121. Xie, L., Yuille, A.: Genetic cnn. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 1379–1388 (2017)
122. Xue, B., Zhang, M., Browne, W.N., Yao, X.: A survey on evolutionary computation approaches to feature selection. *IEEE Trans. Evol. Comput.* **20**(4), 606–626 (2015)
123. Xue, Y., Zhu, H., Liang, J., Słowik, A.: Adaptive crossover operator based multi-objective binary genetic algorithm for feature selection in classification. *Knowl.-Based Syst.* **227**, 107218 (2021)

124. Yan, L., Dodier, R.H., Mozer, M., Wolniewicz, R.H.: Optimizing classifier performance via an approximation to the wilcoxon-mann-whitney statistic. In: Proceedings of the International Conference on Machine Learning, pp. 848–855 (2003)
125. Yuan, X., Liu, Z., Miao, Z., Zhao, Z., Zhou, F., Song, Y.: Fault diagnosis of analog circuits based on IH-PSO optimized support vector machine. *IEEE Access* **7**, 137945–137958 (2019). <https://doi.org/10.1109/ACCESS.2019.2943071>
126. Zeng, N., Qiu, H., Wang, Z., Liu, W., Zhang, H., Li, Y.: A new switching-delayed-PSO-based optimized SVM algorithm for diagnosis of alzheimer's disease. *Neurocomputing* **320**, 195–202 (2018)
127. Zhang, B.T., Mühlenbein, H.: Balancing accuracy and parsimony in genetic programming. *Evol. Comput.* **3**(1), 17–38 (1995)
128. Zhang, M., Ciesielski, V.: Genetic programming for multiple class object detection. In: Advanced Topics in Artificial Intelligence: 12th Australian Joint Conference on Artificial Intelligence, AI'99 Sydney, Australia, December 6–10, 1999 Proceedings 12, pp. 180–192. Springer (1999)
129. Zhang, M., Gao, X., Lou, W.: A new crossover operator in genetic programming for object classification. *IEEE Trans. Syst. Man Cybernet. Part B (Cybernet.)* **37**(5), 1332–1343 (2007)
130. Zhang, M., Smart, W.: Multiclass object classification using genetic programming. In: Workshops on Applications of Evolutionary Computation, pp. 369–378. Springer (2004)
131. Zhang, M., Smart, W.: Using gaussian distribution to construct fitness functions in genetic programming for multiclass object classification. *Pattern Recogn. Lett.* **27**(11), 1266–1274 (2006)
132. Zhang, M., Wong, P.: Genetic programming for medical classification: a program simplification approach. *Genet. Program Evolvable Mach.* **9**(3), 229–255 (2008)
133. Zhang, S., Qin, Z., Ling, C., Sheng, S.: “missing is useful”: missing values in cost-sensitive decision trees. *IEEE Trans. Knowl. Data Eng.* **17**(12), 1689–1693 (2005)
134. Zhong, W., Zhuang, Y., Sun, J., Gu, J.: A load prediction model for cloud computing using PSO-based weighted wavelet support vector machine. *Appl. Intell.* **48**(11), 4072–4083 (2018)
135. Zhou, Z.H.: Cost-sensitive learning. In: International Conference on Modeling Decisions for Artificial Intelligence, pp. 17–18. Springer (2011)

Chapter 8

Evolutionary Ensemble Learning



Malcolm I. Heywood

Abstract Evolutionary Ensemble Learning (EEL) provides a general approach for scaling evolutionary learning algorithms to increasingly complex tasks. This is generally achieved by developing a diverse complement of models that provide solutions to different (yet overlapping) aspects of the task. This chapter reviews the topic of EEL by considering two basic application contexts that were initially developed independently: (1) ensembles as applied to classification and regression problems and (2) multi-agent systems as typically applied to reinforcement learning tasks. We show that common research themes have developed from the two communities, resulting in outcomes applicable to both application contexts. More recent developments reviewed include EEL frameworks that support variable-sized ensembles, scaling to high cardinality or dimensionality, and operation under dynamic environments. Looking to the future we point out that the versatility of EEL can lead to developments that support interpretable solutions and lifelong/continuous learning.

8.1 Introduction

Evolutionary Ensemble Learning (EEL) is taken to encompass the development of team behaviours that collectively solve a problem through a process of ‘divide-and-conquer’. As such the terms *team* and *ensemble* will be used interchangeably. Team members will be referred to as *participants* or *agents* and a team must have more than one participant. The definition we will assume for a participant will take the following form:

Definition 8.1 Each participant (agent) performs an independent mapping from input (state) space to output (action) space.

Such a definition is adopted in order to distinguish participants from other approaches to ‘divide-and-conquer’ such as modularity, e.g. automatically defined

M. I. Heywood (✉)
Faculty of Computer Science, Dalhousie University, Halifax, NS, Canada
e-mail: mheywood@dal.ca

functions [115]. In short, the above definition implies that all participants are modules (because a participant never acts outside of a team), but not all modules are participants. Indeed, most instances of modularity follow a sequence of a callee referencing the module, passing a set of arguments, the module performing some computation and the callee using the returned variable(s) in some larger calculation. As such modules are typically representation-specific, whereas team participants are generally agnostic to the representation. Given such a stance, the focus of the chapter will be on the mechanisms which define the relationships between participants and therefore facilitate the processes of problem-solving through divide-and-conquer.

Historically, prior assumptions were often made about task decomposition and therefore team complement. For example, entirely homogeneous teams in a herding task [168] or requiring heterogeneous teams to consist of one instance of each agent ‘type’, e.g. writing and reading to memory [32] or classes in a classification task [147]. The ‘level of selection’ represents a reoccurring theme (Sect. 8.3), which is to say, does selection/variation/replacement appear at the level of team or participant? Initially, two basic approaches for evolving teams became established: the team as a single unit of selection versus sampling a participant from independent cooperative populations each time a team is composed.¹ Early examples in which teams were the unit of selection assumed a multi-tree representation, e.g. [81, 134]. At the same time, multi-population models (e.g. [87, 201]) increasingly became associated with cooperative coevolution and multi-agent systems (Sect. 8.4).

Diversity maintenance also represents a reoccurring theme, with different state representations having an impact on preferences for heterogeneous versus homogeneous team compositions [134]. However, diversity maintenance is also linked to a desire to solve tasks of increasing difficulty [15, 133]. Hence, there is no need to use an ensemble if a single participant can solve the task, but if an ensemble is necessary, how can a *meaningful* division of duties across participants be achieved?

This chapter develops the topic of EEL through two basic perspectives that initially developed independently:

- Ensemble learning as applied to regression and classification or a supervised learning perspective, hereafter supervised EEL or **sEEL** (Sect. 8.2).
- Cooperative coevolution as applied to multi-agent systems or a form of reinforcement learning, hereafter **maEEL** (Sect. 8.4)

Participants in sEEL are always heterogeneous, whereas maEEL places more emphasis on finding m types of agent to appear in a team of n agents where $n \geq m$. The concept of a team also introduces the topic of ‘level of selection’ (Sect. 8.3) to the team or the agent (or both). Section 8.5 reviews research that attempts to extend the concept of an ensemble to variable-sized ensembles (hereafter **vEEL**), with the objective of further scaling the scope of tasks that EEL can be applied to. The chapter concludes with a summary of applications that potentially benefit from assuming ensemble formulations (Sect. 8.6) and a concluding discussion (Sect. 8.7).

¹ Also synonymous with the Pittsburgh versus Michigan approaches to learning classifier systems.

8.2 Ensembles for Supervised Learning

An ensemble under a supervised learning context is taken to imply a partnership between n decision-makers to solve a supervised learning task such as regression or classification. Classically, a bias–variance decomposition [33, 118] might be used to establish the extent to which error can be attributed to:

- the average behaviour differing from the desired function. Typically denoted the *bias* in which case under-fitting the data is said to result in a solution with ‘high bias’.
- the ensemble is sensitive to the data set used to train the model. Typically denoted the *variance* in which case overfitting on training leads to high variance on test data.

Ensemble learning in general, therefore, attempts to compose a team of agents that exhibit diversity in their respective behaviours to optimize the bias–variance tradeoff. Such a framework has seen use with neural networks [29, 73, 78], genetic programming [2, 99, 100, 158] and ‘wide’ neural networks [24] as well as providing early insights to ensemble construction. With this in mind, ensemble learning as practiced outside of evolutionary learning often enforces diversity using one of three mechanisms:

- **Bagging:** n agents are independently constructed from n different samples (or ‘bags’) taken from the original training partition [33].
- **Boosting:** constructs n agents sequentially with the performance of earlier agents used to bias the selection of data to appear in the training partition for the next agent [34].
- **Stacking:** $n - 1$ agents comprising the ensemble are trained on $n - 1$ training folds. A “meta agent” is then trained from the $n - 1$ agents’ predictions to define the ensemble’s overall prediction [221]. Other variants include cascading in which n agents are added sequentially, augmenting the data with their prediction [70].

Successful ensembles identify participants that are sufficiently accurate, yet ‘disagree’ with each other [118, 156]. As a consequence, many approaches have been proposed for maintaining ensemble diversity [119, 120]. However, diversity in itself is not a guarantee for an effective ensemble, i.e. diversity is relative to the behaviour of other participants comprising the ensemble. Moreover, ensemble learning as defined above implicitly assumes that only one candidate solution is developed at a time. Conversely, *evolutionary* learning algorithms maintain multiple candidate solutions simultaneously (the population). This implies that there are potentially more avenues for pursuing diversity and participant composition than under non-evolutionary approaches to ensemble learning. Assuming a broader perspective on diversity and/or composition enables us to identify five themes that sEEL might support exclusively or collectively, Fig. 8.1. We detail each of the five themes in Sect. 8.2.1 and make summary observations on sEEL in Sect. 8.2.2.

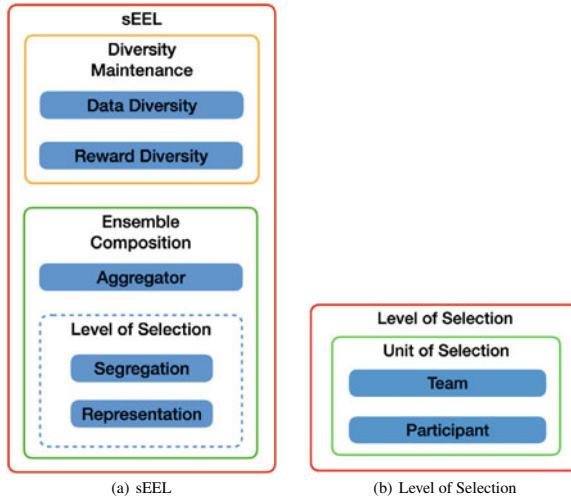


Fig. 8.1 Properties of significance to Supervised Evolutionary Ensemble Learning (sEEL). **a** Diversity maintenance takes the form of data diversity (what data is an ensemble participant constructed from) and reward diversity (what is the performance function each participant experiences). Ensemble composition reflects (1) the diversity of mechanisms assumed for aggregating participants' predictions, (2) the degree of segregation appearing in the operation of participants and (3) the diversity in representations assumed for participants. **b** Level of Selection has an impact on segregation and representation (Sect. 8.3)

8.2.1 *Diversity Maintenance in Supervised Evolutionary Ensemble Learning*

Figure 8.1 divides sources of diversity in sEEL as applied to supervised learning tasks such as regression and classification into explicit ‘diversity maintenance’ versus ‘ensemble composition’. Diversity maintenance is divided further into diversity through the data that different ensemble participants experience during training versus adaptation of the performance (reward) function, i.e. each participant potentially experiences a different performance function. Ensemble composition reflects different mechanisms by which the ensemble might be constructed. We divide this concept into three themes. Aggregation defines the mechanism assumed for combining the predictions from individual participants into an overall ensemble recommendation. Segregation characterizes how the role of participants might be revised during evolution and is related to credit assignment but reflects properties specific to evolutionary computation (e.g. the role of selection). Finally, representational diversity captures the ability of evolutionary computation to develop unique topologies as well as parameterize them. That said, most sEEL as applied to supervised learning tasks assume that the number of participants, n , is known *a priori*. The case of evolved participant complements and consequently context-specific participant deployment will be developed later (Sect. 8.5). In the following, we detail each theme in more detail.

Data diversity: implies that different participants of an ensemble are trained with different distributions of data. Adopting bagging strategies for ensemble learning represents a reoccurring theme [27, 68, 88]. Bagging might also be used with massively parallel computing platforms in order to accelerate the evolution of multiple participants from different partitions of the data, thus scaling sEEL to large regression and classification datasets [17, 18, 212]. Bagging methods were recently revisited for outlier reduction using niching [50], coevolution of participants and ensembles [170] and assessing participants over *multiple* bootstrap samples [214]. Likewise, partitioning the data into n folds (one fold held out from each data subset) enables n ensemble participants to be independently trained [90]. Competitive coevolution [69, 72, 126, 192] or active learning [83, 111, 185] have been used to adapt the data that ensemble participants experience *during* the course of evolution. That is to say, during evolution the most discriminatory exemplars will change as a function of the performance of the ensemble. In a further development, Lexicase selection represents a recent evolutionary selection mechanism for diversity maintenance using as little as a single exemplar to identify a parent [82]. Support for explicitly modular representations, such as sEEL, appears to provide a more effective mechanism for utilizing the available diversity [165]. Finally, ensembles have been incrementally constructed with the first participant evolved against the entire training partition. Thereafter, each additional ensemble participant is only evolved against the data that could not previously be labelled. This focuses each additional participant on what the ensemble cannot previously label [228].

Reward diversity: manipulates the performance function so that different participants of the ensemble experience different rewards. Early examples include boosting [68, 88, 162], cascade correlation [166], fitness sharing [172, 184] and negative correlation [128].² Other natural extensions include the use of multi-objective methods to trade off diversity and accuracy of ensemble participants [40, 41, 63, 123] and the simultaneous minimization of the resulting ensemble complexity [42]. Moreover, multi-objective performance criteria may represent a more robust predictor of (post-training) ensemble performance than ranking using a single objective [129]. Recently, novelty measures [37] and surrogate models [36] have been used to evolve ensembles for computer vision benchmarks such as CIFAR and SVHN. Under streaming tasks, other properties such as participant age have been used to prioritize participant replacement [67, 111]. Cooperative coevolutionary formulations imply that ensemble participants are sampled from n different populations [166]. The fitness that ensemble participants receive is used to direct development within each of the n different populations. This introduces issues regarding the fitness of ensembles versus participants (level of selection, Sect. 8.3) and a range of advantages and potential pathologies [159].

Ensemble aggregator: recognizes that once the participants of an ensemble are identified, then mechanisms need to be adopted to map from the n independent (participant) recommendations to a single (ensemble) recommendation [31]. Depending

² Negative correlation is related to the concept of covariance, the minimization of which potentially helps address the bias–variance trade off [41].

on the task, the optimal approach for combining recommendations might differ, i.e. majority voting, winner takes all [31, 85, 194] or Bayesian networks [198] in classification versus weighted average [47] and Bayesian model averaging [3] in regression. For example, an averaging assumption might penalize the development of specialists in regression tasks [161]. One potential approach for addressing this issue is to augment the ensemble with an additional participant (the ‘gate’). Such a participant learns how to switch or blend the output from the n ensemble ‘experts’ [161]. Such a ‘mixtures of experts’ architecture can be deployed hierarchically [96] and has seen widespread use with neural networks [226]. Nguyen et al. go a step further and actively coevolve ensembles using the mixtures of experts framework [150, 151]. More recently, the concept of a convex hull has been deployed to identify the ensemble participants with Boolean operators defining the ensemble aggregation function [123]. Tsakonas and Gabrys use grammatical evolution to optimize a hierarchy of aggregation functions deployed to different combinations of ensemble participants [208]. The aggregator itself can be evolved [31, 121], where this would be synonymous with wrapper approaches to feature construction, i.e. the aggregator would be a regressor or classifier [77]. Evolving either linear or non-linear aggregators has been shown to be more effective than a ‘fixed’ voting scheme [31, 121]. However, there is a computational overhead associated with the case of evolved non-linear aggregators [121]. Finally, we note that solution interpretability is likely to be impacted by the choice of aggregator, i.e. a ‘winner-takes-all’ (versus majority) operator attributing a prediction to a single (versus all) participant(s) of an ensemble.

Participant segregation: captures the degree to which the n participants *evolve* independently and is influenced by the approach to selection (level of selection, Sect. 8.3). This is distinct but related to the degree to which the data (or performance function) is manipulated to establish n distinct behaviours. For example, n independent populations could be evolved on the same data (e.g. [90]) as opposed to evolving n independent populations on different samples from the training partition (bagging). This also leads to the use of libraries or archives of previous behaviours so that an evolutionary method can identify: (1) the participants to include in the ensemble from the library; and (2) how to weigh their respective contributions [27, 94]. Hybrid approaches have also been proposed in which: (1) a spatial embedding is used to define the migration protocol between independent populations [68]; or (2) variation is allowed between populations associated with different partitions of the data [27]. Alternatively, the champions from each of the n independent runs can be compared for their relative diversity and entire populations pruned should their champions be deemed too similar [38]. Rebuli and Vanneschi propose a model for multi-class classification with n demes, i.e. variation takes place between demes [167]. Later, a ‘phase change’ takes place after which the n demes are treated as islands, i.e. variation limited to the same population. Multifactorial evolutionary algorithms solve multiple optimization problems using a single population in which different ‘tasks’ are present. Unlike cooperative coevolution, sharing takes place between participants associated with different tasks, i.e. soft segregation. The approach has been demonstrated in the context of evolving ensembles for multi-class classification problems [217]. Finally, ensemble participants might instead be strictly organized as a stack/cascade with a

participant evolved to make a prediction or defer judgement to a later participant(s) [228]. Different subsets of the dataset are then labelled by different ‘levels’ of the stack/cascade.

Representational diversity: implies that participants comprising the ensemble are free to adopt different types of representation. We distinguish between coarse- and fine-grained representational differences. *Coarse-grained representational differences* imply that entirely different machine learning paradigms were deployed to develop participants, e.g. decision tree induction, Naive Bayes, multi-layer perceptron. Such an approach has been demonstrated for non-evolutionary machine learning [132], but could be achieved using a cooperative coevolutionary approach (different populations for each representation). Neural networks potentially benefit from assuming different architectures, thus diversity in the networks appearing within ensembles has been considered [128] as has diversity in the activation function [209]. In addition, rather than assume a behavioural performance metric to select ensemble participants, a genotypic performance metric might be preferred. As such, ensemble participants are selected for those that have the most unique representations [68, 85, 112]. Likewise, grammatical evolution has been used to evolve ensembles with different types of representation for rule induction [95]. Cloud-based computing services have also been used to simultaneously evolve multiple participants with different representations in parallel for solving large classification problems, e.g. learning classifiers versus genetic programming [18]. In a related approach, Fletcher et al. assume that multiple ensembles are trained, each with a different ‘base classifier’ and an evolutionary multi-objective approach is adopted to select the best ensemble [63]. Note that the base classifier is common to the same ensemble, but different across ensembles.

Fine-grained representational differences acknowledges the ability of the same evolutionary computational paradigm to evolve the topology of individuals as well as parameterize them. As such this will be impacted by the approach to selection or the level of selection, Sect. 8.3. Two general approaches have been widely adopted: **multi-trees** [31, 147, 194] (i.e. specific to genetic programming) or **cooperative coevolution** [159, 166] (i.e. agnostic to the underlying participant representation). Both approaches assume that the number of participants, n , is known a priori. For example, it might be assumed that the number of participants and classes requiring classification are the same. Section 8.5 reviews representations that evolve ensembles/teams that support a variable number of participants. This ultimately implies that participants can be deployed depending on input context rather than always simultaneously deploying all participants.

8.2.2 Discussion

Section 8.2.1 established that sEEL for supervised learning provides multiple paths by which diversity in ensemble learning can be maintained/introduced. Bagging and boosting can be considered specific mechanisms by which data and reward diver-

sity might be supported. Stacking—the third form of diversity classically recognized in non-evolutionary ensemble learning—appears as an instance of representational diversity. However, assuming that participants are ‘evolved’ using variable length representations such as genetic programming means that unique participant topologies can be discovered.³ Representational diversity might additionally be considered when the number of participants, n , is not defined *a priori* as a hyper-parameter. Section 8.5 will develop this topic further. The property of participant segregation is specific to evolutionary computation on account of multiple participants being developed simultaneously. Diversity in the aggregation operation has seen a wide range of approaches, ranging from methods developed for Bayesian and neural networks to evolving the aggregator itself. We note, however, that such approaches are still limited to the ‘classical’ model of ensemble deployment: an ensemble consists of n participants and all participants participate in each prediction. We consider the implications of relaxing this constraint in Sect. 8.5.2 when graph-based representations are adopted.

A further set of observations can also be made independently of the specific diversity methods adopted, as follows:

- Strong ensemble performance does not imply that individual participants are also strong [31, 195]. Thus, there could be orders of magnitude differences between the performance of the ensemble and the participants comprising the ensemble. Indeed, selection for explicitly strong yet complementary ensemble members, as opposed to the original emphasis on ensembles composed from weak learners alone (e.g. [177]), represents a recent theme in sEEL [180–182].
- Participants of an ensemble are typically much simpler than when single ‘monolithic’ solutions were evolved for the same task. Depending on the task, the ensemble might also be collectively simpler than single monolithic solutions evolved for the same task [31, 88, 127]. Indeed, the participants of the ensemble might be more effective when simplicity is emphasized [123, 171].
- Assuming pairwise diversity measures does not necessarily lead to system-wide diversity [62]. Conversely, system-wide metrics, such as measuring the expected failure rate [90], have to date only been applied post-training. Using multi-objective formulations may benefit from defining the dominance relation on the basis of an entire performance characteristic, as opposed to single operating points [123], or modifying multi-objective formulations to incorporate validation data [176].
- Methods based on segregation and/or fixed partitions of training data are not able to adapt performance to changes in the behaviour of different ensemble participants. Adaptive reward functions might only be able to achieve this by rendering the training process serial, i.e. cascade correlation [166]. Conversely, serialization of the training process can result in considerable speedups when participants are able to distinguish between making a prediction versus deferring to another participant [228].

³ Other successful ensemble methods such as Decision Forests also have this property.

- The multi-tree representation assumes that performance is evaluated at the ‘level’ of the team (Sect. 8.3). Constraints can potentially be enforced to ensure context between different participants. For example, under a class classification problem, crossover might be limited to exchanging material between participants labelling the same class [147]. Additional participant-specific performance functions might be included in order to penalize unwanted participant properties [205, 223], although the ability to do this might be limited to specific applications. Multi-trees can also be evolved to provide *multiple* layers of abstraction. Such an approach has been demonstrated to be particularly effective for designing operators for image processing [89].
- Coevolutionary formulations for composing an ensemble have to sample one participant from n different populations in order to construct a single ensemble. Fitness then needs to be interpreted at the level of individual ensemble participants, resulting in various potential pathologies. This is discussed further in Sect. 8.3 and potential solutions are reviewed in Sect. 8.4.

Finally, we note that an ensemble actually presents multiple decisions regarding the ‘level of selection’, i.e. the team versus the participant. This represents a theme common to both sEEL and maEEL. Section 8.3 will therefore develop this topic specifically before ensembles are discussed from the perspective of multi-agent systems (Sect. 8.4).

8.3 Level of Selection

The concept of level of selection reflects the two levels at which credit assignment operates when multiple agents are involved in making decisions. As per Definition 1, an ‘agent’ (or ensemble participant) in this context is a fully functional decision-making partner that performs a mapping from input (state) to output action. Such an agent might be a neural network, genetic program, decision tree, etc. Thus, given a pool of agents and a team/ensemble comprising of n agents, the level of selection problem reflects the following two issues.

Definition 8.2 Team composition: is the likelihood of mapping an agent to a ‘position’ in the team/ ensemble consisting of a fixed number of agents.⁴ There are two extremes: all participants of a team are unique (*heterogeneous team composition*) or all participants of a team are the same (*homogeneous team composition*), Fig. 8.2.

Definition 8.3 Unit of selection: operates at the *level of agents* appearing within a team or at the *level of a team*, as shown in Fig. 8.2. Naturally, this implies that it is possible to express performance objectively at the level in question. Generally, the performance of a team can always be expressed objectively. However, depending on the task, expressing performance at the level of agents participating with a team may

⁴ Section 8.5 considers the case of variable-sized teams.

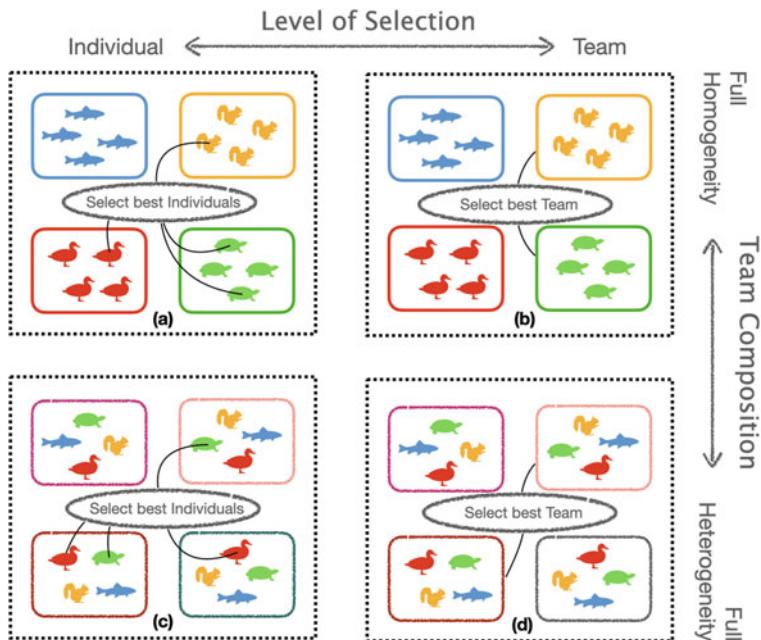


Fig. 8.2 Level of selection [215]. Full homogeneity (a) and (b) assume that all n agents per team are cloned from one of P genotypes. Full heterogeneity (c) and (d) assume M teams by n agents. Fitness evaluated at the level of individuals (a) and (c) versus team-level fitness evaluation (b) and (d). Reproduction at the level of individuals implies that two agents are chosen and an offspring agent results. Reproduction at the level of teams implies that two teams are chosen and variation operates at the level of team and possibly agent as well

or may not be possible. In the worst case, team and agent performance might not be aligned.

The previous discussion of ensembles (Sect. 8.2) implicitly assumed that agents participating within an ensemble were all different (heterogeneous). Moreover, when the multi-tree representation is assumed the unit of selection is typically that of the team [147]. Two parents are selected using team performance and crossover swaps team participants to create offspring, i.e. performed at the level of the team. In addition, ‘strong typing’ might also be assumed to direct the action of crossover such that only agents (sub-trees) with the same type are interchanged. Thus, for example, multi-trees applied to classification tasks might limit sub-tree crossover to classes of the same type [147]).

However, this need not be the case. Assuming that suitably informative performance functions could be designed for participants as well as at the complete ensemble/team, then the Orthogonal Evolution of Teams (OET) is able to [173, 204, 205]:

1. select parents at the level of participants but replace at the level of teams (OET1), or;

2. select parents at the level of teams but replace at the level of participants (OET2).

Such an ‘orthogonalization’ of the processes of selection and replacement was motivated to have credit assignment operate on the two levels simultaneously. Given an ensemble of size n there are n participant populations. Teams potentially represent columns sampled across the n participant populations [173]. Benchmarking with different classification tasks indicated preferences for different OET configurations [205]. However, OET2 appeared to be more effective at ‘repair’ when applied to a multi-agent foraging task [204]. We also note that the OET approach was independently discovered and used to evolve neural networks with an a priori fixed architecture [76], where each participant population was associated with a specific weight population and the ensemble was a complete network. A similar ‘orthogonalized’ process for the level of selection was used to direct the action of selection and replacement.

The concept of level of selection implies that decisions made regarding the team composition could have an impact on the degree of specialization versus the generality of agents supported in a team. For example, cooperative coevolution (Sect. 8.4) often assumes fully heterogeneous teams, making it difficult to establish teams composed of multiple instances of different types of agents. Moreover, as the level of selection is often that of the team, coevolutionary pathologies may arise such as:

- **mediocre stable states:** a form of deception in which agents collaborate to lead progress away from the global optima [74, 159].
- **relative overgeneralization:** agents with specialist behaviours are explicitly selected against [159].
- **loss of fitness gradient:** the performance of a few ‘affective’ agents is hidden by the poor performance of the majority of agents within a team. Also referred to as the ‘signal-to-noise’ problem [6].
- **hitchhikers:** in this case is synonymous with agents that exist within a team that does not contribute anything. Such agents reproduce, but do not contribute to the performance of the team [138, 141].

Section 8.4 revisits these pathologies in more detail under the context of multi-agent systems (cooperative coevolution is frequently used with multi-agent systems). Figure 8.2 summarizes the level of selection concept, albeit assuming 4 ‘types’ of agent and teams of size $n = 4$. Waibel et al. performed a series of empirical evaluations of all four discrete parameterizations of team composition and level of selection using three tasks [215]. The tasks were designed to reward: (1) individual foraging, (2) cooperative foraging and 3) altruistic foraging. Agent specialization was not considered in these tasks. Heterogeneous teams with individual selection (Fig. 8.2c) were preferable when no cooperation was necessary. When cooperation is necessary, homogenous teams are preferable. However, the study came with some caveats, in particular teams were entirely homogeneous or heterogeneous. This means that it was not possible to construct teams with a instances of agent type i . Such hybrid team compositions might be considered the norm for composing optimal solutions to many tasks, such as multi-agent robotics.

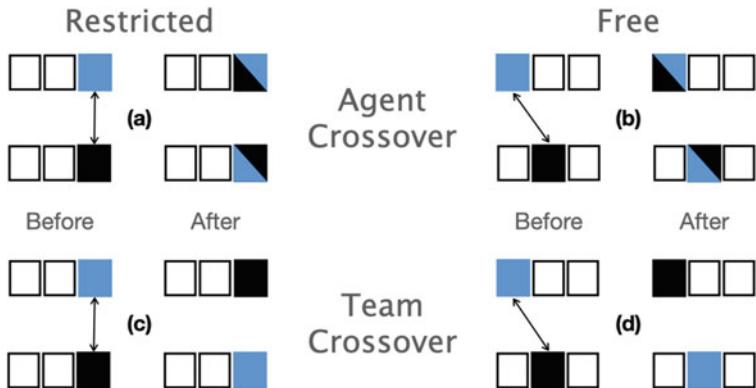


Fig. 8.3 Level of crossover [124]. Agent-level crossover **a** and **b** identify two agents within two teams and recombine the agent's genotypic material. Team-level crossover **c** and **d** identifies two agents within two teams and swaps the (entire) agent genomes. Restricted crossover **a** and **c** assume the position of the first agent. Free crossover **b** and **d** may choose agents from different team positions

A further study considered the impact of variation operators under hybrid team compositions [124]. The authors set themselves the goal of attempting to evolve teams consisting of 1,000 agents, in which specific combinations of agent types have to be found given 10,000 distinct types of agents. Uniform crossover with free or restricted gene transfer (FAR and RAS respectively) was assumed (Fig. 8.3). The underlying conclusions were that RAS would converge quickly, where this would enable it to solve tasks involving many different teams (highly heterogeneous team compositions). However, when more homogeneous teams were required, the diversity maintenance provided by FAS was the most effective. In addition, the authors show that by deploying FAS for the early generations and RAS in the latter generations, then hybrid team compositions can be discovered. This topic will be particularly important when composing teams for multi-agent systems (Sect. 8.4).

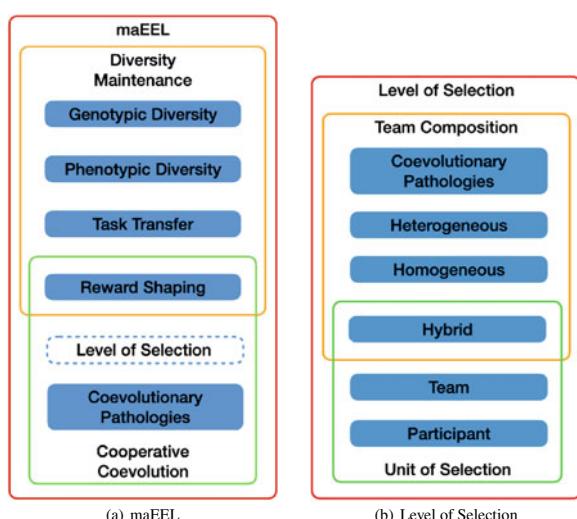
Questions left unanswered include the relative impact of attempting to evolve both agent and team composition simultaneously and the impact of gene linkage during the course of evolving team composition. Also left unanswered is the effect of reinventing agent policies. Lichocki et al. concentrated on team composition [124], whereas agent discovery might benefit from the trading of generic abilities.

8.4 Multi-agent Systems and Cooperative Coevolution

Multi-agent systems attempt to solve a task cooperatively using a finite set of agents and are typically applied to reinforcement learning⁵ (RL) tasks involving more than one decision-making agent. On the one hand, it is possible that a task might be solved optimally with the same agent deployed across the entire team or a purely homogeneous deployment. Conversely, at the other extreme, a task might be solved optimally with each agent comprising the team being unique (a purely heterogeneous deployment). Naturally, the number of agents, n , comprising the (multi-agent) team is known a priori. Thus, when describing the players participating in a soccer team, the number of players is known. Likewise, the number of robots available for performing a collective task might well be known. Under the sEEL context (Sect. 8.2) these issues are not as prevalent because the only team compositions that are appropriate are purely heterogeneous. Figure 8.4 summarizes the reoccurring themes that will be developed from an explicitly maEEL perspective.

Under a homogeneous deployment, one population is sufficient for sourcing the agents, and the concept of fitness at the level of team versus individual agent is aligned (Sect. 8.3). However, under heterogeneous settings, a multitude of possible mappings exist between population(s) sourcing the agents, and team composition (Sect. 8.3). At one extreme, a single population exists with each agent representing a participant of the multi-agent team. Under such a setting, incremental models of selection and replacement are assumed in order to gradually turn over the content of the population

Fig. 8.4 Properties of significance to Multi-agent Evolutionary Ensemble Learning (maEEL). **a** Two major themes are identified: (1) Diversity maintenance is parameterized from the perspective of genotypic/phenotypic diversity, task transfer and reward shaping. (2) Cooperative evolution which is impacted by the level of selection (Sect. 8.3), coevolutionary pathologies and also reward shaping



⁵ Reinforcement learning implies that an agent interacts with an environment and is rewarded for maximizing the cumulative rewards as experienced over a finite or unlimited number of interactions [200]. Applications include robotics, scheduling, game playing and stock trading.

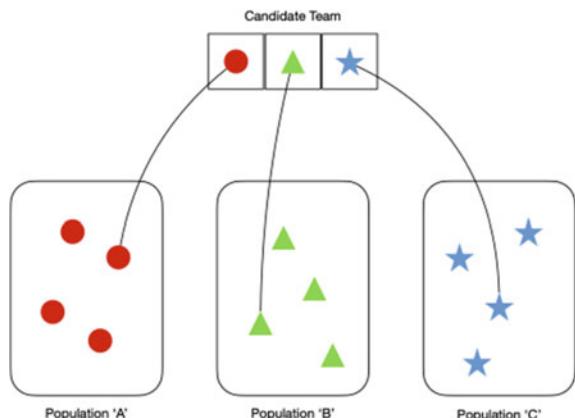
and speciation/fitness sharing used to maintain population diversity, e.g. [197]. At the other extreme, each agent is associated with an independent population, this is the case of cooperative coevolution as formulated by Potter and De Jong [166].

To date, the cooperative coevolutionary formulation represents the typical starting point, Fig. 8.5. As such, one agent is sampled from each population in order to construct a single multi-agent team of n agents [166]. Thus, population i only ever source agents for ‘position’ i in the multi-agent team. Put another way, population i only ever source agents of ‘type’ i ; therefore, the *context* between agent and position within the team is explicit. Reproduction and replacement only occur between participants of the same population. Moreover, cooperative coevolution is agnostic to the representation assumed to define the agents. Indeed, the populations associated with each agent (type) could have entirely different representations.

Naturally, performance is evaluated at the level of a team. However, fitness of agent i from an n agent team is a function of the other $n - 1$ agents participating within the multi-agent team. As a consequence, N samples (and therefore fitness evaluations) are made of the agents from the other $n - 1$ populations in order to establish the fitness of agent i . However, cooperative coevolution requires a measure of fitness to be returned to individual agents in order to direct the population-specific process of reproduction and replacement. At this point, pathologies can appear during credit assignment. For example, it became apparent that using the average (team) fitness from the N partnerships used to quantify the performance of agent i results in team compositions that favour mediocre stable states [74, 159]. In addition, relative overgeneralization may appear, where many individuals represent ‘jack-of-all-trades’ style solutions. This in turn precludes the development of specialists that could improve the overall collective performance of a team [159].

An early mechanism adopted for reducing these biases was to assign an agent its best fitness from the N partnerships as opposed to the average of all N partnerships [220]. This was later refined to using an annealing schedule to reduce the number of partnerships assessed as the number of generations increased [160]. Most recently, the

Fig. 8.5 Cooperative Coevolution [166]. Populations A, B and C only provide agents for team positions 1, 2 and 3, respectively. The context/type for each agent is therefore explicit. However, this forces teams to be heterogeneous. Evolving hybrid compositions requires the introduction of different team-level representations (Sect. 8.4.3)



issue of premature convergence in cooperative coevolutionary approaches to multi-agent systems has been addressed through the use of diversity measures (Sect. 8.4.1) and/or task variation (Sect. 8.4.2).

A related pathology that multi-agent systems can experience is with regards to a loss of fitness gradient. Specifically, as the team size increases, then the performance of one ‘good’ agent can be lost in the ‘noise’ created by all the poor-performing agents, i.e. a signal-to-noise problem. Attempting to address this problem by increasing the number of partners that agent i is evaluated with will not scale as the team size increases.

Agogino and Tumar proposed to address this problem by adopting factored (difference) performance functions [6]. This implies that for any state, the runner-up solution is known such that the effect of substituting the runner-up for the target agent can be estimated. Such functions have been demonstrated for a cross-section of applications involving agent deployments, e.g. sensor networks [4], air traffic control [5], multi-rover co-ordination [6].

Factored performance functions effectively reshape the reward such that improvements by a single agent also improve the multi-agent reward [44]. Shaping the reward in this way is easier to achieve when the agents are ‘loosely coupled’. Loose coupling implies that the actions of one agent are not closely dependent on another, i.e. a form of gene linkage. It is more difficult to formulate factored performance functions when agents are tightly coupled [51]. For example, should one agent be doing something useful, such as attempting to push a highly valued object, unless the other agents also perform the same action, there might be no reward. This plays into being able to more explicitly control the degree of homogeneity/heterogeneity so that there are a instances of agent type i and b instances of agent type k . Hence, rather than attempting to evolve all agents independently, it might only be necessary to evolve 2 different agent types in a team of 20. Evolving teams with a hybrid mix of homogeneity/heterogeneity is discussed further in Sect. 8.4.3.

8.4.1 Diversity Maintenance

Diversity in cooperative coevolution can be promoted using behavioural (phenotypic) or genotypic properties at the level of team and/or agent. Diversity in the underlying team objective is often achieved by adopting multi-objective formulations in which several possibly conflicting objectives describe the underlying goal [43, 227]. Pareto formulations encourage tradeoffs between the different objectives to be investigated by different team complements. Moreover, they can also be used to provide a sequence of objectives of incremental difficulty that can lead to solving some (more difficult) overall objective [207].

Diversity maintenance represents a general challenge when evolving multi-agent systems. As such multi-objective methods have been widely adopted in an attempt to simultaneously develop task-specific objectives and promote diversity [144, 145]. Several approaches have appeared, including initially developing diverse behaviours

on a set of source tasks using multiple novelty objectives. The non-dominated individuals are used to seed the population for the target task(s) [143]. Conversely, Doncieux and Mouret include the task-specific objective throughout, but switch between different diversity objectives [54]. Such task switching has been recognized as a potential mechanism for promoting ‘modular’ solutions in general [164].

Several studies have demonstrated their applicability across a range of benchmark tasks: predator-prey [74], herding [74], multi-rover [74], half-field offence soccer [103] and Ms Pac-Man [103]. Genotypic diversity can be captured by measuring the pairwise similarity of team content between teams [74]. Moreover, such metrics can also be formulated for variable size teams [103]. Novelty metrics have been evaluated at the level of individual participants of a team as well as at the team level. A distinct preference for maintaining diversity at the level of the team has been reported [74, 152]. Moreover, experiments with and without behavioural diversity, genotypic team diversity and multiple source tasks indicate that the most general solutions appear when multiple forms of diversity appear [74, 103, 152].

8.4.2 Task Transfer

Task transfer (or layered learning) represents a mechanism for scaling learning algorithms in general and multi-agent (or cooperative coevolutionary) systems in particular to tasks that cannot be solved directly (*tabula rasa*), e.g. [178, 199, 202, 219]. This is also referred to as the bootstrap problem [143, 207]. As such, one or more source task(s) need identifying, typically *a priori*, with the evolution of the multi-agent system first performed on the source task(s). Policies or entire teams are then used as a ‘run transferable library’ for use during an independent evolutionary cycle conducted against a target task [101, 103]. The library might be used as seed material for initializing the population evolved against the target task, i.e. the agent-teams discovered under the source task are modified. For example, participants taking the form of code fragments⁶ have been evolved using learning classifier systems on small-scale Boolean tasks and scaled to solve Boolean problems of larger dimension [13, 91]. Conversely, the solutions from the source task might be referenced by agents as evolved against the target task [101, 103], i.e. the solutions identified under the source task are not subject to variation during the development of the target task. The end result is an ensemble with a variable-sized structure that deploys solutions to source tasks in innovative ways to solve target tasks [108, 189] or the evolution of ensembles for solving multiple target tasks simultaneously [104, 108, 109] (reviewed in Sect. 8.5). In some cases, configurations of the task can be specified by members of an independent population. Competitive coevolution can then be used to establish an ‘arms race’ between candidate solutions (the teams) and the task [117]. This leads to a more open-ended process of team development [186, 190].

⁶ Tree structured GP with a depth limit of two.

To date, task transfer under neural evolution tends to assume a population-seeding approach to task transfer [149]. Early examples evolved a neural network under a lower dimensional input space⁷ and transferred this to a higher dimensional version of the task [22]. The HyperNEAT framework was assumed for this purpose, and teams were not explicitly present. However, this set the scene for use of a ‘bird’s eye’ representation in which an explicitly multi-agent task (e.g. evolution of agents to play keepaway soccer) could be evolved to transfer between unrelated tasks [213]. One element of the approach was to reformulate the original egocentric task description to that of a two-dimensional global ‘grid world’ representation as viewed from above. HyperNEAT could then be used to scale to increasing numbers of players for the keepaway soccer benchmark task without target task training by adding a third dimension to the original bird’s-eye view [45, 46]. The concept of a team is now a continuum. HyperNEAT represents an example of a developmental framework in which neural networks evolved with cyclic activation functions (denoted a Composite Pattern Producing Network, CPPN) describing the parameters appearing in the target architecture. The inputs to the CPPN represent the co-ordinates of the input and output of the target architecture. Adding a further HyperNEAT input to index the number of teams effectively scales the approach to arbitrary numbers of agents per team [45, 46]. Diversity was again a significant issue, with the combination of (task-specific) performance objectives and novelty search resulting in the most effective agents under the keepaway soccer task [152]. Such an approach rests on the use of (1) a bird’s-eye representation and (2) HyperNEAT. For example, the bird’s-eye representation removes some of the properties of the keepaway soccer task that made it challenging (e.g. navigation using an egocentric representation). HyperNEAT also composed solutions in the form of a 160,000 parameter feed-forward neural network, therefore losing solution transparency.

8.4.3 *Hybrid Homogeneous–Heterogeneous Multi-agent Systems*

The ‘signal-to-noise’ pathology in multi-agent systems (cooperative coevolution) can potentially be addressed by explicitly supporting the evolution of hybrid team compositions (see also Sect. 8.3). Thus, a team of 11 soccer-playing agents could be parameterized by specifying four types of agents (goalie, defender, mid-field and striker) and the number of each type of agent evolved. Nitschke et al. adapted the classic cooperative coevolutionary framework of Potter and De Jong [166] (fully heterogeneous) to address this issue by introducing an inter-population crossover operator [153, 154]. To do so, the genotypic and behavioural similarities between different populations are measured. This implied that a particular neural encoding had to be adopted. When the inter-population similarity passes a threshold, then

⁷ Representing a board game with complete state information.

crossover of material between populations can take place. There are still as many populations as agents, but subsets of populations can now influence each other.

Early examples of evolving hybrid team compositions specific to genetic programming include the use of an ‘automatically defined group’ [79] and the ‘Legion system’ [30], both of which assume a tree-structured representation. Automatically defined groups rely on special purpose crossover operations to manage the development of teams over multiple level of selection. The Legion system not only relied on specialized crossover operators (specific to tree-structured genetic programming) but also introduced an entropy based heterogeneity measure in order to encourage/reward the division of labour between agents.

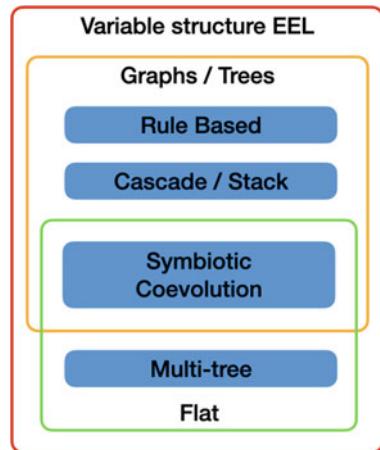
More recently, Gomes et al. explicitly define a team encoding to distinguish between agent type and the number of instances of each agent [75]. Specifically, the Potter–De Jong cooperative coevolutionary framework is still assumed, but this time the number of independent populations reflects the number of agent types. One set of variation operators functions at the team level and another set operates at the agent level [75]. Team-level variation can decrease or increase the number of agent types, thus merges or splits the corresponding agent populations. The approach is still independent of the agent representation, but the same representation has to be employed throughout.

A further aspect of the signal-to-noise pathology is that there are two components to the reward function: a ‘high-frequency’ component and a ‘low-frequency’ component. The high-frequency component is associated with the agent to environmental interaction, i.e. reinforcement learning [200]. The low-frequency component is associated with satisfying multi-agent components of the reward. With this in mind, neuro-evolutionary approaches have been proposed in which gradient-based temporal difference methods are used to optimize properties of individual agents, while evolutionary computation is employed to design the team [110]. Naturally, such an approach assumes a real-valued numerical representation [218] in order to support both high-frequency (gradient decent) and low-frequency (evolutionary computation) credit assignment.

Finally, we also note the use of ‘tagging’ to dynamically identify which team a participant belongs to [86, 169]. Thus, participants are assigned on the basis of the similarity⁸ of their respective tag values. This method of dynamic team selection has been extensively analysed within the context of the iterated prisoners dilemma [20]. In particular, only members of the same group play each other, resulting in participants increasingly adopting altruistic strategies as opposed to defector strategies as the number of tags increases. This is to say, the altruistic participants learn to increase the number of teams in order to decrease the likelihood of their team including a defector. More recently, Lalejini et al. used tags to identify the conditions under which agents were associated with states. This enabled agents to evolve ‘event driven’ decompositions of tasks [122].

⁸ The similarity metric could also be probabilistic, resulting in a source of variation in participant-to-team association.

Fig. 8.6 Properties of significance to Variable-sized Evolutionary Ensemble Learning (vEEL). Flat implies that the ensemble is organized with all agents participating in every decision. Graph/Tree implies that ensemble participants are organized hierarchically with different subsets of individuals participating in decisions depending on the input state



8.5 Ensembles with Variable Size-Structures

All the above research assumed ensembles/multi-agent systems in which the number of participants/agents per team was specified a priori. However, it might actually be desirable for this to evolve as well. Figure 8.6 highlights themes of significance for evolving variable-sized evolutionary ensemble learners (vEEL).

One approach to vEEL might be to repeatedly evolve a fixed-sized ensemble approach (Sect. 8.2) over a range of ensemble sizes. Naturally, this would incur a significant computational overhead. Multi-tree representations have been proposed for evolving teams of up to n participants by introducing a ‘null’ program at the sub-tree level [21]. Multi-objective archiving has also been used to cooperatively evolve ensembles of classifiers for multi-class [139] and binary [25, 26] classification problems. As such the complexity of the resulting archive is a function of the task difficulty, i.e. the number of participants per class is an evolved property. Such an approach deploys participants in parallel (or ‘Flat’ in Fig. 8.6). Conversely, at the other extreme, participants might be organized as a hierarchy or a cascade [70]. Potter and De Jong assumed the specific case of cascade correlation in order to let cooperative coevolution incrementally evolve a neural network without a priori specifying the number of hidden layer neurones [166]. However, this solution was specific to the case of neural networks with a cascade correlation topology [61], so the coevolutionary process was no longer agnostic to the representation assumed for participants. A further approach to cascade/stack construction has been proposed in which participants distinguish between making a prediction or not [228]. If a prediction is made, no further participants need to make a decision. If a prediction is not made, then the next participant in the hierarchy is queried.

Sections 8.2 and 8.4 for the most part assumed that all participants collaborated at the same level (parallel/flat agent deployment). Conversely, graphs have the ability to describe hierarchical relationships and enforce spatial and/or temporal dependencies

between different participants. Graphs have previously been used as an organizing principle for instructions within programs (e.g. [135, 193]) or state machines (e.g. [16, 91]). However, two works have also considered using conditional statements attached to a ‘header’ of ensemble participants to define which out of several ‘following’ participants to execute: PADO [203] and Linear Graph GP [97]. In both cases, given a start or root participant, the participant is executed and the participant’s conditional statement is assessed. Depending on the conditional statement, either another participant is executed or not. The conditional statements from PADO assess the state defined in a common memory (to all participants), whereas the conditionals of Linear Graph GP assess the register values of the parent participant. As such, a single participant is associated with graph nodes and arcs are associated with each condition statement. The concept of a team is therefore ‘distributed’ across the graph as a whole. Note, that a participant’s action is now either a reference to another participant or a task-specific action, i.e. the number of actions has increased. This is still consistent with Definition 1 because a participant is completely executed (without transfer of execution to different participants) before action selection can take place. In effect, by adopting a graph, hierarchical relationships now exist so that a participant can defer task-specific action selection to a more specialist participant. We identify these approaches as ‘rule based’ in Fig. 8.6.

More recently, graphs have been evolved for which each node represents a team and each participant an arc. Given a start or root node, a subset of the teams in the graph is visited, depending on the state of the environment. The process of visiting nodes (teams) continues until an arc is selected that ends in a task-specific action as opposed to another team. In the following, we review attempts to evolve variable-sized ensembles (including trees of teams) using this process (Sect. 8.5.1) and then generalize this to the case of graphs (Sect. 8.5.2).

8.5.1 Variable Size Ensembles Through Symbiosis

Symbiotic models of cooperative coevolution provide a generic approach for discovering the composition of variable-size ensembles / multi-agent teams using only two populations [84]. Thus, unlike the Potter–De Jong formulation of cooperative coevolution (Sect. 8.4), the number of populations is independent of the number of participants appearing in the team. Specifically, there is a team (host) population and a participant/ agent (symbiont) population, Fig. 8.7. The team population attempts to discover useful team compositions whereas the agent population provides the pool of participants that may appear within a team. Participants may appear in multiple teams, and the team composition has to be unique. An early example of symbiosis was used to evolve fixed topology neural networks, i.e equivalent to a fixed size team [142].

In order to extend the two population model into a variable length representation (thus hybrid homogeneous/ heterogeneous compositions), agents need to distinguish

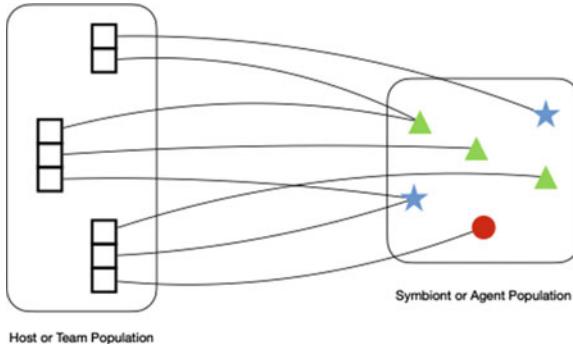


Fig. 8.7 Symbiotic cooperative coevolution with bid based agents [126]. Participants from the Team population (LHS) are defined as pointers to participants of the Agent population (RHS). For illustration the Agent population is considered to consist of three types of action (e.g. class 1, 2, 3), represented by the star, triangle and circle. Valid teams should consist of Agents representing at least two different types of action. The same Agent can appear in multiple Teams, but the team complement should be unique

between context and action [125]. Thus, agent execution⁹ is used to identify the bid (or confidence) given the environment's state. All agents from the same team provide a bid; however, only the agent with the highest bid ‘wins’ the right to suggest its action [126, 127, 223]. This means that multiple agents might appear in the same team with the same action, but with different contexts, adding another degree of flexibility to the process of divide-and-conquer.¹⁰ Moreover, teams need not start with the full complement of agent types, but instead incrementally develop the relevant type complexity.

In the simplest case the action, a , is just a discrete scalar value.¹¹ Agent actions are chosen from the set of task-specific actions $a \in \mathcal{A}$, e.g. class labels. We now have an agent representation that can evolve teams consisting of any number of team participant types and different sizes. Moreover, the parent pool is identified at the level of teams. Any team participants not associated with a surviving team are deleted, i.e. task specific fitness need only be defined at the level of the team population. Hitchhiking is still an issue but can be addressed by periodically dropping team participants that never win a round of bidding. The resulting symbiotic model of coevolution was demonstrated to be superior to evolution without ensembles [127] and competitive with learning classifiers and support vector machines under multi-class classification tasks [126, 223]. Further developments included scaling to high-dimensional ($\geq 5,000$) classification tasks [56, 127] and operation under non-stationary streaming data classification tasks (Sect. 8.6).

⁹ The executable component of an agent could be a program or a neural network.

¹⁰ Agent context divides the input space into a region and associates its region with an action. As multiple programs appear in the same ensemble, multiple regions–actions appear.

¹¹ Support for real-valued actions introduces an action program at each agent [23, 106].

The approach has also been extended to produce hierarchical teams for reinforcement evolutionary ensemble learning (**rEEL**) [55, 105]. This is distinct from maEEL as rEEL solutions describe a single agent policy but explicitly decompose the task/representation. One implication of this is that strategies for solving one aspect of a task can be reused for solving other aspects of a task [103]. The resulting tree structure represents teams as tree nodes and agents as arcs. Leaves represent atomic actions. The tree is constructed bottom-up (but evaluated top-down from a single root team), with successive layers describing their actions in terms of pointers to previously evolved teams [55, 105].

In order to ensure that different teams represent different strategies, then diversity maintenance (during evolution) represents a re-occurring theme (Sect. 8.4.1). In particular, different tasks could be interleaved with the development of the hierarchical team, thus a natural approach for task transfer [101, 103]. Alternatively, competitive coevolution has been used to develop an ‘arms race’ between tasks and solution strategies resulting in the organization of thousands of team participants for optimally solving (tens of millions of) Rubik’s Cube configurations [186, 190]. As an additional benefit, unlike monolithic solutions (a single large agent), only one team per tree level is evaluated to determine the ultimate action, making for extremely efficient solutions when compared to neural networks [103, 187].

8.5.2 *Tangled Program Graphs*

The symbiotic framework was also generalized to organizing teams into graphs of teams, leading to results that are competitive with deep learning solutions on (visual) reinforcement learning problems [102, 187]. Indeed, the resulting ‘tangled program graphs’ could learn policies for playing multiple game titles simultaneously under the ALE benchmark, i.e. multitask learning [104]. The graph representation gives direct insights into the nature of the decomposition of agents to decision-making under different game titles, i.e. interpretable machine learning. Later developments demonstrated that the approach also scales to multiple types of partially observable task¹² such as Dota 2 [188] and ViZDoom navigation [108]. Support for real-valued actions under tangled program graphs enabled problems in recursive forecasting [108], multitask control [109], and biped locomotion [14] to be addressed.

One of the unique benefits of the graph-based ensemble is that they are exceptionally efficient to train and deploy. The composition of agents, teams, and graphs is incremental and entirely emergent. This results in solutions whose complexity reflects the properties of the task, not the initial dimensionality of the state space or a priori assumptions about suitable solution topologies. Thus, under visual reinforcement tasks, less than 20% of the pixels are used to reach a decision [102, 104, 187].¹³

¹² Implies that the agents can only solve a task by forming decisions from the internal state (memory) as well as the external state as provided by the environment.

¹³ Decreases to < 5% as the dimension of the visual state space increases [107].

The complexity of the entire solution is typically three or more orders of magnitude lower than deep learning approaches to the same task [102, 104, 187]. Specifically, in order to make a decision, only a fraction of the graph is evaluated, which this changing as a function of the state. Insights are then possible about the functionality of participants in the evolved team [107]. In addition, this has led to the use of graph ensembles in ultra-low power signal processing applications such as real-time intrusion detection on IoT devices [196]. Indeed, solutions to the ALE visual reinforcement learning benchmark [137] have been demonstrated using nothing other than Raspberry PI embedded controllers [48].

8.6 Applications and Future Research

Table 8.1 provides a review of specific application developments that have benefited from adopting an evolutionary ensemble learning approach. Thus, aside from the application of evolutionary ensemble methods to a wide range of regression and classification problems (summarized in Sect. 8.2), we note that the underlying requirements for constructing ensembles are also the requirements for feature construction/engineering using wrapper or filter methods [77]. Specifically, feature construction requires that a diverse set of features be engineered to improve the performance of a regression or classification task. Indeed, many evolutionary approaches to feature engineering assume a multi-tree representation, e.g. [7, 21, 148, 206]. Thus, the number of ensemble participants (n) represents the number of features constructed [7, 116, 148, 183]. More recently, multiple features (participants) have been engineered per class (e.g. [56, 127, 206]) or features (participants) are evolved that are capable of transferring between different environments [8]. Multidimensional genetic programming approaches the (feature construction) task from a different perspective by attempting to discover a low-dimensional space appropriate for describing all aspects of the task [39, 139]. In general, what is particularly impressive with ensemble solutions to the feature construction problem is the capacity to discover very accurate low-dimensional feature spaces from applications described in terms of higher dimensions [56, 148, 171] or from low cardinality datasets [9].

Future work might consider the use of ensemble diversity measures originally developed from the perspective of feature construction. For example, limitations may appear when relying on pairwise (feature) diversity measures [90] or attribute frequency importance [49], whereas a permutation-based sensitivity analysis can avoid the linear correlation constraint (e.g. [12, 49, 93]). Future work might further consider the utility of permutation schemes for interpretable solutions [57].

In general, scalability represents an underlying theme for sEEL. One approach might be to make use of the increasing availability of parallel computing platforms, such as cloud [18, 68, 212] or GPU [17]. Alternatively, methods for algorithmically reducing the number of evaluations might be adopted, such as surrogate models [36] or active learning [185]. Both of the latter have been benchmarked on computer vision benchmarks such as CIFAR resulting in much simpler solutions than

Table 8.1 Summary of EEL research with specific application contexts

Application area	Publication
Control systems	
Direct control	[55, 76, 161]
Dynamical systems modelling	[1, 47]
Path planning/obstacle avoidance	[54, 105, 144, 145]
Robot locomotion	[14]
Data analysis	
Boolean expression learning	[13, 91, 92]
Cancer prediction	[7, 11, 85, 113, 222]
Instance selection	[71]
Interpretable solutions	[35]
Multi-label classification	[146]
Multi-objective	[21, 40, 42, 63, 123, 129, 139, 176]
Outlier reduction	[50]
Software fault utilization	[191]
Feature Construction	
Application specific	[7, 21, 85]
High-dimensional (input)	[56, 127, 148]
Image data	[9, 27, 28, 36, 37, 89, 171, 180, 203]
Inter-task feature transfer	[8, 28, 149]
Low cardinality (training)	[9]
Multi-feature construction	[56, 89, 206]
Multi-task or transfer learning	
Supervised	[8, 28, 149, 178, 180]
Reinforcement learning agents (memory)	[106, 108, 109, 188, 189]
Reinforcement learning agents (reactive)	[22, 48, 101–104, 112, 186, 187, 190]
Multi-agent Reinforcement learning	
Air traffic control	[5]
Five aside soccer	[75]
Keepaway soccer	[101, 152, 213, 219]
Half-field offence	[101, 103]
Multi-agent communication	[52, 140, 225]
Multi-rover	[6, 51, 74, 75, 110, 175, 224]
Predator-prey	[74, 225]
RoboCup	[15, 133]
Sensor networks	[4]
Scalable training	
Active learning	[69, 72, 126, 127, 139, 185, 192]
Algorithmic efficiency	[214, 228]
Cloud or cluster computing	[18, 68, 212]
GPU platform	[17]
Surrogate fitness	[36]

(continued)

Table 8.1 (continued)

Application area	Publication
Scheduling	
Capacitated arc routing	[216]
Dispatching rules	[58–60, 80, 163]
Trip planning	[98]
Streaming	
Benchmarks	[19, 64, 65, 67, 83, 210]
Churn detection	[211]
Intrusion detection	[66, 111, 196]
Trading agents	[130, 136]

currently available using deep learning. In addition, organizing ensemble agents as a stack/cascade was shown to scale sSEL to data cardinalities in the hundreds of thousands in less than 30 s on a regular CPU [228]. Future work might continue to investigate how different ways of composing ensembles trades off accuracy versus training efficiency versus interpretability [35].

A related application of sSEL is that of streaming data forecasting and classification [83]. Some properties that make the streaming data environment challenging yet appropriate for sSEL might include.

- Non-stationary nature of the underlying task (drift and/or shift) which might imply that mechanisms need to be identified for detecting the onset of change and reacting appropriately [64, 65, 67]. Ensembles are capable of reacting to changes more effectively than non-ensemble approaches because the implicit modularity enables specific participants to be retired/replaced as their performance degrades. This then leads to solutions that are more adaptable than without the use of ensembles [210].
- Anytime nature of deployment implies that in time series classification a champion classifier has to be available for labelling the *next* exemplar before any model has encountered it. This means that the champion classifier might vary over the course of time.
- Imbalanced or limited availability of label information. Given that streaming data is typically experienced on a continuous basis (there is no ‘end’ to network or stock market data), models are constructed from the content of a sliding window, i.e. a finite number of exemplars. This can lead to different strategies being adopted for retaining data beyond the most recent window content, e.g. data subset archiving and ensemble archiving [19, 111].

To date, streaming ensemble methods have been applied to applications in trading agents [130, 136], intrusion detection [66, 111], electricity utilization [131], satellite data [64], and churn detection [211]. Specifically, Mabu et al. assume a graph representation that explicitly captures dynamics of stock trading, whereas Loginov and Heywood coevolve an ensemble of technical indicators and decision trees for cur-

rency trading and utility forecasting. Both Folino et al. and Khanchi et al. emphasize the ability of ensembles to dynamically react to changes to the underlying properties of the streaming data. A related topic is that of dynamical systems identification in which each (independent) variable has a participant evolved and a suitable aggregation function applied [1]. Particular challenges in this setting might include evolving explainable solutions. Other tasks with dynamic properties that have deployed evolutionary ensemble approaches include different formulations of scheduling tasks that are addressed through the evolution of dispatching rules, e.g. [58–60, 80, 163].

Task transfer and multi-task learning also represent areas in which rEEL can potentially produce advances to the state-of-the-art. Task transfer is typically assumed when the ultimate objective is too complex to solve ‘tabula rasa’ [219]. Instead, solutions are first evolved to solve simpler source tasks before the ultimate target task is encountered [202]. Likewise, multitask learning requires that solutions to multiple tasks are discovered such that a single champion for all tasks is discovered. This can be particularly difficult as, aside from the difficulty of solving each task, the agent has to establish what environment it is in. Current results indicate that EEL approaches are well placed to incrementally absorb multiple source tasks [8, 101, 103, 149, 186–188] as well as solve multiple tasks simultaneously [28, 98, 104, 108, 109].

Future challenges might include extending these results to environments requiring lifelong/continuous learning (e.g. [179]) and addressing pathologies such as catastrophic forgetting (e.g. [114]) or returning solutions that are interpretable [57, 174]. Given the explicit use of structured representations in EEL, interpretable solutions might represent a potentially significant new development for EEL. Moreover, some of the opaque properties of individual participants might be amenable to simplification using techniques developed for interpretable AI (e.g. model debugging using adversarial learning or perturbation-based analysis [57]). Indeed, there is already a rich history of employing competitive coevolution (the EC approach to adversarial learning) to develop more robust solutions to computer security applications [157].

Multi-agent systems will continue to develop, particularly with respect to evolutionary robotics [53]. One avenue that is beginning to see some results is with regards to the evolution of communication [140] or stigmergy [225] in multi-agent systems. In particular, Mingo and Aler demonstrate that agents can evolve spatial languages with specific syntactical properties using the evolution of grammatical evolution [155]. As the number of agents and objects increases, then the sophistication of the evolved language also increases [140]. Developments of this nature may lead to agents teaching agents [178] and/or forms of problem-solving that uniquely reflect the mixed ability of the agents to perform different tasks [10].

In addition, multi-agent approaches have appeared in gaming applications in which group behaviours might be desirable. The RoboCup competition represented an early example [15, 133], with more recent works using specific aspects of the full-team competition as smaller scale benchmarks, e.g. keepaway [101, 152, 213, 219], half field offence [101, 103] or five-a-side soccer [75]. First-person video games have also been used to demonstrate the development of squad behaviours using EEL [197] and confirmed that teams increasingly make use of communication as the amount of

visual information decreases [52]. Likewise, partially observable environments have also been used to demonstrate: (1) navigation behaviours under visual reinforcement problems [108, 188, 189] and (2) time series prediction [106, 108] and (3) agents able to solve multiple control problems simultaneously [109]. The resulting EEL graph structures demonstrate an emergent division of duties between, for example, participants that write to memory (a specialist) and those that read (everyone) or the types of tasks addressed by different parts of the graph-based ensemble.

8.7 Discussion

EEL in general has a long and rich history in which the desire to scale evolutionary computation to increasingly more demanding tasks represents an underlying motivation. To do so, the divide-and-conquer approach to problem-solving is assumed as a general principle. However, in doing so, several pathologies potentially appear/need recognition of which level of selection and diversity maintenance represent reoccurring themes. The level of selection reflects the fact that a solution is composed of multiple participants, whereas the performance function might only operate at one level. Moreover, gene linkage can appear between participants and diversity can appear at multiple ‘levels’, making credit assignment difficult to measure. In addition, EEL as applied to multi-agent systems cannot just assume that teams will be heterogeneous. Instead, specific combinations of different types of agents might be the norm.

Historically, supervised learning applications of EEL have assumed a fixed-sized ensemble defined by a multi-tree representation (Sect. 8.2). This means that the participants are always heterogeneous and the unit of selection is that of the team. However, as demonstrated by the OET algorithm (Sect. 8.3), this might represent a sub-optimal model of selection. Conversely, multi-agent tasks often assume cooperative coevolution as the starting point for defining a team of agents. The cooperative coevolutionary model not only provides a wider opportunity for developing mechanisms for answering the level of selection question but also potentially introduces multiple pathologies (Sect. 8.4). Attempting to develop variable-sized ensembles means that a participant has to distinguish between learning context (decomposing the state/input space into regions) versus suggesting an action (Sect. 8.5). Some of the benefits of adopting a variable-sized team are that the evolved ensemble imparts additional knowledge about what was learnt. However, pathologies such as hitchhiking might result in bloated ensembles unless mitigation strategies are taken.

The future of EEL will likely continue to grow with the development of applications on the one hand and challenges to machine learning as a whole on the other. EEL is central to a body of work on feature construction that is now leading to the adoption of task transfer techniques, e.g. high-dimensional tasks with missing information and/or low cardinality. Likewise, EEL has repeatedly been successfully applied to streaming data in general, but the number of new streaming data applications continues to grow (e.g. IoT, social media, e-commerce). Streaming data applications

also point to the concept of lifelong (continuous) learning in which case there is potentially no end to the learning process.

EEL as formulated to address multi-agent or reinforcement learning problems is able to answer questions about hybrid homogeneous–heterogeneous team composition as well as variable-size ensembles. Incorporating hierarchical relationships into EEL means that participants who systematically mispredict can defer their decision to another (specialist) team that concentrates on resolving this ambiguity. Approaches for discovering graph ensembles provide a further opportunity for establishing structures that might be appropriate for continuous learning and interpretable solutions. A wide range of empirical results has already established that participants are significantly less complex than monolithic solutions (e.g. when using SVM or deep learning). Moreover, the appropriate selection of the ensemble aggregation operation (e.g. winner-tasks-all or the additive operator) provides explicit support for interpretable solutions [174]. This in combination with parsimonious participants may lead to truly scalable ensembles that support low-dimensional saliency, i.e. do not rely on post hoc ‘explanations’. In short, there is still ‘plenty of room’ in the divide-and-conquer approach to evolutionary machine learning.

Acknowledgements The author gratefully acknowledges support from the NSERC Discovery program (Canada).

References

1. Abdelbari, H., Shafi, K.: A genetic programming ensemble method for learning dynamical system models. In: Proceedings of the International Conference on Computer Modeling and Simulation, pp. 47–51. ACM (2017)
2. Agapitos, A., Loughran, R., Nicolau, M., Lucas, S.M., O’Neill, M., Brabazon, A.: A survey of statistical machine learning elements in genetic programming. *IEEE Trans. Evol. Comput.* **23**(6), 1029–1048 (2019)
3. Agapitos, A., O’Neill, M., Brabazon, A.: Ensemble bayesian model averaging in genetic programming. In: Proceedings of the IEEE Congress on Evolutionary Computation, pp. 2451–2458. IEEE (2014)
4. Agogino, A.K., Parker, C.H., Tumer, K.: Evolving distributed resource sharing for cubesat constellations. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1015–1022. ACM (2012)
5. Agogino, A.K., Tumer, K.: Evolving distributed agents for managing air traffic. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1888–1895. ACM (2007)
6. Agogino, A.K., Tumer, K.: Efficient evaluation functions for evolving coordination. *Evol. Comput.* **16**(2), 257–288 (2008)
7. Ain, Q.U., Al-Sahaf, H., Xue, B., Zhang, M.: A genetic programming approach to feature construction for ensemble learning in skin cancer detection. In: C.A.C. Coello (ed.) *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1186–1194. ACM (2020)
8. Al-Helali, B., Chen, Q., Xue, B., Zhang, M.: Multitree genetic programming with new operators for transfer learning in symbolic regression with incomplete data. *IEEE Trans. Evol. Comput.* **25**(6), 1049–1063 (2021)
9. Al-Sahaf, H., Al-Sahaf, A., Xue, B., Zhang, M.: Automatically evolving texture image descriptors using the multitree representation in genetic programming using few instances. *Evol. Comput.* **29**(3), 331–366 (2021)

10. Albrecht, S.V., Liemhetcharat, S., Stone, P.: Special issue on multi-agent interaction without prior coordination: guest editorial. *Autonomous Agents and Multi Agent Systems* **31**(4), 765–766 (2017)
11. Ali, S., Majid, A.: Can-evo-ens: Classifier stacking based evolutionary ensemble system for prediction of human breast cancer using amino acid sequences. *J. Biomed. Inform.* **54**, 256–269 (2015)
12. Altmann, A., Tolosi, L., Sander, O., Lengauer, T.: Permutation importance: a corrected feature importance measure. *Bioinformatics* **26**(10), 1340–1347 (2010)
13. Alvarez, I.M., Nguyen, T.B., Browne, W.N., Zhang, M.: A layered learning approach to scaling in learning classifier systems for boolean problems. *CoRR* **abs/2006.01415** (2020)
14. Amaral, R., Ianta, A., Bayer, C., Smith, R.J., Heywood, M.I.: Benchmarking genetic programming in a multi-action reinforcement learning locomotion task. In: Proceedings of the Genetic and Evolutionary Computation Conference. ACM (2022)
15. Andre, D., Teller, A.: Evolving team darwin united. In: RoboCup-98: Robot Soccer World Cup II, *LNCS*, vol. 1604, pp. 346–351. Springer (1998)
16. Angelie, P.J., Saunders, G.M., Pollack, J.B.: An evolutionary algorithm that constructs recurrent neural networks. *IEEE Trans. Neural Networks* **5**(1), 54–65 (1994)
17. Arnaldo, I., Veeramachaneni, K., O'Reilly, U.: Flash: A GP-GPU ensemble learning system for handling large datasets. In: Proceedings of the European Conference on Genetic Programming, *LNCS*, vol. 8599, pp. 13–24 (2014)
18. Arnaldo, I., Veeramachaneni, K., Song, A., O'Reilly, U.: Bring your own learner: A cloud-based, data-parallel commons for machine learning. *IEEE Comput. Intell. Mag.* **10**(1), 20–32 (2015)
19. Atwater, A., Heywood, M.I.: Benchmarking Pareto archiving heuristics in the presence of concept drift: diversity versus age. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 885–892. ACM (2013)
20. Axelrod, R.: Evolution of co-operation. Basic Books (1984)
21. Badran, K.M.S., Rockett, P.I.: Multi-class pattern classification using single, multi-dimensional feature-space feature extraction evolved by multi-objective genetic programming and its application to network intrusion detection. *Genet. Program Evolvable Mach.* **13**(1), 33–63 (2012)
22. Bahçeci, E., Miikkulainen, R.: Transfer of evolved pattern-based heuristics in games. In: Proceedings of the IEEE Symposium on Computational Intelligence and Games, pp. 220–227. IEEE (2008)
23. Bayer, C., Amaral, R., Smith, R.J., Ianta, A., Heywood, M.I.: Finding simple solutions to multi-task visual reinforcement learning problems with tangled program graphs. In: Genetic Programming Theory and Practice, vol. XVIII, pp. 1–19 (2022)
24. Belkin, M., Hsu, D., Ma, S., Mandal, S.: Reconciling modern machine learning and the bias-variance trade-off. *Proceedings of the National Academy of Science* **116**(32), 15849–15854 (2019)
25. Bhowan, U., Johnston, M., Zhang, M., Yao, X.: Evolving diverse ensembles using genetic programming for classification with unbalanced data. *IEEE Trans. Evol. Comput.* **17**(3), 368–386 (2013)
26. Bhowan, U., Johnston, M., Zhang, M., Yao, X.: Reusing genetic programming for ensemble selection in classification of unbalanced data. *IEEE Trans. Evol. Comput.* **18**(6), 893–908 (2014)
27. Bi, Y., Xue, B., Zhang, M.: A divide-and-conquer genetic programming algorithm with ensembles for image classification. *IEEE Trans. Evol. Comput.* **25**(6), 1148–1162 (2021)
28. Bi, Y., Xue, B., Zhang, M.: Learning and sharing: A multitask genetic programming approach to image feature learning. *IEEE Trans. Evol. Comput.* **26**(2), 218–232 (2022)
29. Bishop, C.M.: *Neural Networks for Pattern Recognition*. Oxford University Press (1995)
30. Bongard, J.C.: The legion system: A novel approach to evolving heterogeneity for collective problem solving. In: Proceedings of the European Conference on Genetic Programming, *LNCS*, vol. 1802, pp. 16–28. Springer (2000)

31. Brameier, M., Banzhaf, W.: Evolving teams of predictors with linear genetic programming. *Genet. Program Evolvable Mach.* **2**(4), 381–407 (2001)
32. Brave, S.: The evolution of memory and mental models using genetic programming. In: Proceedings of the Annual Conference on Genetic Programming. Morgan Kaufmann (1996)
33. Breiman, L.: Bagging predictors. *Mach. Learn.* **24**(2), 123–140 (1996)
34. Breiman, L.: Arcing classifier. *Annals of. Statistics* **26**(3), 801–849 (1998)
35. Cagnini, H.E.L., Freitas, A.A., Barros, R.C.: An evolutionary algorithm for learning interpretable ensembles of classifiers. In: Proceedings of the Brazilian Conference on Intelligent Systems, *Lecture Notes in Computer Science*, vol. 12319, pp. 18–33. Springer (2020)
36. Cardoso, R.P., Hart, E., Kurka, D.B., Pitt, J.: Augmenting novelty search with a surrogate model to engineer meta-diversity in ensembles of classifiers. In: Proceedings of the European Conference on Applications of Evolutionary Computation, *LNCS*, vol. 13224, pp. 418–434 (2022)
37. Cardoso, R.P., Hart, E., Kurka, D.B., Pitt, J.V.: Using novelty search to explicitly create diversity in ensembles of classifiers. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 849–857. ACM (2021)
38. Castelli, M., Gonçalves, I., Manzoni, L., Vanneschi, L.: Pruning techniques for mixed ensembles of genetic programming models. In: Proceedings of the European Conference on Genetic Programming, *LNCS*, vol. 10781, pp. 52–67 (2018)
39. Cava, W.G.L., Moore, J.H.: Learning feature spaces for regression with genetic programming. *Genet. Program Evolvable Mach.* **21**(3), 433–467 (2020)
40. Chandra, A., Chen, H., Yao, X.: Trade-off between diversity and accuracy in ensemble generation. In: Y. Jin (ed.) *Multi-Objective Machine Learning, Studies in Computational Intelligence*, vol. 16, pp. 429–464. Springer (2006)
41. Chandra, A., Yao, X.: Evolving hybrid ensembles of learning machines for better generalization. *Neurocomputing* **69**, 686–700 (2006)
42. Chen, H., Yao, X.: Multiobjective neural network ensembles based on regularized negative correlation learning. *IEEE Trans. Knowl. Data Eng.* **22**(12), 1738–1751 (2010)
43. Coello, C.A.C.: Evolutionary multi-objective optimization: a historical view of the field. *IEEE Comput. Intell. Mag.* **1**(1), 28–36 (2006)
44. Colby, M.K., Tumer, K.: Shaping fitness functions for coevolving cooperative multiagent systems. In: International Conference on Autonomous Agents and Multiagent Systems, pp. 425–432. IFAAMAS (2012)
45. D'Ambrosio, D.B., Lehman, J., Risi, S., Stanley, K.O.: Evolving policy geometry for scalable multiagent learning. In: International Conference on Autonomous Agents and Multiagent Systems, pp. 731–738. IFAAMAS (2010)
46. D'Ambrosio, D.B., Stanley, K.O.: Scalable multiagent learning through indirect encoding of policy geometry. *Evol. Intel.* **6**(1), 1–26 (2013)
47. Defoin-Platel, M., Chami, M., Clergue, M., Collard, P.: Teams of genetic predictors for inverse problem solving. In: Proceedings of the European Conference on Genetic Programming, *LNCS*, vol. 3447, pp. 341–350 (2005)
48. Desnos, K., Sourbier, N., Raumer, P., Gesny, O., Pelcat, M.: Gegelati: Lightweight artificial intelligence through generic and evolvable tangled program graphs. In: Workshop on Design and Architectures for Signal and Image Processing, pp. 35–43. ACM (2021)
49. Dick, G.: Sensitivity-like analysis for feature selection in genetic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 401–408. ACM (2017)
50. Dick, G., Owen, C.A., Whigham, P.A.: Evolving bagging ensembles using a spatially-structured niching method. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 418–425. ACM (2018)
51. Dixit, G., Zerbel, N., Tumer, K.: Dirichlet-multinomial counterfactual rewards for heterogeneous multiagent systems. In: IEEE International Symposium on Multi-Robot and Multi-Agent Systems, pp. 209–215. IEEE (2019)
52. Doherty, D., O'Riordan, C.: Effects of shared perception on the evolution of squad behaviors. *IEEE Transactions on Computational Intelligence and AI in Games* **1**(1), 50–62 (2009)

53. Doncieux, S., Bredèche, N., Mouret, J., Eiben, A.E.: Evolutionary robotics: What, why, and where to. *Frontiers Robotics AI* **2**, 4 (2015)
54. Doncieux, S., Mouret, J.: Behavioral diversity with multiple behavioral distances. In: Proceedings of the IEEE Congress on Evolutionary Computation, pp. 1427–1434. IEEE (2013)
55. Doucette, J.A., Lichodzijewski, P., Heywood, M.I.: Hierarchical task decomposition through symbiosis in reinforcement learning. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 97–104. ACM (2012)
56. Doucette, J.A., McIntyre, A.R., Lichodzijewski, P., Heywood, M.I.: Symbiotic coevolutionary genetic programming: a benchmarking study under large attribute spaces. *Genet. Program Evolvable Mach.* **13**(1), 71–101 (2012)
57. Du, M., Liu, N., Hu, X.: Techniques for interpretable machine learning. *Commun. ACM* **63**(1), 68–77 (2020)
58. Durasevic, M., Jakobovic, D.: Comparison of ensemble learning methods for creating ensembles of dispatching rules for the unrelated machines environment. *Genet. Program Evolvable Mach.* **19**(1–2), 53–92 (2018)
59. Durasevic, M., Jakobovic, D.: Creating dispatching rules by simple ensemble combination. *J. Heuristics* **25**(6), 959–1013 (2019)
60. Durasevic, M., Planinic, L., Gala, F.J.G., Jakobovic, D.: Novel ensemble collaboration method for dynamic scheduling problems. *CoRR* **abs/2203.14290** (2022)
61. Fahlman, S.E., Lebiere, C.: The cascade-correlation learning architecture. In: Advances in Neural Information Processing Systems, vol. 2, pp. 524–532. Morgan Kaufmann (1989)
62. Feldt, R.: Generating diverse software versions with genetic programming: and experimental study. *IEE Proceedings-Software* **145**(6), 228–236 (1998)
63. Fletcher, S., Verma, B.K., Zhang, M.: A non-specialized ensemble classifier using multi-objective optimization. *Neurocomputing* **409**, 93–102 (2020)
64. Folino, G., Guarascio, M., Papuzzo, G.: Exploiting fractal dimension and a distributed evolutionary approach to classify data streams with concept drifts. *Appl. Soft Comput.* **75**, 284–297 (2019)
65. Folino, G., Papuzzo, G.: Handling different categories of concept drifts in data streams using distributed GP. In: Proceedings of the European Conference on Genetic Programming, *LNCS*, vol. 6021, pp. 74–85. Springer (2010)
66. Folino, G., Pisani, F.S., Pontieri, L.: A gp-based ensemble classification framework for time-changing streams of intrusion detection data. *Soft. Comput.* **24**(23), 17541–17560 (2020)
67. Folino, G., Pizzuti, C., Spezzano, G.: Mining distributed evolving data streams using fractal GP ensembles. In: Proceedings of the European Conference on Genetic Programming, *LNCS*, vol. 4445, pp. 160–169. Springer (2007)
68. Folino, G., Pizzuti, C., Spezzano, G.: Training distributed GP ensemble with a selective algorithm based on clustering and pruning for pattern classification. *IEEE Trans. Evol. Comput.* **12**(4), 458–468 (2008)
69. Gagné, C., Sebag, M., Schoenauer, M., Tomassini, M.: Ensemble learning for free with evolutionary algorithms? In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1782–1789. ACM (2007)
70. Gama, J., Brazdil, P.: Cascade generalization. *Mach. Learn.* **41**(3), 315–343 (2000)
71. García-Pedrajas, N., del Castillo, J.A.R., Ortiz-Boyer, D.: A cooperative coevolutionary algorithm for instance selection for instance-based learning. *Mach. Learn.* **78**(3), 381–420 (2010)
72. Gathercole, C., Ross, P.: Dynamic training subset selection for supervised learning in genetic programming. In: Proceedings of the Parallel Problem Solving from Nature Conference, *LNCS*, vol. 866, pp. 312–321. Springer (1994)
73. Geman, S., Bienenstock, E., Doursat, R.: Neural networks and the bias/variance dilemma. *Neural Comput.* **4**(1), 1–58 (1992)
74. Gomes, J.C., Mariano, P., Christensen, A.L.: Novelty-driven cooperative coevolution. *Evol. Comput.* **25**(2), 275–307 (2017)
75. Gomes, J.C., Mariano, P., Christensen, A.L.: Dynamic team heterogeneity in cooperative coevolutionary algorithms. *IEEE Trans. Evol. Comput.* **22**(6), 934–948 (2018)

76. Gomez, F.J., Schmidhuber, J., Miikkulainen, R.: Accelerated neural evolution through cooperatively coevolved synapses. *J. Mach. Learn. Res.* **9**, 937–965 (2008)
77. Guyon, I., Nikravesh, M., Gunn, S.R., Zadeh, L.A. (eds.): Feature Extraction - Foundations and Applications, *Studies in Fuzziness and Soft Computing*, vol. 207. Springer (2006)
78. Hansen, L.K., Salamon, P.: Neural network ensembles. *IEEE Trans. Pattern Anal. Mach. Intell.* **12**(10), 993–1001 (1990)
79. Hara, A., Nagao, T.: Emergence of the cooperative behavior using adg; automatically defined groups. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1039–1046. Morgan Kaufmann (1999)
80. Hart, E., Sim, K.: On constructing ensembles for combinatorial optimisation. *Evolutionary Computation* **26**(1) (2018)
81. Haynes, T., Sen, S., Schoenfeld, D.A., Wainwright, R.L.: Evolving a team. In: AAAI Fall Symposium on Genetic Programming, pp. 23–30. AAAI Press (1995)
82. Helmuth, T., Spector, L., Matheson, J.: Solving uncompromising problems with lexicase selection. *IEEE Trans. Evol. Comput.* **19**(5), 630–643 (2015)
83. Heywood, M.I.: Evolutionary model building under streaming data for classification tasks: opportunities and challenges. *Genet. Program Evolvable Mach.* **16**(3), 283–326 (2015)
84. Heywood, M.I., Lichodzijewski, P.: Symbiogenesis as a mechanism for building complex adaptive systems: A review. In: Proceedings of the Applications of Evolutionary Computation Conference—Part I, *LNCS*, vol. 6024, pp. 51–60. Springer (2010)
85. Hong, J., Cho, S.: The classification of cancer based on DNA microarray data that uses diverse ensemble genetic programming. *Artif. Intell. Med.* **36**(1), 43–58 (2006)
86. Howley, E., O’Riordan, C.: The emergence of cooperation among agents using simple fixed bias tagging. In: Proceedings of the IEEE Congress on Evolutionary Computation, pp. 1011–1016. IEEE (2005)
87. Iba, H.: Emergent cooperation for multiple agents using genetic programming. In: Proceedings of the International Conference on Parallel Problem Solving from Nature, *LNCS*, vol. 1141, pp. 32–41. Springer (1996)
88. Iba, H.: Bagging, boosting, and bloating in genetic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1053–1060. Morgan Kaufmann (1999)
89. Ibarra-Vázquez, G., Olague, G., Chan-Ley, M., Puente, C., Souberbielle-Montalvo, C.: Brain programming is immune to adversarial attacks: Towards accurate and robust image classification using symbolic learning. *Swarm Evol. Comput.* **71**, 101059 (2022)
90. Imamura, K., Soule, T., Heckendorf, R.B., Foster, J.A.: Behavioral diversity and a probabilistically optimal GP ensemble. *Genet. Program Evolvable Mach.* **4**(3), 235–253 (2003)
91. Iqbal, M., Browne, W.N., Zhang, M.: Reusing building blocks of extracted knowledge to solve complex, large-scale boolean problems. *IEEE Trans. Evol. Comput.* **18**(4), 465–480 (2014)
92. Iqbal, M., Browne, W.N., Zhang, M.: Extending xcs with cyclic graphs for scalability on complex boolean problems. *Evol. Comput.* **25**(2), 173–204 (2017)
93. Ivert, A., de Castro Aranha, C., Iba, H.: Feature selection and classification using ensembles of genetic programs and within-class and between-class permutations. In: Proceedings of the IEEE Congress on Evolutionary Computation, pp. 1121–1128. IEEE (2015)
94. Johansson, U., Löfström, T., König, R., Niklasson, L.: Building neural network ensembles using genetic programming. In: Proceedings of the International Joint Conference on Neural Networks, pp. 1260–1265. IEEE (2006)
95. Johansson, U., Sönströd, C., Löfström, T., König, R.: Using genetic programming to obtain implicit diversity. In: Proceedings of the IEEE Congress on Evolutionary Computation, pp. 2454–2459. IEEE (2009)
96. Jordan, M.I., Jacobs, R.A.: Hierarchical mixtures of experts and the EM algorithm. *Neural Comput.* **6**(2), 181–214 (1994)
97. Kantschik, W., Banzhaf, W.: Linear-graph GP - A new GP structure. In: Proceedings of the European Conference on Genetic Programming, *LNCS*, vol. 2278, pp. 83–92. Springer (2002)

98. Karunakaran, D., Mei, Y., Zhang, M.: Multitasking genetic programming for stochastic team orienteering problem with time windows. In: IEEE Symposium Series on Computational Intelligence, pp. 1598–1605. IEEE (2019)
99. Keijzer, M., Babovic, V.: Genetic programming, ensemble methods and the bias/variance tradeoff – introductory investigations. In: Proceedings of the European Conference on Genetic Programming, *LNCS*, vol. 1802, pp. 76–90 (2000)
100. Keijzer, M., Babovic, V.: Declarative and preferential bias in gp-based scientific discovery. *Genet. Program Evolvable Mach.* **3**(1), 41–79 (2002)
101. Kelly, S., Heywood, M.I.: Knowledge transfer from keepaway soccer to half-field offense through program symbiosis: Building simple programs for a complex task. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1143–1150. ACM (2015)
102. Kelly, S., Heywood, M.I.: Emergent tangled graph representations for atari game playing agents. In: Proceedings of the European Conference on Genetic Programming, *LNCS*, vol. 10196, pp. 64–79 (2017)
103. Kelly, S., Heywood, M.I.: Discovering agent behaviors through code reuse: Examples from half-field offense and ms. pac-man. *IEEE Transactions on Games* **10**(2), 195–208 (2018)
104. Kelly, S., Heywood, M.I.: Emergent solutions to high-dimensional multitask reinforcement learning. *Evolutionary Computation* **26**(3) (2018)
105. Kelly, S., Lichodziewski, P., Heywood, M.I.: On run time libraries and hierarchical symbiosis. In: Proceedings of the IEEE Congress on Evolutionary Computation, pp. 1–8. IEEE (2012)
106. Kelly, S., Newsted, J., Banzhaf, W., Gondro, C.: A modular memory framework for time series prediction. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 949–957. ACM (2020)
107. Kelly, S., Smith, R.J., Heywood, M.I.: Emergent policy discovery for visual reinforcement learning through tangled program graphs: A tutorial. In: W. Banzhaf, L. Spector, L. Sheneman (eds.) *Genetic Programming Theory and Practice XVI*, Genetic and Evolutionary Computation, pp. 37–57. Springer (2018)
108. Kelly, S., Smith, R.J., Heywood, M.I., Banzhaf, W.: Emergent tangled program graphs in partially observable recursive forecasting and vizdoom navigation tasks. *ACM Transactions on Evolutionary Learning and Optimization* **1**(3), 11:1–11:41 (2021)
109. Kelly, S., Voegler, T., Banzhaf, W., Gondro, C.: Evolving hierarchical memory-prediction machines in multi-task reinforcement learning. *Genet. Program Evolvable Mach.* **22**(4), 573–605 (2021)
110. Khadka, S., Majumdar, S., Miret, S., McAleer, S., Turner, K.: Evolutionary reinforcement learning for sample-efficient multiagent coordination. In: Proceedings of the International Conference on Machine Learning, *Proceedings of Machine Learning Research*, vol. 119, pp. 6651–6660. PMLR (2020)
111. Khanchi, S., Vahdat, A., Heywood, M.I., Zincir-Heywood, A.N.: On botnet detection with genetic programming under streaming data label budgets and class imbalance. *Swarm Evol. Comput.* **39**, 123–140 (2018)
112. Kim, K., Cho, S.: Systematically incorporating domain-specific knowledge into evolutionary speciated checkers players. *IEEE Trans. Evol. Comput.* **9**(6), 615–627 (2005)
113. Kim, K., Cho, S.: An evolutionary algorithm approach to optimal ensemble classifiers for DNA microarray data analysis. *IEEE Trans. Evol. Comput.* **12**(3), 377–388 (2008)
114. Kirkpatrick, J., Pascanu, R., Rabinowitz, N.C., Veness, J., Desjardins, G., Rusu, A.A., Milan, K., Quan, J., Ramalho, T., Grabska-Barwinska, A., Hassabis, D., Clopath, C., Kumaran, D., Hadsell, R.: Overcoming catastrophic forgetting in neural networks. *Proc. Natl. Acad. Sci.* **114**(13), 3521–3526 (2017)
115. Koza, J.R.: *Genetic Programming II: Automatic discovery of reusable programs*. MIT Press (1994)
116. Krawiec, K.: Genetic programming-based construction of features for machine learning and knowledge discovery tasks. *Genet. Program Evolvable Mach.* **3**(4), 329–343 (2002)
117. Krawiec, K., Heywood, M.I.: Solving complex problems with coevolutionary algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference (Tutorial Program), pp. 832–858. ACM (2020)

118. Krogh, A., Vedelsby, J.: Neural network ensembles, cross validation, and active learning. In: Advances in Neural Information Processing Systems, vol. 7, pp. 231–238. MIT Press (1994)
119. Kuncheva, L.I.: Combining Pattern Classifiers: Methods and Algorithms. Wiley (2004)
120. Kuncheva, L.I., Whitaker, C.J.: Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy. *Mach. Learn.* **51**(2), 181–207 (2003)
121. Lacy, S.E., Lones, M.A., Smith, S.L.: A comparison of evolved linear and non-linear ensemble vote aggregators. In: Proceedings of the IEEE Congress on Evolutionary Computation, pp. 758–763. IEEE (2015)
122. Lalejini, A., Ofria, C.: Evolving event-driven programs with signalgp. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1135–1142. ACM (2018)
123. Levesque, J., Durand, A., Gagné, C., Sabourin, R.: Multi-objective evolutionary optimization for generating ensembles of classifiers in the ROC space. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 879–886. ACM (2012)
124. Lichocki, P., Wischmann, S., Keller, L., Floreano, D.: Evolving team compositions by agent swapping. *IEEE Trans. Evol. Comput.* **17**(2), 282–298 (2013)
125. Lichodzijewski, P., Heywood, M.I.: Pareto-coevolutionary genetic programming for problem decomposition in multi-class classification. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 464–471. ACM (2007)
126. Lichodzijewski, P., Heywood, M.I.: Managing team-based problem solving with symbiotic bid-based genetic programming. In: C. Ryan, M. Keijzer (eds.) Proceedings of the Genetic and Evolutionary Computation Conference, pp. 363–370. ACM (2008)
127. Lichodzijewski, P., Heywood, M.I.: Symbiosis, complexification and simplicity under GP. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 853–860. ACM (2010)
128. Liu, Y., Yao, X., Higuchi, T.: Evolutionary ensembles with negative correlation learning. *IEEE Trans. Evol. Comput.* **4**(4), 380–387 (2000)
129. Löfström, T., Johansson, U., Boström, H.: Ensemble member selection using multi-objective optimization. In: Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, pp. 245–251. IEEE (2009)
130. Loginov, A., Heywood, M.I.: On the impact of streaming interface heuristics on GP trading agents: an FX benchmarking study. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1341–1348. ACM (2013)
131. Loginov, A., Heywood, M.I., Wilson, G.C.: Benchmarking a coevolutionary streaming classifier under the individual household electric power consumption dataset. In: International Joint Conference on Neural Networks, pp. 2834–2841. IEEE (2016)
132. Lu, Z., Wu, X., Bongard, J.C.: Active learning through adaptive heterogeneous ensembling. *IEEE Trans. Knowl. Data Eng.* **27**(2), 368–381 (2015)
133. Luke, S., Hohn, C., Farris, J., Jackson, G., Handler, J.A.: Co-evolving soccer softbot team coordination with genetic programming. In: RoboCup-97: Robot Soccer World Cup I, LNCS, vol. 1395, pp. 398–411. Springer (1997)
134. Luke, S., Spector, L.: Evolving teamwork and coordination with genetic programming. In: Proceedings of the Annual Conference on Genetic Programming, pp. 150–156. MIT Press (1996)
135. Mabu, S., Hirasawa, K., Hu, J.: A graph-based evolutionary algorithm: Genetic network programming (GNP) and its extension using reinforcement learning. *Evol. Comput.* **15**(3), 369–398 (2007)
136. Mabu, S., Hirasawa, K., Obayashi, M., Kuremoto, T.: Enhanced decision making mechanism of rule-based genetic network programming for creating stock trading signals. *Expert Syst. Appl.* **40**(16), 6311–6320 (2013)
137. Machado, M.C., Bellemare, M.G., Talvitie, E., Veness, J., Hausknecht, M.J., Bowling, M.: Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents. *Journal of Artificial Intelligence Research* **61**, 523–562 (2018)
138. Maynard-Smith, J., Haigh, J.: The hitch-hiking effect of a favourable gene. *Genet. Res.* **23**(1), 23–35 (1974)

139. McIntyre, A.R., Heywood, M.I.: Classification as clustering: A pareto cooperative-competitive GP approach. *Evol. Comput.* **19**(1), 137–166 (2011)
140. Mingo, J.M., Aler, R.: Evolution of shared grammars for describing simulated spatial scenes with grammatical evolution. *Genetic Programming and Evolvable Machine* **19**(1–2), 235–270 (2018)
141. Mitchell, M., Forrest, S., Holland, J.H.: The Royal Road for genetic algorithms: Fitness landscapes and GA performance. In: *Proceedings of the European Conference on Artificial Life*, pp. 245–254. MIT Press (1992)
142. Moriarty, D.E., Miikkulainen, R.: Efficient reinforcement learning through symbiotic evolution. *Mach. Learn.* **22**(1–3), 11–32 (1996)
143. Moshaiov, A., Tal, A.: Family bootstrapping: A genetic transfer learning approach for onsetting the evolution for a set of related robotic tasks. In: *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 2801–2808. IEEE (2014)
144. Mouret, J., Doncieux, S.: Overcoming the bootstrap problem in evolutionary robotics using behavioral diversity. In: *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 1161–1168. IEEE (2009)
145. Mouret, J., Doncieux, S.: Encouraging behavioral diversity in evolutionary robotics: An empirical study. *Evol. Comput.* **20**(1), 91–133 (2012)
146. Moyano, J.M., Ventura, S.: Auto-adaptive grammar-guided genetic programming algorithm to build ensembles of multi-label classifiers. *Inf. Fusion* **78**, 1–19 (2022)
147. Muni, D.P., Pal, N.R., Das, J.: A novel approach to design classifiers using genetic programming. *IEEE Trans. Evol. Comput.* **8**(2), 183–196 (2004)
148. Muni, D.P., Pal, N.R., Das, J.: Genetic programming for simultaneous feature selection and classifier design. *IEEE Transactions on Systems, Man, and Cybernetics - Part B* **36**(1), 106–117 (2006)
149. Muñoz, L., Trujillo, L., Silva, S.: Transfer learning in constructive induction with genetic programming. *Genet. Program Evolvable Mach.* **21**(4), 529–569 (2020)
150. Nguyen, M.H., Abbass, H.A., McKay, R.I.: A novel mixture of experts model based on cooperative coevolution. *Neurocomputing* **70**(1–3), 155–163 (2006)
151. Nguyen, M.H., Abbass, H.A., McKay, R.I.: Analysis of CCME: coevolutionary dynamics, automatic problem decomposition, and regularization. *IEEE Transactions on Systems, Man, and Cybernetics-Part C* **38**(1), 100–109 (2008)
152. Nitschke, G., Didi, S.: Evolutionary policy transfer and search methods for boosting behavior quality: Robocup keep-away case study. *Frontiers Robotics AI* **4**, 62 (2017)
153. Nitschke, G.S., Schut, M.C., Eiben, A.E.: Collective neuro-evolution for evolving specialized sensor resolutions in a multi-rover task. *Evol. Intel.* **3**(1), 13–29 (2010)
154. Nitschke, G.S., Schut, M.C., Eiben, A.E.: Evolving behavioral specialization in robot teams to solve a collective construction task. *Swarm Evol. Comput.* **2**, 25–38 (2012)
155. O'Neill, M., Ryan, C.: Grammatical evolution by grammatical evolution: The evolution of grammar and genetic code. In: *Proceedings of the European Conference on Genetic Programming, LNCS*, vol. 3003, pp. 138–149 (2004)
156. Opitz, D.W., Maclin, R.: Popular ensemble methods: An empirical study. *Journal of Artificial Intelligence Research* **11**, 169–198 (1999)
157. O'Reilly, U., Toutouh, J., Pertierra, M.A., Sanchez, D.P., Garcia, D., Lugo, A.E., Kelly, J., Hemberg, E.: Adversarial genetic programming for cyber security: a rising application domain where GP matters. *Genet. Program Evolvable Mach.* **21**(1–2), 219–250 (2020)
158. Owen, C.A., Dick, G., Whigham, P.A.: Characterizing genetic programming error through extended bias and variance decomposition. *IEEE Trans. Evol. Comput.* **24**(6), 1164–1176 (2020)
159. Panait, L., Luke, S., Wiegand, R.P.: Biasing coevolutionary search for optimal multiagent behaviors. *IEEE Trans. Evol. Comput.* **10**(6), 629–645 (2006)
160. Panait, L., Sullivan, K., Luke, S.: Lenience towards teammates helps in cooperative multiagent learning. *Tech. Rep. GMU-CS-TR-2013-2*, George Mason University (2013)

161. Pardoe, D., Ryoo, M.S., Miikkulainen, R.: Evolving neural network ensembles for control problems. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1379–1384. ACM (2005)
162. Paris, G., Robilliard, D., Fonlupt, C.: Applying boosting techniques to genetic programming. In: International Conference on Artificial Evolution, *LNCS*, vol. 2310, pp. 267–280 (2001)
163. Park, J., Nguyen, S., Zhang, M., Johnston, M.: Evolving ensembles of dispatching rules using genetic programming for job shop scheduling. In: Proceedings of the European Conference on Genetic Programming, *LNCS*, vol. 9025, pp. 92–104 (2015)
164. Parter, M., Kashtan, N., Alon, U.: Facilitated variation: How evolution learns for past environments to generalize to new environments. *PLoS Computational Biology* **4**(11), e1000206:1–16 (2008)
165. Portman, B., Heywood, M.I.: On the interaction between lexicase selection, modularity and data subsets. In: Proceedings of the Genetic and Evolutionary Computation Conference (Companion), pp. 586–589. ACM (2022)
166. Potter, M.A., Jong, K.A.D.: Cooperative coevolution: An architecture for evolving coadapted subcomponents. *Evol. Comput.* **8**(1), 1–29 (2000)
167. Rebuli, K.B., Vanneschi, L.: Progressive insular cooperative GP. In: T. Hu, N. Lourenço, E. Medvet (eds.) Proceedings of the European Conference on Genetic Programming, *LNCS*, vol. 12691, pp. 19–35 (2021)
168. Reynolds, C.W.: An evolved, vision-based behavioral model of coordinated group motion. In: Proceedings of the International Conference on Simulation of Adaptive Behavior, pp. 384–392. MIT Press (1993)
169. Riolo, R.L.: The effects and evolution of tag-mediated selection of partners in populations playing the iterated prisoner’s dilemma. In: Proceedings of the International Conference on Genetic Algorithms, pp. 378–385. Morgan Kaufmann (1997)
170. Rodrigues, N.M., Batista, J.E., Silva, S.: Ensemble genetic programming. In: Proceedings of the European Conference on Genetic Programming, *LNCS*, vol. 12101, pp. 151–166 (2020)
171. Rodriguez-Coayahuitl, L., Morales-Reyes, A., Escalante, H.J., Coello, C.A.C.: Cooperative co-evolutionary genetic programming for high dimensional problems. In: Proceedings of the Parallel Problem Solving from Nature Conference–Part II, *LNCS*, vol. 12270, pp. 48–62 (2020)
172. Rosin, C.D., Belew, R.K.: New methods for competitive coevolution. *Evol. Comput.* **5**(1), 1–29 (1997)
173. Rubini, J., Heckendorf, R.B., Soule, T.: Evolution of team composition in multi-agent systems. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1067–1074. ACM (2009)
174. Rudin, C.: Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence* **1**(5), 206–215 (2019)
175. Sachdeva, E., Khadka, S., Majumdar, S., Tumar, K.: MAEDyS: multiagent evolution via dynamic skill selection. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 163–171. ACM (2021)
176. dos Santos, E.M., Sabourin, R., Maupin, P.: Overfitting cautious selection of classifier ensembles with genetic algorithms. *Information Fusion* **10**(2), 150–162 (2009)
177. Schapire, R.E.: The strength of weak learnability. *Mach. Learn.* **5**, 197–227 (1990)
178. da Silva, F.L., Warnell, G., Costa, A.H.R., Stone, P.: Agents teaching agents: a survey on inter-agent transfer learning. *Autonomous Agents and Multi Agent Systems* **34**(1), 9 (2020)
179. Silver, D.L., Yang, Q., Li, L.: Lifelong machine learning systems: Beyond learning algorithms. In: Papers from the Spring Symposium, *AAAI Technical Report*, vol. SS-13-05. AAAI (2013)
180. Sipper, M.: Classy ensemble: A novel ensemble algorithm for classification. *CoRR* [abs/2302.10580](https://arxiv.org/abs/2302.10580) (2023)
181. Sipper, M., Moore, J.H.: Symbolic-regression boosting. *Genet. Program Evolvable Mach.* **22**(3), 357–381 (2021)
182. Sipper, M., Moore, J.H.: AddGBoost: A gradient boosting-style algorithm based on strong learners. *Machine Learning with Applications* **7**(100243) (2022)

183. Smith, M.G., Bull, L.: Genetic programming with a genetic algorithm for feature construction and selection. *Genetic Programming Evolvable Machines* **6**(3), 265–281 (2005)
184. Smith, R.E., Forrest, S., Perelson, A.S.: Searching for diverse, cooperative populations with genetic algorithms. *Evol. Comput.* **1**(2), 127–149 (1993)
185. Smith, R.J., Amaral, R., Heywood, M.I.: Evolving simple solutions to the CIFAR-10 benchmark using tangled program graphs. In: Proceedings of the IEEE Congress on Evolutionary Computation, pp. 2061–2068. IEEE (2021)
186. Smith, R.J., Heywood, M.I.: Coevolving deep hierarchies of programs to solve complex tasks. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1009–1016. ACM (2017)
187. Smith, R.J., Heywood, M.I.: Scaling tangled program graphs to visual reinforcement learning in vizdoom. In: Proceedings of the European Conference on Genetic Programming, *LNCS*, vol. 10781, pp. 135–150 (2018)
188. Smith, R.J., Heywood, M.I.: Evolving Dota 2 shadow fiend bots using genetic programming with external memory. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 179–187. ACM (2019)
189. Smith, R.J., Heywood, M.I.: A model of external memory for navigation in partially observable visual reinforcement learning tasks. In: Proceedings of the European Conference on Genetic Programming, *LNCS*, vol. 11451, pp. 162–177 (2019)
190. Smith, R.J., Kelly, S., Heywood, M.I.: Discovering rubik's cube subgroups using coevolutionary GP: A five twist experiment. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 789–796. ACM (2016)
191. Sohn, J., Yoo, S.: Why train-and-select when you can use them all?: ensemble model for fault localisation. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1408–1416. ACM (2019)
192. Song, D., Heywood, M.I., Zincir-Heywood, A.N.: Training genetic programming on half a million patterns: an example from anomaly detection. *IEEE Trans. Evol. Comput.* **9**(3), 225–239 (2005)
193. Sotto, L.F.D.P., Kaufmann, P., Atkinson, T., Kalkreuth, R., Basgalupp, M.P.: Graph representations in genetic programming. *Genet. Program Evolvable Mach.* **22**(4), 607–636 (2021)
194. Soule, T.: Voting teams: a cooperative approach to non-typical problems using genetic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 916–922. Morgan Kaufmann (1999)
195. Soule, T.: Cooperative evolution on the intertwined spirals problem. In: Proceedings of the European Conference on Genetic Programming, *LNCS*, vol. 2610, pp. 434–442. Springer (2003)
196. Sourbier, N., Desnos, K., Guyet, T., Majorczyk, F., Gesny, O., Pelcat, M.: SECURE-GEGELATI always-on intrusion detection through GEGELATI lightweight tangled program graphs. *Journal of Signal Processing Systems* **94**(7), 753–770 (2022)
197. Stanley, K.O., Bryant, B.D., Miikkulainen, R.: Real-time neuroevolution in the NERO video game. *IEEE Trans. Evol. Comput.* **9**(6), 653–668 (2005)
198. Stefano, C.D., Fontanella, F., Folino, G., di Freca, A.S.: A bayesian approach for combining ensembles of GP classifiers. In: Proceedings of the International Workshop on Multiple Classifier Systems, *LNCS*, vol. 6713, pp. 26–35. Springer (2011)
199. Stone, P.: Layered learning in multiagent systems - a winning approach to robotic soccer. MIT Press, Intelligent robotics and autonomous agents (2000)
200. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An introduction, 2nd edn. MIT Press (2018)
201. Tackett, W.A., Carmi, A.: The donut problem: Scalability and generalization in genetic programming. In: K.E. Kinnear (ed.) *Advances in Genetic Programming*, pp. 143–176. MIT Press (1994)
202. Taylor, M.E., Stone, P.: An introduction to intertask transfer for reinforcement learning. *AI Magine* **32**(1), 15–34 (2011)

203. Teller, A., Veloso, M.M.: A controlled experiment: Evolution for learning difficult image classification. In: Proceedings of the Portuguese Conference on Progress in Artificial Intelligence, *LNCS*, vol. 990, pp. 165–176. Springer (1995)
204. Thomason, R., Heckendorf, R.B., Soule, T.: Training time and team composition robustness in evolved multi-agent systems. In: Proceedings of the European Conference on Genetic Programming, *LNCS*, vol. 4971, pp. 1–12 (2008)
205. Thomason, R., Soule, T.: Novel ways of improving cooperation and performance in ensemble classifiers. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1708–1715. ACM (2007)
206. Tran, B., Xue, B., Zhang, M.: Genetic programming for multiple-feature construction on high-dimensional classification. *Pattern Recogn.* **93**, 404–417 (2019)
207. Trianni, V., Lopez-Ibanez, M.: Advantages of task-specific multi-objective optimization in evolutionary robotics. *PLOS one* **10**(10), e0140056:1–27 (2015)
208. Tsakonas, A., Gabrys, B.: GRADIENT: grammar-driven genetic programming framework for building multi-component, hierarchical predictive systems. *Expert Syst. Appl.* **39**(18), 13253–13266 (2012)
209. Turner, A.J., Miller, J.F.: Neuroevolution: Evolving heterogeneous artificial neural networks. *Evol. Intel.* **7**(3), 135–154 (2014)
210. Vahdat, A., Morgan, J., McIntyre, A.R., Heywood, M.I., Zincir-Heywood, A.N.: Evolving GP classifiers for streaming data tasks with concept change and label budgets: A benchmarking study. In: A.H. Gandomi, A.H. Alavi, C. Ryan (eds.) *Handbook of Genetic Programming Applications*, pp. 451–480. Springer (2015)
211. Vahdat, A., Morgan, J., McIntyre, A.R., Heywood, M.I., Zincir-Heywood, A.N.: Tapped delay lines for GP streaming data classification with label budgets. In: Proceedings of the European Conference Genetic Programming, *LNCS*, vol. 9025, pp. 126–138. Springer (2015)
212. Veeramachaneni, K., Arnaldo, I., Derby, O., O'Reilly, U.: Flexgp - cloud-based ensemble learning with genetic programming for large regression problems. *Journal of Grid Computing* **13**(3), 391–407 (2015)
213. Verbancsics, P., Stanley, K.O.: Evolving static representations for task transfer. *J. Mach. Learn. Res.* **11**, 1737–1769 (2010)
214. Virgolin, M.: Genetic programming is naturally suited to evolve bagging ensembles. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 830–839. ACM (2021)
215. Waibel, M., Keller, L., Floreano, D.: Genetic team composition and level of selection in the evolution of cooperation. *IEEE Trans. Evol. Comput.* **13**(3), 648–660 (2009)
216. Wang, S., Mei, Y., Zhang, M.: Novel ensemble genetic programming hyper-heuristics for uncertain capacitated arc routing problem. In: A. Auger, T. Stützle (eds.) *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1093–1101. ACM (2019)
217. Wen, Y., Ting, C.: Learning ensemble of decision trees through multifactorial genetic programming. In: *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 5293–5300. IEEE (2016)
218. Whiteson, S.: Adaptive representations for reinforcement learning, *Studies in Computational Intelligence*, vol. 291. Springer (2010)
219. Whiteson, S., Kohl, N., Miikkulainen, R., Stone, P.: Evolving soccer keepaway players through task decomposition. *Mach. Learn.* **59**(1–2), 5–30 (2005)
220. Wiegand, R.P., Liles, W.C., Jong, K.A.D.: An empirical analysis of collaboration methods in cooperative coevolutionary algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1235–1245. Morgan Kaufmann (2001)
221. Wolpert, D.H.: Stacked generalization. *Neural Netw.* **5**(2), 241–259 (1992)
222. Worzel, W.P., Yu, J., Almal, A., Chinnaian, A.: Applications of genetic programming in cancer research. *The International Journal of Biochemistry & Cell Biology* **41**, 405–413 (2009)
223. Wu, S.X., Banzhaf, W.: Rethinking multilevel selection in genetic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1403–1410. ACM (2011)

224. Yates, C., Christopher, R., Tumar, K.: Multi-fitness learning for behaviour-driven cooperation. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 453–461. ACM (2020)
225. Yong, C.H., Miikkulainen, R.: Coevolution of role-based cooperation in multiagent systems. *IEEE Trans. Auton. Ment. Dev.* **1**(3), 170–186 (2009)
226. Yüksel, S.E., Wilson, J.N., Gader, P.D.: Twenty years of mixture of experts. *IEEE Transactions on Neural Networks and Learning Systems* **23**(8), 1177–1193 (2012)
227. Zhou, A., Qu, B., Li, H., Zhao, S., Suganthan, P.N., Zhang, Q.: Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm Evol. Comput.* **1**(1), 32–49 (2011)
228. Zhou, Z., Qiu, Z., Niblett, B., Johnston, A., Zincir-Heywood, N., Heywood, M.I.: A boosting approach to constructing an ensemble stack. In: Proceedings of the European Conference on Genetic Programming, *LNCS*, vol. 13986. Springer (2023)

Part III

Evolution and Neural Networks

In which we set neural networks in the context of evolution. We'll address hardware needs of deep neural networks, large language models, generative models, and adversarial learning.

Chapter 9

Evolutionary Neural Network Architecture Search



Zeqiong Lv, Xiaotian Song, Yuqi Feng, Yuwei Ou, Yanan Sun,
and Mengjie Zhang

Abstract Deep Neural Networks (DNNs) have been remarkably successful in numerous scenarios of machine learning. However, the typical design for DNN architectures is manual, which highly relies on the domain knowledge and experience of neural networks. Neural architecture search (NAS) methods are often considered an effective way to achieve automated design of DNN architectures. There are three approaches to realizing NAS: reinforcement learning approaches, gradient-based approaches, and evolutionary computation approaches. Among them, evolutionary computation-based NAS (ENAS) has received much attention. This chapter will detail ENAS in terms of four aspects. First, we will present an overall introduction to NAS and the commonly used approaches to NAS. Following that, we will introduce the core components of ENAS and discuss the details of how to design an ENAS algorithm with a focus on search space, search strategy, and performance evaluation of the ENAS algorithm. Moreover, detailed implementations of these components will be presented to help readers implement an ENAS algorithm step by step. We will discuss state-of-the-art ENAS methods with the three core components. Finally, we will provide five major challenges and identify corresponding future directions.

9.1 Introduction

Deep Neural Networks (DNNs), as the cornerstone of machine learning [30], have achieved great success in many real-world applications, ranging from image and video processing [29] to artificial language processing [9]. Many observations have demonstrated that the architectures of DNNs have a significant impact on their performance [29, 43]. Therefore, the design of DNN architectures has always been an

Z. Lv · X. Song · Y. Feng · Y. Ou · Y. Sun (✉)
College of Computer Science, Sichuan University, Chengdu 610064, China
e-mail: ysun@scu.edu.cn

M. Zhang
Centre for Data Science and Artificial Intelligence and School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6140, New Zealand

important research topic. For a long time, the architectures of promising DNNs have been manually designed, such as ResNet [21] and DenseNet [23]. Although these DNNs can achieve state-of-the-art performance at development time, their architectures are designed by researchers with rich expertise in neural networks and data. However, in practice, most interested users do not have such knowledge. Moreover, the architectures of DNNs are often problem-dependent. When the problem changes, a new DNN architecture is required to be redesigned manually. Particularly, promising architectures for different datasets are significantly different, e.g., the best ResNet for CIFAR-10 [28] is ResNet110 and the best for ImageNet [42] is ResNet152. As a result, more and more researchers are trying to figure out how to automatically design the architecture of DNNs. This has led to a new field of research known as Neural Architecture Search (NAS). Formally, NAS is a technique that can automatically design high-performance DNN architectures without expert experience [68].

NAS is often formulated as an optimization problem represented by Eq. 9.1:

$$\left\{ \begin{array}{l} \arg \max_A = \mathcal{L}(A, \mathcal{D}_{\text{train}}, \mathcal{D}_{\text{fitness}}) \\ \text{s.t. } A \in \mathcal{A} \end{array} \right. \quad (9.1)$$

where \mathcal{A} denotes a set of network architectures and A denotes a single architecture in the set. $\mathcal{L}(\cdot)$ measures the performance of A on the fitness evaluation dataset $\mathcal{D}_{\text{fitness}}$ after A has been trained on the training dataset $\mathcal{D}_{\text{train}}$. For example, when Convolutional Neural Networks (CNNs) are used for image classification tasks, A is a CNN architecture, and $\mathcal{L}(\cdot)$ is the classification error on the dataset to which A is applied. Generally, NAS is considered an optimization problem facing multiple challenges, such as expensive computation, complex constraints, and multiple conflicting objectives. To solve the NAS problem in Eq. 9.1, researchers have proposed a variety of effective optimization algorithms. Existing NAS methods can be generally classified into Reinforcement Learning (RL) [26] based NAS algorithms, gradient-based NAS algorithms, and Evolutionary Computation (EC) [2] based NAS algorithms (ENAS).

In the early stages, NAS was mostly based on RL algorithms. The initial work of NAS is commonly viewed as NAS-RL [68], which was proposed by Google researchers and accepted for publication by the International Conference on Learning Representations (ICLR) in 2017. As shown in Fig. 9.1, the NAS-RL algorithm is an iterative process with two parts, with the iterative process continuing until the termination condition is satisfied. In the first part, the NAS-RL algorithm uses a Recurrent Neural Network (RNN) controller to sample an architecture. In the second part, NAS-RL uses the trained architecture to update the controller. Through this process, NAS-RL can automatically generate a neural network without manual design, and the designed network can achieve success on the corresponding dataset. However, RL-based algorithms have two major shortcomings. Firstly, they are very time-consuming and resource-consuming. For example, NAS-RL used 800 Graphic Processing Units (GPUs) for four weeks to find a promising CNN architecture on the CIFAR-10 dataset. Secondly, RL-based algorithms are often not completely automatic and rely on expertise. For example, NAS-RL only searched for the hyperparameters of the convolution layers, while other layers were still set manually. In

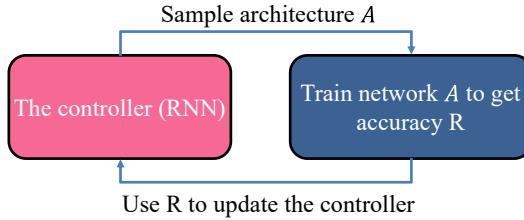


Fig. 9.1 An overview of NAS-RL

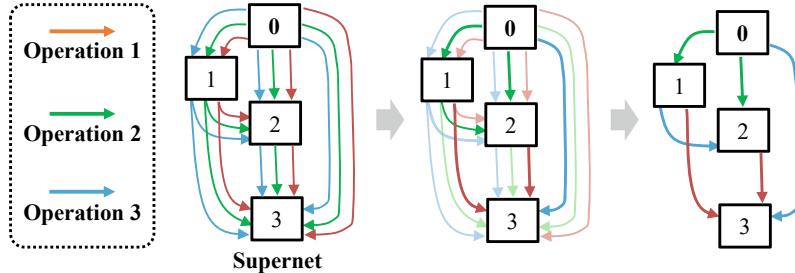


Fig. 9.2 An overview of DARTS

general, using hundreds of GPUs is impractical for most universities and research labs. In addition, domain expertise in designing architectures limits the application of RL-based NAS.

In recent years, gradient-based NAS has been proposed by researchers to improve search efficiency. Gradient-based NAS algorithms relax the discrete search into a continuous optimization problem and then use a gradient-based algorithm [41] to solve it. The classical gradient-based NAS algorithm is DARTS [32], which was proposed by Google researchers and accepted for publication at ICLR in 2019. Figure 9.2 shows the search process of DARTS, which contains three main steps. First, a continuous space known as a supernet is manually constructed, which contains a set of operations on each edge. After that, a gradient descent algorithm is used to optimize operation probabilities and network weights. The final architecture is obtained from the adapted operation probabilities. During this process, DARTS only used one GPU for four days to find the superior CNN architecture on the CIFAR10 dataset, which is a significant improvement over NAS-RL regarding time and resource consumption. Unfortunately, gradient-based NAS techniques require prior manual construction of the supernet, which needs considerable expertise. In addition, the convergence of the optimization algorithm is not guaranteed, and ill-conditioned architectures will be searched, which often results in poor performance.

EC is another major approach to solving the NAS problem. Specifically, EC is a class of population-based meta-heuristic optimization paradigms inspired by the evolution of species and the behaviors of creatures in nature. It has been used to solve complex problems and real-world optimization problems by evolving multiple

solutions at the same time. The most popular EC techniques include Evolutionary Algorithms (EAs) and Swarm Intelligence (SI). EAs are mainly inspired by the process of natural evolution, such as Genetic Algorithms (GAs) and Genetic Programming (GP), while SI is motivated by the behavior of flocking and swarming, such as Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO). Compared to other existing optimization algorithms, EC algorithms do not require gradient information and extensive domain knowledge. Also, EC algorithms have the potential to generate globally optimal solutions. Due to these characteristics, EC methods have been widely used in practice.

In fact, EC algorithms have been already used to search for neural networks over 30 years and the period can be divided into three stages [65]. The first stage is termed “evolving neural networks”, which ended in 1999 with a survey [62]. During this stage, the algorithms were often used to search for both neural architectures and optimal weight values. The neural networks in this stage were often small-scale, i.e., only a couple of normally hidden layers in the evolved networks. The second stage runs from 2000 to 2016 and is known as “neuroevolution” [18]. Different from the first stage, neuroevolution focused on median-scale neural networks. A series of works on the Neuroevolution of Augmenting Topologies (NEAT) [48], such as rtNEAT [46] and HyperNEAT [47], serve as prime examples of neuroevolution. Since 2017, EC has been used to solve the NAS problem, with the third stage being called ENAS. This stage is different from the previous two stages. To begin, previous stages of work frequently used EC to evolve either architectures or weights, or a combination of them, which is very time-consuming. In contrast, ENAS mainly focuses on searching for architectures, and good weight values are obtained by applying efficient gradient-based algorithms. Furthermore, the previous two stages are commonly applied to small-scale and medium-scale neural networks, whereas ENAS generally works on DNNs, such as deep CNNs [40, 52] and deep stacked autoencoders [49], which are stacked with building blocks of deep learning techniques [30]. In general, the seminal work of ENAS is often viewed as the LargeEvo algorithm [40], which was proposed by Google researchers and finally accepted into the 24th International Conference on Machine Learning (ICML) in 2017. The LargeEvo algorithm employed a GA to search for the best architecture of a CNN, and the experimental results on CIFAR-10 and CIFAR-100 [28] have demonstrated their effectiveness. Since then, ENAS algorithms have continued to make breakthroughs in various tasks, such as image classification and signal processing [52].

This chapter aims to provide an introduction to ENAS algorithms, to help readers to learn the basic knowledge of ENAS and design ENAS algorithms for different kinds of application tasks. To achieve this goal, the rest of this chapter is organized as follows. Section 9.2 documents the fundamentals of ENAS and gives a step-by-step example to develop an ENAS algorithm. Section 9.3 discusses the state-of-the-art ENAS algorithms in terms of search space, search strategy, and fitness evaluation. Section 9.4 shows advanced topics with challenges and prospects.

9.2 ENAS Algorithms: Fundamentals and Step-by-step Example

In this section, we first introduce the core components of ENAS in Sect. 9.2.1 and then detail the step-by-step design of an ENAS algorithm in Sect. 9.2.2. To help readers better understand how to build an ENAS algorithm, we will provide an overall flowchart of a common ENAS, after which we document a framework with details for designing each step.

9.2.1 Core Components of ENAS

In general, an ENAS algorithm is composed of three components [14]: search space, search strategy, and performance evaluation. Specifically, the search process takes place in the predefined search space. By using EC as the search strategy, the ENAS algorithm continuously iterates the search-evaluation process in a loop. Each iteration will first search for architectures from the search space and then evaluate their performance in turn. For illustration purposes, we refer to Fig. 9.3, a core component relationship map of an ENAS algorithm.

9.2.1.1 Search Space

The search space consists of all possible architectures that we want to examine, which are determined by the task at hand. It is often exponentially large or even unbounded. In the following, we will introduce how to define a search space.

Search Space: It defines the form and scope of the DNN architectures that the ENAS algorithm can search for. Generally, DNN architectures are composed of different basic units and the connections between the units. Table 9.1 shows the possible architecture configurations of basic units, connections, and other parameters. According to the configurations, the search space can be designed differently by considering a combination of different parameters. Specifically, we can derive different search spaces by considering the configuration of the basic structural units

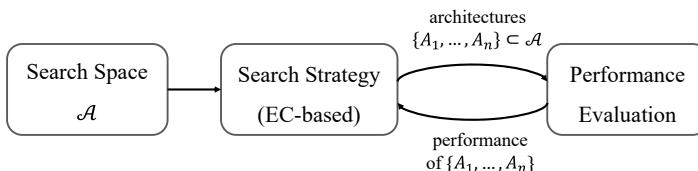


Fig. 9.3 Abstract illustration of the ENAS

Table 9.1 Common parameters optimized in different DNNs [35]

	Parameters
CNN	Global parameters Number of layers, connections between layers
	Convolution layer Filter size (width and height), stride size (width and height), feature map size, convolution type, standard deviation, and mean value of the filter elements
	Pooling layer Filter size (width and height), stride size (width and height), and pooling type
	Fully connected layer Number of neurons, standard deviation, and mean value of weights
DBN, AE	Number of hidden layers, neurons per layer
RNN	Number of hidden layers, neurons per layer, number of time slots

(e.g., layer, block, and cell), the connections between units (e.g., topology), the operations (e.g., convolution, pooling), and other parameters. In addition, the constraints on the search space are important to restrict the search space and lighten the burden of the search process. Constraints mainly focus on *fixed depth*, *rich initialization*, and *partial fixed structure*. In general, there are many different ways to define the search space, and regardless of how it is designed, the goal is to define a set containing potentially good and diverse DNN architectures.

Encoding Strategy: It aims to represent a neural network architecture as an individual in the population. Each ENAS method needs to determine its encoding strategy before starting the first stage of the ENAS method. One of the key issues in encoding DNN architectures is to decide how much information about architecture should be encoded [62]. According to the neural network encoding method, the encoding strategies can be divided into *direct encoding* and *indirect encoding*. Direct encoding is a method that can directly encode the architecture parameters in the chromosome, e.g., every connection and node of an architecture are encoded as binary with the help of a connection matrix. Indirect encoding only represents the important parameters of architecture, such as the hidden layers, while other details are predefined and fixed to reduce the search space. In addition, another encoding strategy category is based on whether all encoded individuals are of the same length. Generally, encoding strategies can be divided into *fixed-length encodings* and *variable-length encodings*. The fixed-length encoding strategy makes it easy to use standard evolutionary operations, which are originally designed for individuals of equal length. In contrast, the variable-length strategy breaks the length limit but possesses stronger capabilities

to find close-to-optimal solutions. This is because the flexibility introduced by the variable length makes the depth, which affects the architecture’s performance, more adaptable. After the encoding strategy has been chosen, the ENAS algorithm can start the evolution of architectures.

Initial Space: It is composed of all the individuals that could be part of an initial population, which can be generated by the initialization procedure. Generally, there are three types of architecture initialization approaches: starting from trivial initial conditions [40], random initialization in the search space [51], and starting from a well-designed architecture (also termed as rich initialization) [19]. These three types of initialization correspond to three different initial spaces: *trivial space*, *random space*, and *well-designed space*. The *trivial space* contains only networks with few primitive layers. The reason for using as little experience as possible is to justify the advantage of the EC-based methods in discovering novel architectures, and the majority of the discovery differs from the manually designed DNN architectures. On the contrary, the *well-designed space* may contain state-of-the-art architectures. In this way, a promising architecture can be obtained at the beginning of evolution, whereas it may hardly evolve to other novel architectures. In fact, many ENAS methods adopting this initial space focus on improving the performance of the well-designed architecture. A *random space* is made up of individuals randomly generated by sampling the search space. This type of initial space aims to reduce human intervention in the initial population.

After the initialization of the population, ENAS algorithms start to search for architectures in the search space. Generally, the search space is the same as the initial space when the random initial space is adopted. For the other two types of initial spaces, however, due to the relatively small initial space, the search space will become much larger with the expectation that promising architectures are included. It is worth noting that many ENAS methods indirectly define the search space by restricting it through evolutionary operators. For example, Irwin-Harris et al. [24] did not specify the maximum depth of the architectures; instead, they used evolutionary operations to extend the architecture to any depth.

9.2.1.2 Search Strategy

The search strategy details how to explore the search space. ENAS algorithms explore the search space by exploiting EC techniques as the optimization approach. These strategies affect the quality and speed of finding well-performing architectures. Based on the strategy adopted, ENAS algorithms can be subdivided to introduce different evolutionary search strategies. Figure 9.4 provides an illustration under three aspects: EA, SI, and others.

EA-based methods account for the majority of the existing ENAS algorithms. Generally, the EA-based search strategy mainly includes the selection strategy and evolutionary operations, which collectively update the population. In practice, GA-based ENAS is the most popular approach, which is largely owed to the convenience of architecture representation in GA. Other categories of EC methods are also

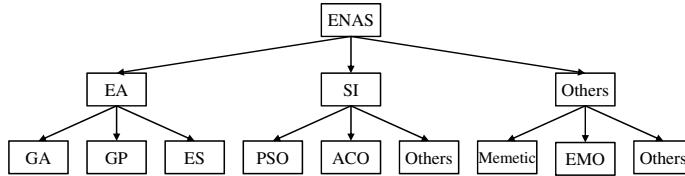


Fig. 9.4 Categories of ENAS from the EC methods regarding the search strategies

important for realizing ENAS algorithms, such as GP, evolutionary strategy (ES), PSO, ACO, memetic algorithm, and evolutionary multiobjective optimization (EMO).

9.2.1.3 Performance Evaluation

The performance evaluation aims to derive the performance (such as accuracy on unseen data) of a given architecture A . The simplest way of evaluating architecture is to train A on training data and estimate its performance on unseen fitness evaluation data.

Model training aims to find the global optimal model parameters based on training data. It can be regarded as an optimization problem:

$$\underset{\theta_A}{\operatorname{argmax}} E_{\text{train}}(A, \theta_A, D_{\text{train}}), \quad (9.2)$$

where θ_A is the modal parameters (e.g., weights) of architecture A , D_{train} represents the training dataset, and E_{train} measures the performance of training θ_A on dataset D_{train} .

Performance evaluation of architecture A usually refers to estimating the trained model θ_A on a fitness evaluation dataset D_{fitness} . It can be represented as

$$E_{\text{fitness}} \left(A, \underset{\theta_A}{\operatorname{argmax}} E_{\text{train}}(A, \theta_A, D_{\text{train}}), D_{\text{fitness}} \right). \quad (9.3)$$

Note that the fitness dataset should not be used for training to guarantee the evaluation of the generalization ability of the model.

9.2.2 Step-by-Step Design of an ENAS Algorithm

This section describes the steps of designing an ENAS algorithm for a real case in detail.

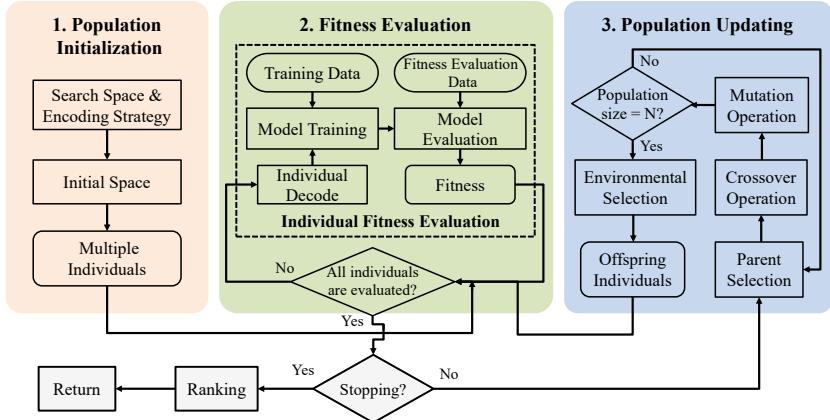


Fig. 9.5 Flowchart of a common ENAS algorithm

9.2.2.1 Overall Flowchart of a Common ENAS

The process of utilizing GA to automatically evolve neural network architectures involves a series of iterations, and during each iteration, numerous architectures will be trained to evaluate their performance to guide the direction of the next iteration.

Figure 9.5 shows a complete illustration of the flowchart of a common ENAS algorithm using standard GA, composed of three parts: population initialization, fitness evaluation, and population updating. In the rest of the subsection, the key steps of the three parts are documented in detail.

9.2.2.2 Population Initialization

At the beginning of the evolutionary process, population initialization creates a base population with multiple individuals. Generally, before designing the detailed population initialization process, the search space and encoding strategy need to be defined based on the type of DNN one wishes to evolve. In this section, we take the CNNs as an example and describe the initialization process of the population in detail. In the following, the search space is layer-based and the initialization procedure randomly samples this search space, while the direct encoding strategy is used for individual initialization.

A CNN architecture is constructed using multiple convolutional layers and pooling layers arranged in a specific order, along with their corresponding parameters (e.g., the feature maps of the convolutional layer and the pooling type). Particularly, all filter and stride sizes use the same settings, i.e., 3×3 and 1×1 , respectively. To this end, the encoding information only has the layer type and one parameter setting. In the designed encoding strategy, each layer is abstracted as a node and is encoded as two parts, i.e., the *node.type* and the *node.parameter*. If the layer type

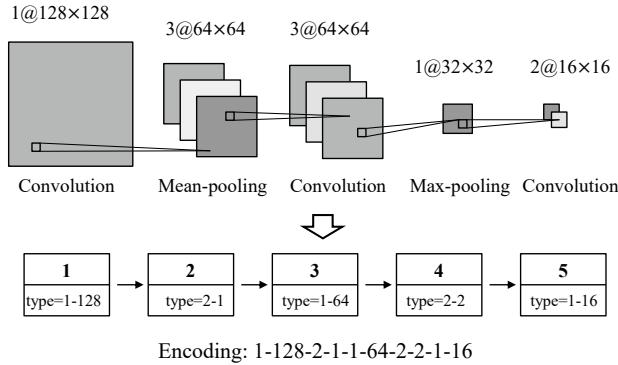


Fig. 9.6 Example of the indirection and fixed-length encoding strategy encoding a CNN with five nodes

is convolutional, its node parameter (denoted as M) is the number of feature maps, and its value is an integer; otherwise, the layer type is pooling, and its parameter (denoted as P) is the pooling type, e.g., max-pooling or mean-pooling. Thus, a CNN architecture encoding can be described as a list of nodes where each node stores the node type and node parameter. Figure 9.6 gives an example of a CNN architecture encoding that contains five nodes.

The population initialization steps are summarized in algorithm 1. μ individuals are initialized with the same node length L (line 1), and each individual randomly generates L nodes (line 1). During the node-generating process, two types of labels can be marked for each node (lines 1–1). Specifically, the assigned node with label $\{node.type = 1, node.M\}$ represents that it is a convolutional layer with M feature maps (lines 1–1), whereas label $\{node.type = 2, node.P\}$ represents the node is a pooling layer with P as the pooling type, where the pooling type is abbreviated as *max* or *mean* (lines 1–1).

9.2.2.3 Population Updating

The population updating process includes two steps: offspring generation and environmental selection. Based on the adopted GA, the evolutionary process for offspring generation includes parent selection, crossover operation, and mutation operation. The conventional mutation strategy employed in GAs includes the bit modification that is achieved by flipping the bit of the encoding, while in the ENAS algorithm, there are three distinct types of mutation strategies:

1. *Addition*: adding any unit to the selected position;
2. *Removal*: removing the unit at the selected position;
3. *Modification*: modifying the encoded information of the unit at the selected position.

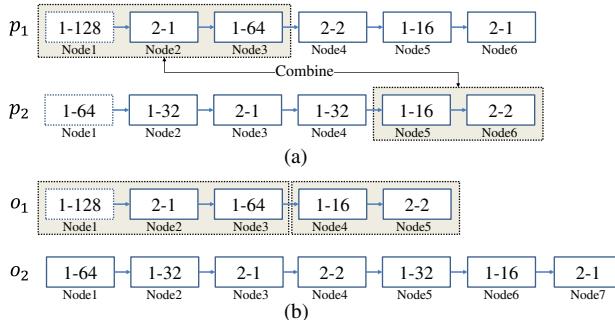
Algorithm 1: Population Initialization

Data: The population size μ .
Output: The initialized population P_0

```

 $P_0 \leftarrow \emptyset;$ 
while  $|P_0| < \mu$  do
     $L \leftarrow$  Randomly generate an integer greater than zero as the individual length;
     $list \leftarrow$  Create a linked list contains  $L$  nodes;
    for  $node$  in the  $list$  do
         $r \leftarrow$  Uniformly generate a number from (0,1);
        if  $r < 0.5$  then
             $node.type \leftarrow 1$ , i.e., this node is the convolutional layer;
             $node.M \leftarrow$  Randomly generate an integer greater than zero;
        else
             $node.type \leftarrow 2$ , i.e., this node is the pooling layer;
             $q \leftarrow$  Uniformly generate a number from (0,1);
            if  $q < 0.5$  then
                 $node.P \leftarrow max$ , i.e., this pooling layer is the max-pooling type;
            else
                 $node.P \leftarrow mean$ , i.e., this pooling layer is the mean-pooling type;
     $P_0 \leftarrow P_0 \cup list;$ 

```

**Fig. 9.7** Example of the “one-point” crossover operation

It is worth noting that operating the *addition* and *removal* mutation strategies can change the length of the individuals. Algorithms 2 and 3 show the offspring generation and environmental selection process, respectively. Figures 9.7 and 9.8 give examples of mutation and recombination operations considering one-point crossover and three types of mutation.

9.2.2.4 Fitness Evaluation

Generally, fitness is calculated based on the encoding information and the task at hand. For example, in the image classification task, an individual’s fitness is the classification accuracy on the corresponding fitness evaluation dataset. For

Algorithm 2: Offspring Generation

Data: The population P_t containing individuals with fitness, the probability p_c for crossover operation, the probability p_m for mutation operation, the operation list l_m .

Output: The offspring population Q_t .

```

 $Q_t \leftarrow \emptyset;$ 
while  $|Q_t| < |P_t|$  do // Crossover
   $p_1 \leftarrow$  Randomly select two individuals from  $P_t$ , and then select one with the better fitness;
   $p_2 \leftarrow$  Repeat Line 3;
  while  $p_2 == p_1$  do
    | Repeat Line 4;
   $r \leftarrow$  Randomly generate a number from  $[0, 1]$ ;
  if  $r < p_c$  then
    | Randomly choose a point in  $p_1$  and divide it into two parts, i.e.,  $p_1^1$  and  $p_1^2$ ;
    | Randomly choose a point in  $p_2$  and divide it into two parts, i.e.,  $p_2^1$  and  $p_2^2$ ;
    |  $o_1 \leftarrow$  Join the first part  $p_1^1$  of  $p_1$  and the second part  $p_2^2$  of  $p_2$ ;
    |  $o_2 \leftarrow$  Join the second part  $p_1^2$  of  $p_1$  and the first part  $p_2^1$  of  $p_2$ ;
    |  $Q_t \leftarrow Q_t \cup o_1 \cup o_2$ ;
  else
    | |  $Q_t \leftarrow Q_t \cup p_1 \cup p_2$ ;
for individual  $p$  in  $Q_t$  do // Mutation
   $r \leftarrow$  Random generate a number from  $[0, 1]$ ;
  if  $r < p_m$  then
    | |  $i \leftarrow$  Randomly choose a point in  $p$ ;
    | |  $type \leftarrow$  Randomly select one mutation strategy from Addition, Removal, Modification;
    | | if  $type$  is Addition then
    | | |  $m \leftarrow$  Select one operation from  $l_m$ ;
    | | | Adding operation  $m$  at the point  $i$ ;
    | | else if  $type$  is Removal then
    | | | Removing the operation at the position  $i$ ;
    | | else
    | | |  $m \leftarrow$  Select one operation from  $l_m$ ;
    | | | Modifying the origin operation as  $m$  at the position  $i$ ;
  
```

Algorithm 3: Environmental Selection

Data: The parent population P_t , the offspring population Q_t

Output: The population for the next generation P_{t+1} .

```

 $P_{t+1} \leftarrow \emptyset;$ 
while  $|P_{t+1}| < |P_t|$  do
  | |  $\{p_1, p_2\} \leftarrow$  Randomly select two individuals from  $P_t \cup Q_t$ ;
  | |  $p \leftarrow$  Select the one with a higher fitness from  $\{p_1, p_2\}$ ;
  | |  $P_{t+1} \leftarrow P_{t+1} \cup p$ ;
 $p_{best} \leftarrow$  Find the individual with the best fitness from  $P_t \cup Q_t$ ;
if  $p_{best}$  is not in  $P_{t+1}$  then
  | | Replace the one who has the worst fitness in  $P_{t+1}$  by  $p_{best}$ ;
  
```

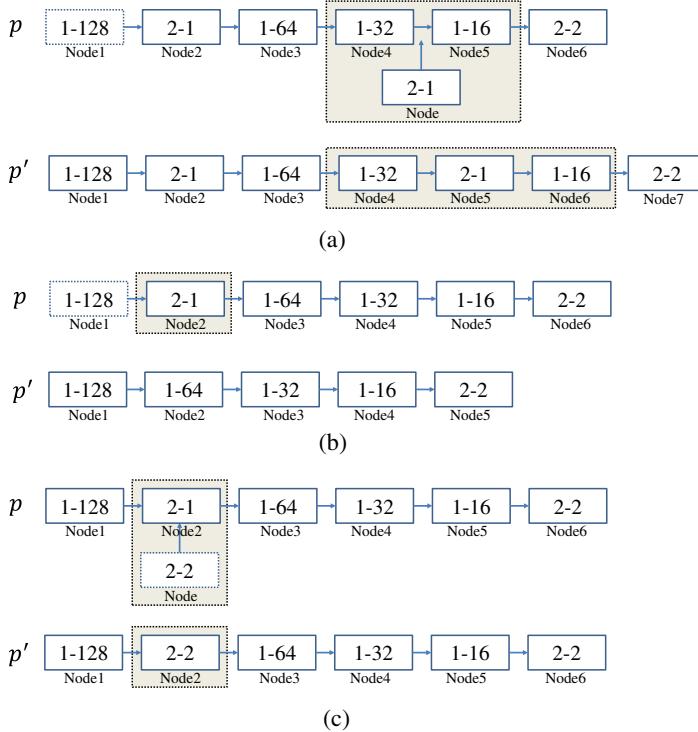


Fig. 9.8 Example of the **a** “Addition”, **b** “Removal”, and **c** “Modification” mutation

evaluating individual fitness, the individuals in each generation need to be decoded into a corresponding neural network architecture and then trained like a common neural network. In the case of searching CNN for the image classification task, the individual is decoded into a CNN and then added to a classifier to be trained. The decoded CNN is trained on training data, and fitness is the highest classification accuracy on the fitness evaluation data after the CNN training.

Algorithm 4 details the fitness evaluation of individuals in the population P_t . First, each individual p in population P_t should be decoded as a CNN architecture from the encoding information (line 4), which is an inverse of the encoding strategy introduced in the population initialization. Second, the CNN is initialized with weights (line 4) like a hand-crafted CNN. Generally, the weight initialization method is the Xavier initializer. Third, each CNN is asynchronously placed on an available GPU for training on the training dataset (line 4). Generally, the training method is stochastic gradient descent (SGD) with momentum, which is commonly used in the deep learning community. Finally, once the training phase is finished, the trained DNN is evaluated on the fitness evaluation dataset (line 4), and the evaluated performance acc represents the fitness of the individual (line 4). It is important to note that both D_{train} and $D_{fitness}$ here are training sets: D_{train} is for training the weights, while $D_{fitness}$

is for training the architectures, and both architecture and the weights are part of the DNN model.

Algorithm 4: Population Fitness Evaluation

Data: The population P_t , the available GPU, the number of training epochs, the training data D_{train} , and the fitness evaluation data D_{fitness} .

Output: The population P_t with fitness.

for individual p in P_t **do**

- | $cnn \leftarrow$ Transform (Decode) the information encoded in the individual p to a CNN with the corresponding architecture;
- | Initialize the weights of cnn ;
- | Train the cnn on D_{train} by using the given GPU;
- | $acc \leftarrow$ Evaluate the performance of the trained cnn on D_{fitness} ;
- | Set acc as the fitness of individual p .

9.3 Discussions of State-of-the-art ENAS

Nowadays, many of the neural architectures generated by ENAS have outperformed hand-crafted architectures across various domains. This breakthrough benefits from large quantities of works studying the three core components of ENAS, i.e., search space, search strategy, and performance evaluation. In this section, some key state-of-the-art designs of these three components are discussed.

9.3.1 Search Space

Commonly, the search space can be divided into three categories according to the basic units adopted: layer-based search space, block-based search space, and cell-based search space [35]. But there is one exception: some ENAS methods only take care of the connections between units instead of the basic units themselves and cannot be classified in any of the above categories [58]. Therefore, an extra category named topology-based search space is introduced to represent this type of search space.

9.3.1.1 Layer-Based Search Space

For this category, the basic units of the search space are primitive layers, such as convolution layers, pooling layers, and fully connected layers. The search space used in the step-by-step design of the ENAS algorithm in Sect. 9.2.2 is exactly the simplest case of the layer-based search space, which only adopts the convolutional

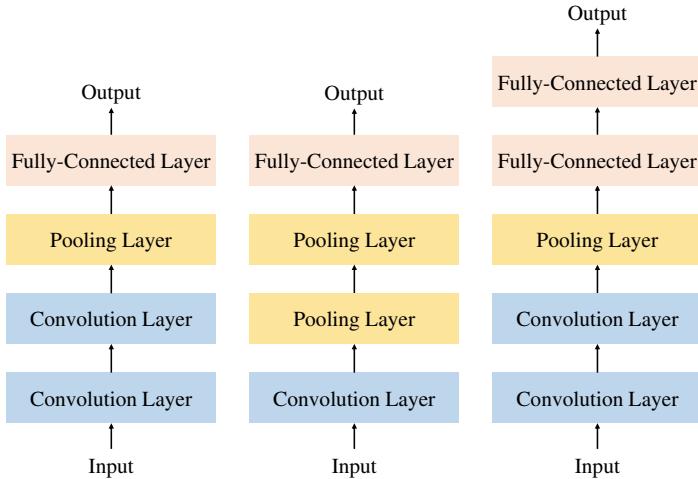


Fig. 9.9 Three examples of neural architecture of the layer-based search space

layer and the pooling layer. Three more complex examples of neural architectures in the layer-based search space are shown in Fig. 9.9. As can be seen in the figure, different combinations of the layers correspond to different candidate architectures in the layer-based search space, and different architectures may have a different number of layers. There may also be some limitations, e.g., pooling layers must come after convolution layers and fully connected layers must come after pooling layers.

This type of search space is a feasible solution commonly adopted by early works [19, 51] because of its simplicity. However, there are two significant limitations in this design. First, the layer-based search space often leads to low search efficiency. Specifically, well-performing DNNs are commonly composed of hundreds of primitive layers. As a result, if we want to search for well-performing DNNs in a layer-based search space, the search space should also include candidate neural architectures containing hundreds of primitive layers. However, the number of candidate neural architectures grows exponentially as the size of neural architectures increases. Second, the quality of the evolved solution may be underwhelming when compared with hand-crafted ones, such as ResNet [21] and DenseNet [23]. This is because the skip connections, recognized as the most important component of these hand-crafted neural architectures for performance improvement, cannot be represented by primitive layers.

9.3.1.2 Block-Based Search Space

The block-based search space implies that the basic units of the search space are blocks composed of primitive layers and skip connections. The blocks are predefined by experts, with fixed primitive layers and topological relationships, with the hope

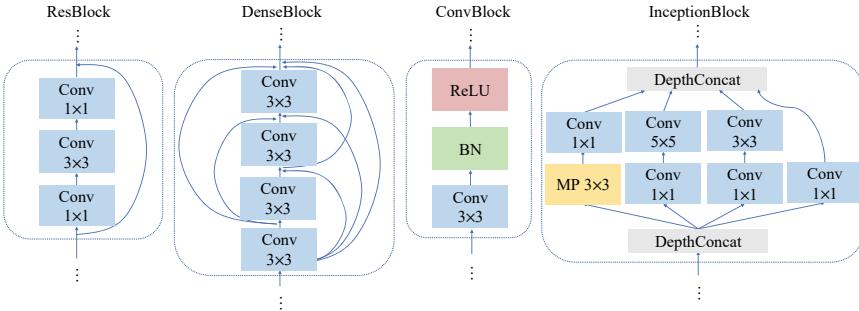


Fig. 9.10 Illustrations of ResBlock, DenseBlock, ConvBlock, and InceptionBlock

that the designed blocks have promising performance. Traditional blocks include ResBlock [21], DenseBlock [23], ConvBlock (Conv2d + Batch-Normalization + Activation) [13], and InceptionBlock [55], which are shown in Fig. 9.10.

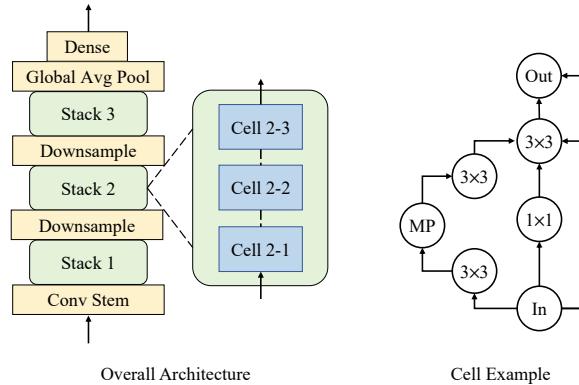
It is easier for the block-based search space to find good neural architectures than the layer-based search space for two reasons. First, the block-based search space contains better neural architectures than those contained in the layer-based search space. This is because the block-based search space can make full use of the effectiveness of skip connections. Some excellent hand-crafted architectures, such as ResNet and DenseNet, are part of this space. Second, the search efficiency of a block-based search space is significantly higher than the layer-based search space, especially when searching for deep neural architectures. This is because the block-based search space regards complex modules (i.e., blocks) as basic units, which leads to significantly fewer candidate neural architectures when the depth of the candidate architectures is fixed.

Obviously, block design is a key task of block-based search. In practice, some ENAS methods employed the conventional blocks directly [50], while others designed different blocks according to specific scenarios. For example, Chen et al. [6] proposed eight blocks, including ResBlock and InceptionBlock encoded in a 3-bit string, and used Hamming distance to determine similar blocks. Song et al. [45] proposed three residual dense mixed blocks to reduce the computational cost caused by the convolution operation of image super-resolution tasks.

9.3.1.3 Cell-Based Search Space

The cell-based search space is similar to the block-based one. Commonly, the search space can be divided into a micro part and a macro part. The micro part focuses on how to construct the basic units, and the macro part defines the topological relationship between basic units. For example, the basic units of the block-based search space are predefined blocks, and only the connections between different blocks remain to be

Fig. 9.11 Overall architecture and cell example of NAS-Bench-101

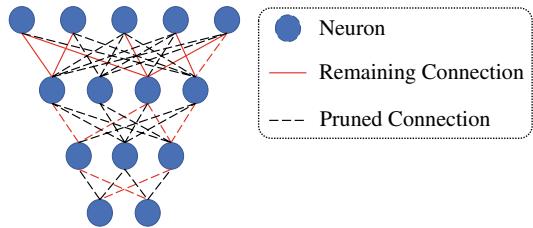


determined. In this situation, the block-based search space only supports the search process on the macro part but not on the micro part.

Generally speaking, the search part of the cell-based search space is exactly opposite to the block-based search space, i.e., the cell-based search space focuses on the micro rather than the macro part. The basic units of the cell-based search space are cells whose structures are determined during the search instead of being predefined. These cells are repeatedly stacked according to a predefined topological relationship, combining their structures to form the final architecture. The commonly used cell-based search spaces include NAS-Bench-101 search space [63], NAS-Bench-201 search space [12], NASNet search space [69], and DARTS search space [32]. For better understanding, we take the NAS-Bench-101 search space as an example, and we explain how to form a candidate architecture in detail. As shown in Fig. 9.11, the overall architecture starts at the convolutional stem and ends with a global average pooling layer and a dense softmax layer. Its body comprises three stacks, and every stack is obtained by stacking three repeated cells. In addition, a downsampling layer is added between every two stacks. Please note that all the cells in one candidate architecture have the same structure and a cell example is shown in Fig. 9.11.

The cell-based search space has several advantages. Firstly, it significantly reduces the size of the search space. This is because the cell-based search space only searches for a few types of cells (typically one to two types) and constructs the final architectures by repeatedly stacking these cells. Secondly, neural architectures in the cell-based search space have the potential to surpass hand-crafted neural architectures. This is because cell-based search space concentrates on the micro part and can find cells that include but are not limited to conventional structures such as ResBlock and DenseBlock. Thirdly, neural architectures searched from the cell-based search space demonstrate excellent transferability in practice. Specifically, the cells can be stacked to form neural architectures with varying depths (i.e., number of cells) and widths (i.e., number of filters), which can be applied to different tasks. For example, in the NASNet search space, the final architectures are commonly obtained by stacking 20 cells with an initial channel number of 36 for small-scale datasets such

Fig. 9.12 An example of combining the topology-based search space with pruning



as CIFAR-10, while the settings change to 14 cells with an initial channel number of 48 for large-scale datasets such as ImageNet.

9.3.1.4 Topology-Based Search Space

Topology-based search does not consider the configuration of the structural units (layer or block) and is only concerned with the connections between units. This type of search space is usually associated with specific practices. Two examples of the topology-based search space are considered next.

One typical case is to combine the topology-based search space with pruning. In particular, Wu et al. [58] construct a topology-based search space based on the shallow VGGNet [43], where the candidate neural architectures can be obtained by pruning unimportant weight connections from the VGGNet. An example of the neural architecture in this topology-based search space is shown in Fig. 9.12. In the figure, the circles denote neurons in the network, the red solid lines denote the weight connections that still remain, and the black dashed lines denote the pruned connections. Only the solid lines and the nodes that are connected by the solid lines together represent a candidate neural architecture. Please note that the figure only shows one specific example of the candidate neural architecture, and all possible combinations of nodes and solid lines denote candidate architectures, which together form this topology-based search space.

Another classical application is in the one-shot NAS method, in which all candidate architectures can be represented by subgraphs of a supergraph [14]. Yang et al. [61] proposed CARS to search for architectures by choosing different connections in the supergraph, and then the architecture was built by the chosen subgraph. Consider the simplest case, as shown in Fig. 9.13, a supergraph composed of one input node, three different operations (including a 3×3 convolution, a 5×5 convolution, and max-pooling), and one output node summing all its inputs. Then a subgraph can be obtained by only activating one operation (i.e., the max-pooling in the figure), which represents the architecture that passes input through the max-pooling layer to derive its output. Similarly, the other two candidate architectures can be obtained by activating the 3×3 convolution and 5×5 convolution, respectively. As a result, the simple search space contains three candidate architectures.

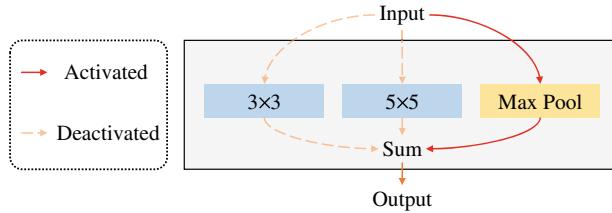


Fig. 9.13 An example of combining the topology-based search space with the one-shot method

9.3.2 Search Strategy

There are two methods to perform the population initialization: fixed-length encoding and variable-length encoding.

9.3.2.1 Fixed-Length Encoding

As the name states, the individuals will all have the same length in the search process under the fixed-length encoding strategy. For example, Xie et al. [59] encode each architecture layer as a fixed-length binary string, and then these binary strings for each layer are put together to form the encoding of the whole architecture. Loni et al. [36] define the chromosome of individuals with a fixed number of parts. In each chromosome, each part has its own specific meaning, such as the type of activation functions and the kernel size of convolution layers. In addition, Broni et al. [5] represent individuals as chromosomes consisting of fixed-length strings. Each gene is encoded through cellular encoding, which is a generative encoding based on simple local graph transformations.

Here we take DeepMaker [36] as an example to show how to encode the neural architectures into a series of fixed-length encodings in detail. Since DeepMaker is proposed to search for both accurate and lightweight architectures, the encoding of each individual contains five parts: the number of condensed layers which makes the resulting architecture light, the number of convolution layers which makes the resulting architecture accurate, the type of activation function and optimizer, and the kernel size of the convolution layers. With this encoding strategy, each individual can be denoted by a series of encodings of a fixed length. In Fig. 9.14, the general genotype and two encoded individuals are shown. The general genotype is a list that contains information on the five aspects introduced above. Moreover, in the individuals, the number of the condense layers and convolution layers are determined, which are shown as 2, 3 and 3, 1 in this example, and then the type of the activation function, the optimizer and the kernel size are selected from the candidates.

Discussion: Because the individuals all have the same length of encoding, the standard crossover and mutation operators can be directly used in the search process. However, though fixed-length encoding has such advantage, there is also a significant

Genome Type:	Condense Layer	Convolution Layer	Activation Function	Optimizer	Kernel Size
	2	3	tanh	adam	3×3
	3	1	relu	sgd	5×5

Individual 1 Individual 2

Fig. 9.14 An example of the fixed-length encoding in DeepMaker [36]

drawback. The length of the individuals needs to be determined in advance under the fixed-length constraint. Because the optimal length of the individuals is not known in advance, determining the optimal depth of the networks (i.e., the length of the individuals) highly relies on expertise in designing neural architectures. ENAS methods that utilize the fixed-length encoding strategy do not have the ability to search globally for optimal network depths, which prevents them from being completely automated. For instance, assume that the optimal depth of the network is eight. However, if the predetermined length of the individuals is six, the NAS algorithm will never search the optimal neural architecture. Variable-length encoding is designed to address the problems above, and this encoding strategy is introduced in the following subsection.

9.3.2.2 Variable-Length Encoding

When it comes to variable-length encoding, the length of different individuals is allowed to be different. For instance, Sun et al. [51] designed a layer-based variable-length encoding strategy to achieve the completely automatic design of CNN architectures. Wang et al. [56] and Sun et al. [53] borrow the idea of the IP address to encode architectures. In this encoding strategy, each layer in the architecture is determined based on the range of their IP addresses, and a layer called the “disabled layer” is designed to achieve the goal of “variable length”. For example, there are two architectures with a depth of five. The first architecture consists of five convolution layers, and the second architecture consists of four convolution layers and one disabled layer. Actually, the depth of the second architecture is considered four because of the presence of the disabled layer. Individuals within the population exhibit different lengths based on this type of design.

The EvoCNN method [51] proposed by Sun et al. is taken as an example and introduced as follows. There are three basic units designed and utilized to construct the individuals, the convolution layer, the pooling layer, and the fully connected layer, as shown in Fig. 9.15. Furthermore, these three kinds of layers are stacked to construct a single chromosome which denotes an individual in the population, and there are three constructed individuals with different lengths shown as examples in Fig. 9.15.

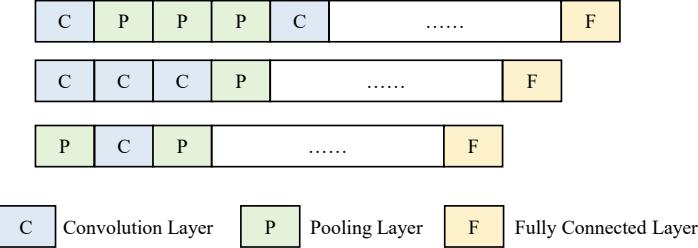


Fig. 9.15 Examples of the variable-length individuals in EvoCNN [51]

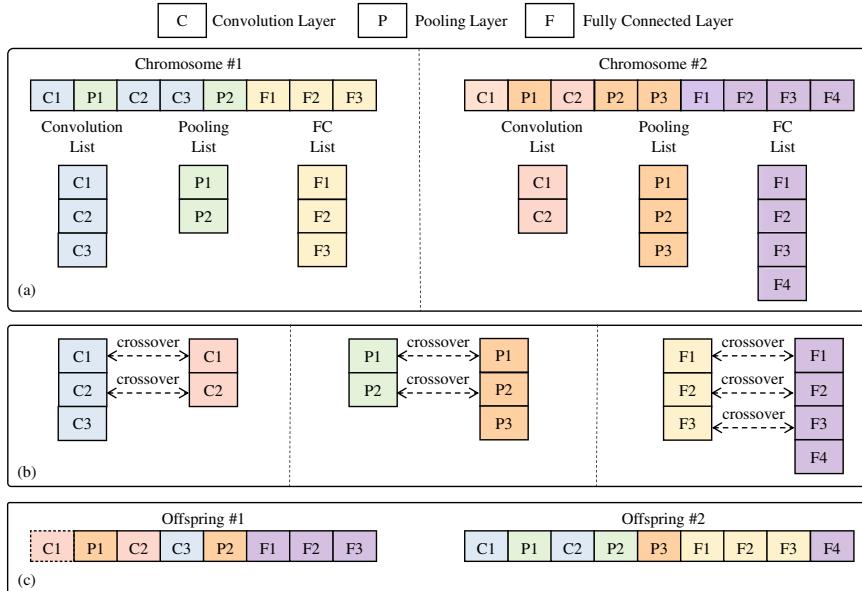


Fig. 9.16 An example of the crossover operation designed in EvoCNN [51]

Under the variable-length condition, classical evolutionary operator design often cannot achieve the desired results. Special operator design needs to be proposed for variable-length individuals, as introduced in the following paragraphs.

Crossover: In this case, we take the design of the crossover operator of EvoCNN ssssssssss [51] proposed by Sun et al. as an example to help the readers understand the overall flow of the crossover operation under the variable-length condition (see Fig. 9.16). The first individual presented is of length eight and contains three convolution layers, two pooling layers, and three fully connected layers, and the second individual presented is of length nine and includes two convolution layers, three pooling layers, and four fully connected layers. With the initialization, the first step of crossover is to stack the layers of the same type by the order they appear in the

chromosome of the individuals, which is shown in part (a). After that, the lists which contain the same type of layers are aligned at the top, the layers are paired, and the crossover operation is performed, as shown in part (b). Finally, as shown in part (c), the layers in the lists are restored based on their initial order in the chromosome.

Mutation: The mutation operator employed in EvoCNN includes adding, removing, or modifying the layer at the selected mutation point. The mutation operation can occur at each point of the chromosome of a specific individual with a pre-determined probability. In addition, for the modifying operation, when replacing a layer with another layer, the information encoded will have to be changed accordingly, such as filter width of the convolution layers and the number of neurons in fully connected layers.

Discussion: Although there are several significant advantages of the variable-length encoding described in Sect. 9.2.1.1, there are also some disadvantages. First, the general genetic operators, especially the crossover operator, cannot be used under variable-length encoding because the general crossover operator specifies that the two individuals crossed over must be the same length. Second, the variable-length encoding may lead to excessively large individuals (i.e., over-depth architectures). These over-length individuals can increase the time and resource budget for the fitness evaluation process, thereby reducing the overall efficiency of the ENAS algorithm.

9.3.3 Fitness Evaluation

Performance evaluation techniques can be divided into four categories: training from scratch, low fidelity, fitness caching, and weight inheritance.

9.3.3.1 Training from Scratch

Training from scratch is the earliest and most accurate way to evaluate architecture performance. If we want to know the architecture performance, we typically make a full training on the training set, evaluate the architecture with its trained weights on the validation set, and finally, the real performance is obtained on the unseen test set.

This type of performance evaluation is inefficient since training complex networks is costly and ENAS algorithms require the evaluation of many individuals. For instance, Real et al. [40] employed 250 computers over 11 days to complete the LargeEvo algorithm. Given its resource-intensive nature and inefficiency, this technique is unaffordable for most users, making it less attractive nowadays.

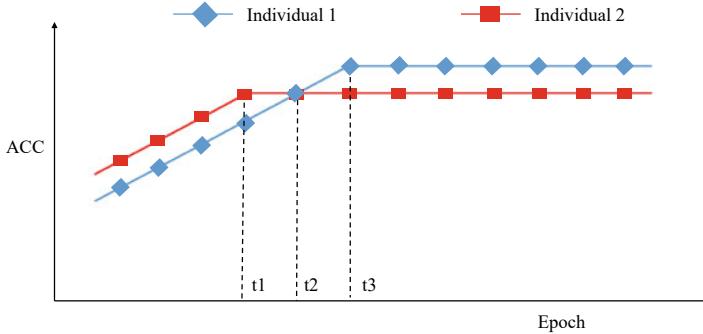


Fig. 9.17 Two learning curves of two different individuals

9.3.3.2 Low Fidelity

Low fidelity evaluation is a performance evaluation technique that assesses neural architectures through proxy tasks. This technique is based on the assumption that the architecture performance on the proxy task is close to the real performance or that their performance rankings are consistent with those on the original task. Low fidelity has long been a popular performance evaluation technique for its efficiency and has produced many specific methods. Next, we will introduce some popular low fidelity methods.

Early Stopping: Early stopping policy is a widely used low fidelity method, acting on the training settings. Its simplest form consists of training the neural architectures for a small and fixed number of epochs, on the basis that the performance after a small number of training epochs can reflect the rankings of the real performance. This method is practiced by the work of [49], where training individuals with a small number of epochs is enough. Assunção et al. [1] improve the method by shortening the training time in each epoch instead of the number of training epochs, which allows the neural architectures to undergo all the training epochs under a more precise evaluation. So et al. [44] proposed to set hurdles after a fixed number of epochs so that weak individuals stop training early to save time, while well-performing individuals can continue training to get precise performance.

Sometimes, early stopping does not reveal the real performance rankings of the individuals (especially when large and complicated neural architectures are involved) and may result in poor individuals being selected. As shown in Fig. 9.17, the individual that performs better in the early training stage is not necessarily better in the final training stage. Specifically, *individual2* appears to be better than *individual1* before epoch t_2 , but the opposite is true after epoch t_2 . As a result, *individual1* is a better neural architecture that should be chosen according to their real performance. However, if the early stopping policy is applied, and the stopping epoch is set to be smaller than t_2 , then *individual2* is chosen by the algorithm, which is not consistent with our expectations. Therefore, it is crucial to determine the point at

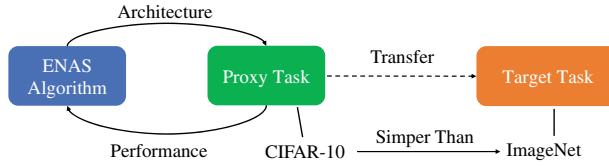


Fig. 9.18 An illustration of the reduced training set method

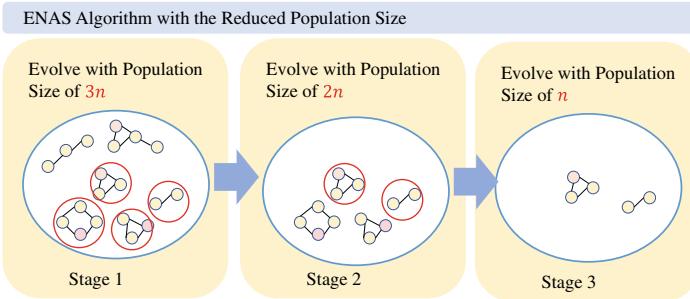


Fig. 9.19 An illustration of the ENAS algorithm with the reduced population size

which the training process should stop. Commonly, neural networks can be trained to convergence, after which the training is meaningless and no longer leads to performance improvement. As can be seen in Fig. 9.17, *individual1* converges at epoch t_3 , and *individual2* converges at epoch t_1 . It is reasonable to choose epoch t_3 as the stopping point of *individual1* and epoch t_1 as the stopping point of *individual2*, respectively. Therefore, some methods [19] stop training when observing there is no significant performance improvement over a certain number of epochs.

Reduced Training Set: In this method, a subset of the original or a smaller dataset is used as the proxy dataset. In the search phases, the individuals are evaluated on the proxy dataset and compared according to the proxy results. Once evolution is over, the evolved neural architecture is transferred to the target task for final use. In practice, thanks to so many benchmark datasets in the image classification field, it is convenient to evaluate and compare the architectures on a smaller dataset such as CIFAR-10 first and then apply the better architectures on a large dataset such as CIFAR-100 and ImageNet. An illustration of the method is shown in Fig. 9.18. As we know, the CIFAR-10 dataset is significantly simpler than the ImageNet dataset, so the search process can be a hundred times faster than searching directly on the ImageNet dataset.

In addition, using a subset of the original dataset as a proxy task is also feasible. For example, Liu et al. [34] searched for promising architectures by training on a subset of data. After that, transfer learning can be used on the large original dataset.

Reduced Population: Reducing the population size is also a low fidelity method, unique to EC algorithms because non-EC algorithms have no concept of population.

The simplest way is to reduce the population size directly to speed up the evolution. However, since the population size is an important parameter, which may limit the global search ability of the ENAS algorithms, simply reducing population size may lead to significant performance degradation. An improved method was proposed to reduce the population size dynamically, with a balance between efficiency and performance. Fan et al. [17] used the $(\mu + \lambda)$ evolution strategy and divided the evolution into three stages with population reduction, as illustrated in Fig. 9.19. At the first stage, the population is set to be $3n$, where n equals two in the example. As a result, the search process is performed with a population size of 6 at the first stage. Once the first stage is completed, population size is reduced to $2n$, which is equal to four in the example, and the best four individuals circled in the figure at the first stage are kept in the second stage for continued evolution. Similarly, after the second stage of evolution, the population size is reduced to n , which is equal to two in the example, and the best two individuals in the first stage are retained for the final stage of evolution. In this method, the population is kept large at the first stage, promoting the global search ability of the evolution process. At the next stages, the population size is reduced to shorten the evolution time.

In addition to reducing population size, reducing the complexity of individuals in the population is also feasible. For example, Liu et al. [33] began the evolutionary process with small-sized architectures for time-saving. Similarly, Wang et al. [57] searched for well-performing blocks instead of the whole architectures at the early stage. After that, the evolutionary process was performed among the architectures composed of the searched blocks.

9.3.3.3 Fitness Caching

Fitness Caching refers to another category of the performance evaluation technique, which saves architectural information for reuse. This category is especially popular in ENAS algorithms, because in the population-based methods, it is natural to always keep the best few individuals in each generation for the next generation. When the performance of the same architecture is repeatedly required, performance is directly obtained by the cache mechanism, which is significantly cheaper and more efficient than evaluating the architecture again.

There are many different approaches in practice. For instance, the simplest way is to directly store the fitness of individuals in memory. Before evaluating an architecture, if the information can be found in memory, it can be retrieved from memory instead of reevaluating the architecture. An example is shown in Fig. 9.20. When the offspring K is generated, we first check the cache that can be a lookup table. If Individual K exists in the lookup table, we directly assign F_K to offspring K instead of reevaluating it.

Similarly, Sun et al. [52] saved information pairs of architectures and their corresponding fitness values, but used a hashing method to accelerate the query process. Johner and Wassner [25] utilized the caching mechanism in another form, prohibiting the appearance of architectures that existed before offspring generation.

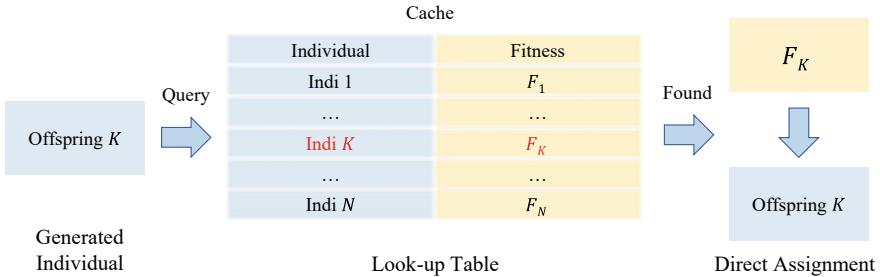


Fig. 9.20 An example of fitness caching

This approach did succeed in reducing time, but with the limitation that the best individuals were forbidden to survive in the population, possibly misleading the search.

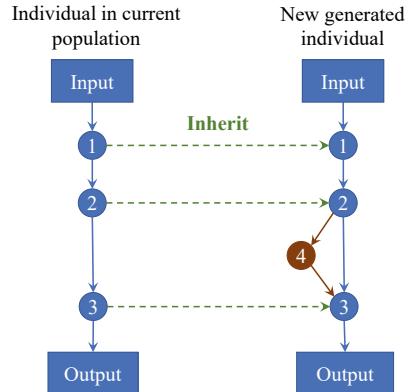
9.3.3.4 Weight Inheritance

Weight inheritance is a category of performance evaluation technique that uses the weight information of previously trained neural networks based on the assumption that similar architectures may have similar weights. This technique is also applied in ENAS algorithms because offspring generated by crossover and mutation during the evolutionary process usually have parts as their parents, and the weights of the same parts can be inherited. One specific example is shown in Fig. 9.21. In the figure, the architecture on the left is an individual in the current population. It has three nodes, numbered 1 to 3 in sequence, connected one by one. The architecture on the right is a newly generated individual with four nodes, numbered 1 to 4. Among them, in addition to node number 4, the other three nodes have the same topological relationship as *Architecture1*. As a result, these three nodes can directly inherit the weights from the corresponding nodes of *Architecture1*, and only the weights of node number 4 are trained from scratch. With the weight inheritance technique, the neural networks only need to be fine-tuned for a few epochs, vastly reducing the search costs.

9.4 Advanced Topics

In this section, we introduce five advanced topics in ENAS: benchmarks of ENAS, efficient evaluation, resource-constraint ENAS, interpretability of ENAS, and theoretical analysis of ENAS.

Fig. 9.21 An example of weight inheritance



9.4.1 Benchmarks of ENAS

As research into ENAS continues, an increasing number of ENAS algorithms are being proposed. To verify their effectiveness, extensive comparison with similar methods is required. However, since different methods use different search spaces and training methods, a completely fair comparison is generally difficult to achieve. Addressing this challenge, a popular benchmark platform called BenchENAS [60] has been implemented recently, serving as a valuable tool for ENAS researchers to compare different ENAS algorithms relatively fairly. The overall framework of BenchENAS, depicted in Fig. 9.22, consists of five main components: the ENAS algorithms (top part), the data settings (left part), the trainer settings (right part), the runner (middle part), and the comparer (bottom part).

In the BenchENAS framework, users begin by selecting an ENAS algorithm to run with the data settings and trainer settings. Subsequently, through the runner and the comparer components, relatively fair comparison results are obtained. In the component of ENAS algorithms, users can choose one ENAS algorithm provided by BenchENAS or implement their own algorithm. The component of data settings allows users to select candidate datasets such as MNIST, CIFAR-10, CIFAR-100, and ImageNet, which are popular benchmark datasets for image classification. In addition, users can configure the scheduler of the learning rate, the settings of the batch size and the number of epochs, and the type of optimizer in the trainer settings component. Once all settings are determined, the runner component executes the selected ENAS algorithm. In particular, in the runner component, the whole algorithm is launched and run by the center node. Then a well-designed evaluator is used to evaluate the fitness of each individual in the population. At the end of the algorithm execution, the comparer component obtains the indicators, including the accuracy, the number of flops, the parameter size, and the GPU days. After obtaining these indicators, the users can compare different ENAS algorithms fairly. Despite BenchENAS taking a significant step forward in enabling fair comparisons

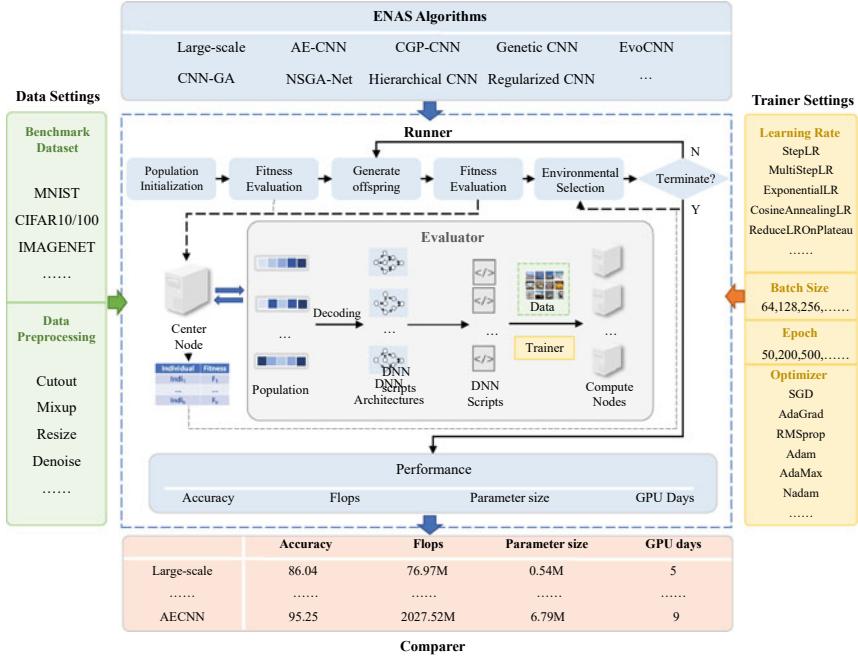


Fig. 9.22 The overall framework of the popular benchmark platform BenchENAS [60]

between ENAS algorithms, several challenges still need to be properly addressed. Firstly, the current inclusion of ENAS algorithms in BenchENAS is limited, and future platforms should aim to incorporate a more diverse and representative set of well-implemented ENAS algorithms. Secondly, though the efficient evaluator in BenchENAS has significantly reduced the computational resource budget already, it does not fundamentally address the problem of high computational resource consumption in ENAS algorithms. To address this problem, it is crucial to incorporate more efficient evaluation methods, which is also a challenge and needs to be investigated in the future.

9.4.2 Efficient Evaluation

Although the evaluation techniques introduced in Sect. 9.3.3 have demonstrated their effectiveness, there are still challenges to be addressed regarding the accuracy and efficiency of the evaluation. In order to alleviate these issues, further research is proposed in two key areas.

In particular, the first category of works is similar to those introduced in Sect. 9.3.3. These approaches still require obtaining the fitness of individuals on the fitness eval-

uation dataset directly. For example, Bender et al. [3] designed a supernet to reduce the time for training thousands of individuals to obtain their fitness. The supernet is a set that contains all possible individuals in the population, and each individual is a subset of the supernet. The evaluation cost can be reduced because the individuals are evaluated by inheriting the trained weights after the supernet is trained. However, because the weights in the supernet are heavily coupled, the performance evaluated with inherited weights may be inaccurate sometimes. In addition, another category of work uses proxy models, which are small-scale versions of the architectures to be evaluated. For instance, Lu et al. [38] construct proxy models of individuals by reducing the number of layers and channels in architectures. The efficiency is improved because training these proxy models costs very little time. However, the prediction accuracy of the performance values is compromised in this case, as the proxy models often do not represent the actual performance of the original architecture.

On the other hand, another category of methods, called performance predictor, utilizes an indirect approach to estimate the performance values of individuals. In particular, the performance predictor is often formulated as a regression model, which can predict the performance of the individuals without training them on the training dataset. The performance predictors can be divided into two categories: performance predictors based on the learning curve and end-to-end performance predictors. In the first category, the final performance is predicted based on the performance of the first several training epochs [39]. In the second category, the end-to-end performance predictor can predict the performance of an architecture without any prior knowledge about the performance of this architecture. To obtain this kind of performance predictor, a certain number of architectures with their corresponding performance values are needed to train the predictor in advance [8]. This process is time-consuming and resource-intensive, defeating the original purpose of a performance predictor to reduce time and resource consumption.

In summary, efficient evaluation techniques are still facing bottlenecks in terms of accuracy and efficiency, and more approaches need to be proposed to address these issues in the future.

9.4.3 *Resource-Constraint ENAS*

In recent years, significant advancements have been made in developing high-performance DNNs that have achieved remarkable results. There is a common belief that “the deeper, the better”, meaning that deeper models often perform better than shallower ones [54]. Therefore, researchers focus on developing deeper DNNs to achieve better accuracy. However, the explosive growth of edge devices such as mobile phones, wearables, and robotic devices in the current Internet of Things (IoT) era has driven the research to quest machine learning platforms that not only prioritize accuracy but also optimize storage and computational requirements. For these IoT devices, the primary focus is on the computational resources that the model will consume, not the optimal performance that the model will achieve. Thus, many

DNN models with excellent performance cannot be directly applied to these IoT devices because of their computational complexity. As a result, designing models under the resource constraints of IoT devices has become an advanced research topic.

There are two ways to design such models: manually with domain expertise and automatically using the NAS method. Manual design focused on lightweight models like MobileNet [22] and ShuffleNet [66] for CNNs, which perform well on specific datasets but struggle with new datasets. Multiobjective optimization methods are commonly used in NAS to design lightweight DNN models [20] automatically. The multiobjective EC can simultaneously optimize competing objectives, such as accuracy and other important objectives, under limited resources. For example, NSGA-Net [37] is a classical ENAS algorithm that applies the nondominated sorting genetic algorithm (NSGA-II) to optimize both accuracy and complexity. Moreover, to address the problem that multiobjective EC cannot find a feasible solution under the specific constraint, Li et al. proposed GA-based CH-CNN [31], which automatically designs CNN architectures and achieves state-of-the-art performance.

However, there are still some challenges on this topic. First, there are various types of DNNs, including DBNs, RNNs, and GNNs, each with significant differences. There is no universal constraint optimization method for all networks. Second, real-world applications often require constraints on multiple metrics, such as energy consumption, spectrum efficiency, and latency. Finally, different hardware platforms have preferences for different network architectures, so it is necessary to consider the specific constraints of each hardware platform.

9.4.4 Interpretability of ENAS

Generally, neural networks are recognized as black-box models with weak interpretability. This shortcoming greatly limits the application of neural networks in some crucial fields, such as security, health, and finance, because these fields have high requirements for interpretability. Although there were some attempts to visualize the feature extraction process of the neural networks [64], the effect/impact is limited. The process is still uninterpretable due to large quantities of the learned features [15].

In order to obtain interpretable solutions, in the ENAS community, GP is a commonly adopted technique, which has long been praised for its outstanding ability to design interpretable models. Specifically, GP is a kind of evolutionary method for automatically designing tree-structured models [27]. Commonly, the models obtained by GP algorithms are much simpler than neural networks, and their tree-like structures also further enhance their interpretability.

Consequently, some works [4, 15, 16] utilized GP to design neural networks automatically. Furthermore, they also provided some methods to analyze the interpretability of the evolved solutions. For example, Evans et al. [15] explained that the convolution filters in their evolved solutions played the role of edge detection in the JAFFE dataset [7], and the large presence of white color in the output result-

ing from the convolution operation could help the classification. In their subsequent work [16], they also explained that the minimum value of a specific area in the hand image extracted by the aggregation function could be used to determine whether the hand is open or closed. Additionally, Bi et al. [4] visualized the features after each function of the searched model. Their research revealed that each branch of the tree-like solutions can generate different salient features which are discriminative for face classification.

Despite their interpretability, most efforts are limited to generating shallow neural networks for simple tasks whose feature number is relatively small. It is hard to transfer existing methods to complex tasks with hundreds of input variables for four reasons: Firstly, when facing complex tasks, the flexible representation of GP tends to generate overcomplex models that overfit the training data, and the performance of the solutions may degrade. Secondly, complex tasks require larger search spaces containing redundant expressions, and GP often leads to unnecessarily large solutions which are difficult to analyze. Thirdly, the search space grows exponentially as the dimension of the input variables increases, resulting in extremely high computational complexity. Last but not the least, a few GP-based methods can fully use the GPUs for acceleration, making the algorithms time-consuming.

9.4.5 *Theoretical Analysis of ENAS*

Despite the success of ENAS, there is still a significant lack of research on theoretical foundations. In the last two decades, there has been a significant increase in EC theoretical studies [67], particularly on the runtime of EAs, which represents the average computational complexity and is thus a core theoretical issue in randomized search heuristics. Most runtime studies begin by analyzing different mutation operators on pseudo-Boolean functions [10]. Furthermore, to improve the ability of EA global exploration, researchers have theoretically demonstrated the importance of the crossover operator in this regard. Specifically, recent works [11] have proven that crossover can provide a noticeable speedup on classical problems like OneMax and Jump_k . Whatever the runtime study goal, the idea behind the above runtime proof is a mathematical evaluation of the runtime bounds using well-known tools, e.g., Markov chain, Chernoff, hitting time, and takeover time.

Most of the studies about EC runtime analysis are based on solving classical benchmark problems. However, as a complex (mostly combinatorial) optimization problem, ENAS has more bottlenecks that must be considered first. One notable bottleneck is the lack of benchmark functions that are crucial in evaluating and comparing optimization algorithms. There is an urgent need to propose insightful benchmark functions tailored to the ENAS. In addition, since no work focuses on the theory of ENAS, bridging the gap between the ENAS algorithm and theoretical research tools and methods is another urgent need. Various works have been devoted to the general EAs runtime analysis methodologies [67], including the fitness-level method, the switch analysis, and drift method. By leveraging these methodologies,

future research can explore the theoretical analysis of ENAS algorithms. In a nutshell, theoretical research on ENAS is a feasible and significant topic that can advance the development of ENAS and accelerate its acceptance by the general machine learning community.

9.5 Conclusions

This chapter provides a comprehensive introduction to ENAS from four aspects. In the first section, we showed that it is a trend to design neural architectures automatically. To address the problem, NAS is proposed. Compared with RL-based NAS and gradient-based NAS, ENAS is an outstanding method because it requires neither a prior construction of the supernet nor domain-rich expertise. In the second section, to begin with, ENAS was detailed with its three core components: search space, search strategy, and performance evaluation. After that, a step-by-step example was given, following the standard process of population initialization, population updating, and fitness evaluation. In the third section, state-of-the-art ENAS techniques were discussed from the perspective of its core components. In the fourth section, advanced topics of ENAS were addressed, including benchmarks of ENAS, efficient evaluation, resource-constraint ENAS, interpretability of ENAS, and theoretical analysis of ENAS.

The success of state-of-the-art ENAS methods has sparked discussions on various advanced topics, aiming to advance the field further. These topics include the establishment of a benchmark platform to facilitate fair comparisons of ENAS techniques, the development of efficient evaluation techniques for efficient and accurate assessments, the exploration of resource-constrained ENAS algorithms tailored for IoT devices, the investigation of interpretability methods for complex tasks in ENAS, and the theoretical analysis of ENAS algorithms. Diving into these advanced topics will play a crucial role in accelerating the development of ENAS algorithms.

References

1. Assunção, F., Correia, J., Conceição, R., Pimenta, M.J.M., Tomé, B., Lourenço, N., Machado, P.: Automatic design of artificial neural networks for gamma-ray detection. *IEEE Access* **7**, 110531–110540 (2019)
2. Bäck, T., Fogel, D.B., Michalewicz, Z.: *Handbook of evolutionary computation*. Release **97**(1), B1 (1997)
3. Bender, G., Kindermans, P.J., Zoph, B., Vasudevan, V., Le, Q.: Understanding and simplifying one-shot architecture search. In: International Conference on Machine Learning, pp. 550–559. PMLR (2018)
4. Bi, Y., Xue, B., Zhang, M.: An evolutionary deep learning approach using genetic programming with convolution operators for image classification. In: 2019 IEEE Congress on Evolutionary Computation (CEC), pp. 3197–3204. IEEE (2019)

5. Broni-Bediako, C., Murata, Y., Mormille, L.H., Atsumi, M.: Evolutionary NAS with gene expression programming of cellular encoding. In: 2020 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 2670–2676. IEEE (2020)
6. Chen, Z., Zhou, Y., Huang, Z.: Auto-creation of effective neural network architecture by evolutionary algorithm and resnet for image classification. In: 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC), pp. 3895–3900. IEEE 2019
7. Cheng, F., Jiangsheng, Yu., Xiong, H.: Facial expression recognition in Jaffe dataset based on Gaussian process classification. *IEEE Trans. Neural Netw.* **21**(10), 1685–1690 (2010)
8. Deng, B., Yan, J., Lin, D.: Peephole: Predicting network performance before training (2017). [arXiv:1712.03351](https://arxiv.org/abs/1712.03351)
9. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: pre-training of deep bidirectional transformers for language understanding (2018). [arXiv:1810.04805](https://arxiv.org/abs/1810.04805)
10. Doerr, B., Doerr, C.: Theory of parameter control for discrete black-box optimization: Provable performance gains through dynamic parameter choices. *Theory of Evolutionary Computation*, pp. 271–321 (2020)
11. Doerr, B., Happ, E., Klein, C.: Crossover can provably be useful in evolutionary computation. In: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, pp. 539–546 (2008)
12. Dong, X., Yang, Y.: Nas-bench-201: extending the scope of reproducible neural architecture search. In: International Conference on Learning Representations (2019)
13. Elskens, T., Metzen, J.H., Hutter, F.: Simple and efficient architecture search for convolutional neural networks (2017). [arXiv:1711.04528](https://arxiv.org/abs/1711.04528)
14. Elskens, T., Metzen, J.H., Hutter, F.: Neural architecture search: a survey. *J. Mach. Learn. Res.* **20**(1), 1997–2017 (2019)
15. Evans, B., Al-Sahaf, H., Xue, B., Zhang, M.: Evolutionary deep learning: a genetic programming approach to image classification. In: 2018 IEEE Congress on Evolutionary Computation (CEC), pp. 1–6. IEEE (2018)
16. Evans, B.P., Al-Sahaf, H., Xue, B., Zhang, M.: Genetic programming and gradient descent: A memetic approach to binary image classification (2019). [arXiv:1909.13030](https://arxiv.org/abs/1909.13030)
17. Fan, Z., Wei, J., Zhu, G., Mo, J., Li, W.: Evolutionary neural architecture search for retinal vessel segmentation (2020). [arXiv:2001.06678](https://arxiv.org/abs/2001.06678)
18. Floreano, D., Dürr, P., Mattiussi, C.: Neuroevolution: from architectures to learning. *Evol. Intel.* **1**(1), 47–62 (2008)
19. Fujino, S., Naoki, M., Matsumoto, K.: Deep convolutional networks for human sketches by means of the evolutionary deep learning. In: 2017 Joint 17th World Congress of International Fuzzy Systems Association and 9th International Conference on Soft Computing and Intelligent Systems (IFSA-SCIS), pp. 1–5. IEEE (2017)
20. Gunantara, N.: A review of multi-objective optimization: methods and its applications. *Cogent Eng.* **5**(1), 1502242 (2018)
21. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
22. Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H.: Mobilenets: efficient convolutional neural networks for mobile vision applications (2017). [arXiv:1704.04861](https://arxiv.org/abs/1704.04861)
23. Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q.: Densely connected convolutional networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4700–4708 (2017)
24. Irwin-Harris, W., Sun, Y., Xue, B., Zhang, M.: A graph-based encoding for evolutionary convolutional neural network architecture design. In: 2019 IEEE Congress on Evolutionary Computation (CEC), pp. 546–553. IEEE (2019)
25. Johner, F.M., Wassner, J.: Efficient evolutionary architecture search for CNN optimization on gtsrb. In: 2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA), pp. 56–61. IEEE (2019)

26. Kaelbling, L.P., Littman, M.L., Moore, A.W.: Reinforcement learning: a survey. *J. Artif. Intell. Res.* **4**, 237–285 (1996)
27. Koza, J.R.: Genetic programming as a means for programming computers by natural selection. *Stat. Comput.* **4**(2), 87–112 (1994)
28. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. *Handbook Syst. Autoimmune Dis.* **1**(4) (2009)
29. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. *Commun. ACM* **60**(6), 84–90 (2017)
30. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. *Nature* **521**(7553), 436–444 (2015)
31. Li, S., Sun, Y., Yen, G.G., Zhang, M.: Automatic design of convolutional neural network architectures under resource constraints. *IEEE Trans. Neural Netw. Learn. Syst.* (2021)
32. Liu, H., Simonyan, K., Yang, Y.: Darts: Differentiable architecture search (2018). [arXiv:1806.09055](https://arxiv.org/abs/1806.09055)
33. Liu, J., Gong, M., Miao, Q., Wang, X., Li, H.: Structure learning for deep neural networks based on multiobjective optimization. *IEEE Trans. Neural Netw. Learn. Syst.* **29**(6), 2450–2463 (2017)
34. Liu, P., El Basha, M.D., Li, Y., Xiao, Y., Sanelli, P.C., Fang, R.: Deep evolutionary networks with expedited genetic algorithms for medical image denoising. *Med. Image Anal.* **54**, 306–315 (2019)
35. Liu, Y., Sun, Y., Xue, B., Zhang, M., Yen, G.G., Tan, K.C.: A survey on evolutionary neural architecture search. *IEEE Trans. Neural Netw. Learn. Syst.* (2021)
36. Loni, M., Sinaei, S., Zoljodi, A., Daneshtalab, M., Sjödin, M.: Deepmaker: a multi-objective optimization framework for deep neural networks in embedded systems. *Microprocess. Microsyst.* **73**, 102989 (2020)
37. Lu, Z., Whalen, I., Boddeti, V., Dhebar, Y., Deb, K., Goodman, E., Banzhaf, W.: NSGA-Net: neural architecture search using multi-objective genetic algorithm. In: Proceedings of the Genetic and Evolutionary Computation conference, pp. 419–427 (2019)
38. Lu, Z., Whalen, I., Dhebar, Y., Deb, K., Goodman, E.D., Banzhaf, W., Boddeti, V.N.: Multi-objective evolutionary design of deep convolutional neural networks for image classification. *IEEE Trans. Evol. Comput.* **25**(2), 277–291 (2020)
39. Rawal, A., Miikkulainen, R.: From nodes to networks: evolving recurrent neural networks (2018). [arXiv:1803.04439](https://arxiv.org/abs/1803.04439)
40. Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y.L., Tan, J., Le, Q.V., Kurakin, A.: Large-scale evolution of image classifiers. In: International Conference on Machine Learning, pp. 2902–2911. PMLR (2017)
41. Ruder, S.: An overview of gradient descent optimization algorithms (2016). [arXiv:1609.04747](https://arxiv.org/abs/1609.04747)
42. Russakovsky, O., Deng, J., Hao, S., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al.: Imagenet large scale visual recognition challenge. *Int. J. Comput. Vision* **115**(3), 211–252 (2015)
43. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition (2014). [arXiv:1409.1556](https://arxiv.org/abs/1409.1556)
44. So, D., Le, Q., Liang, C.: The evolved transformer. In: International Conference on Machine Learning, pp. 5877–5886. PMLR (2019)
45. Song, D., Chang, X., Jia, X., Chen, Y., Chunjing, X., Wang, Y.: Efficient residual dense block search for image super-resolution. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 34, pp. 12007–12014 (2020)
46. Stanley, K.O., Bryant, B.D., Miikkulainen, R.: Real-time neuroevolution in the NERO video game. *IEEE Trans. Evol. Comput.* **9**(6), 653–668 (2005)
47. Stanley, K.O., D'Ambrosio, D.B., Gauci, J.: A hypercube-based encoding for evolving large-scale neural networks. *Artif. Life* **15**(2), 185–212 (2009)
48. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. *Evol. Comput.* **10**(2), 99–127 (2002)
49. Sun, Y., Xue, B., Zhang, M., Yen, G.G.: A particle swarm optimization-based flexible convolutional autoencoder for image classification. *IEEE Trans. Neural Netw. Learn. Syst.* **30**(8), 2295–2309 (2018)

50. Sun, Y., Xue, B., Zhang, M., Yen, G.G.: Completely automated CNN architecture design based on blocks. *IEEE Trans. Neural Netw. Learn. Syst.* **31**(4), 1242–1254 (2019)
51. Sun, Y., Xue, B., Zhang, M., Yen, G.G.: Evolving deep convolutional neural networks for image classification. *IEEE Trans. Evol. Comput.* **24**(2), 394–407 (2019)
52. Sun, Y., Xue, B., Zhang, M., Yen, G.G., Lv, J.: Automatically designing CNN architectures using the genetic algorithm for image classification. *IEEE Trans. Cybern.* **50**(9), 3840–3854 (2020)
53. Sun, Y., Yen, G.G., Zhang, M.: Internet protocol based architecture design. In: *Evolutionary Deep Neural Architecture Search: Fundamentals, Methods, and Recent Advances*, pp. 181–192. Springer, Cham (2023)
54. Sze, V., Chen, Y.H., Yang, T.J., Emer, J.S.: Efficient processing of deep neural networks: a tutorial and survey. *Proc. IEEE* **105**(12), 2295–2329 (2017)
55. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9 (2015)
56. Wang, B., Sun, Y., Xue, B., Zhang, M.: Evolving deep convolutional neural networks by variable-length particle swarm optimization for image classification. In: *2018 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8. IEEE (2018)
57. Wang, B., Xue, B., Zhang, M.: Particle swarm optimisation for evolving deep neural networks for image classification by evolving and stacking transferable blocks. In: *2020 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8. IEEE (2020)
58. Wu, T., Shi, J., Zhou, D., Lei, Y., Gong, M.: A multi-objective particle swarm optimization for neural networks pruning. In: *2019 IEEE Congress on Evolutionary Computation (CEC)*, pp. 570–577. IEEE (2019)
59. Xie, L., Yuille, A.: Genetic CNN. In: *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1379–1388 (2017)
60. Xie, X., Liu, Y., Sun, Y., Yen, G.G., Xue, B., Zhang, M.: Benchenas: a benchmarking platform for evolutionary neural architecture search. *IEEE Trans. Evol. Comput.* (2022)
61. Yang, Z., Wang, Y., Chen, X., Shi, B., Xu, C., Xu, C., Tian, Q., Xu, C.: Cars: continuous evolution for efficient neural architecture search. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1829–1838 (2020)
62. Yao, X.: Evolving artificial neural networks. *Proc. IEEE* **87**(9), 1423–1447 (1999)
63. Ying, C., Klein, A., Christiansen, E., Real, E., Murphy, K., Hutter, F.: Nas-bench-101: towards reproducible neural architecture search. In: *International Conference on Machine Learning*, pp. 7105–7114. PMLR (2019)
64. Zeiler, M.D., Fergus, R.: Visualizing and understanding convolutional networks. In: *European Conference on Computer Vision*, pp. 818–833. Springer (2014)
65. Zhang, M.: Evolutionary deep learning for image analysis. Talk at World Congress on Computational Intelligence (WCCI) (2018). Published on July 2, 2020
66. Zhang, X., Zhou, X., Lin, M., Sun, J.: Shufflenet: an extremely efficient convolutional neural network for mobile devices. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 6848–6856 (2018)
67. Zhou, Z.-H., Yang, Yu., Qian, C.: *Evolutionary Learning: Advances in Theories and Algorithms*. Springer, Singapore (2019)
68. Zoph, B., Le, Q.V.: Neural architecture search with reinforcement learning (2016). [arXiv:1611.01578](https://arxiv.org/abs/1611.01578)
69. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8697–8710 (2018)

Chapter 10

Evolutionary Generative Models



João Correia, Francisco Baeta, and Tiago Martins

Abstract In the last decade, generative models have seen widespread use for their ability to generate diverse artefacts in an increasingly simple way. Historically, the use of evolutionary computation as a generative model approach was dominant, and recently, as a consequence of the rise in popularity and amount of research being conducted in artificial intelligence, the application of evolutionary computation to generative models has broadened its scope to encompass more complex machine learning approaches. Therefore, it is opportune to propose a term capable of accommodating all these models under the same umbrella. To address this, we propose the term *evolutionary generative models* to refer to generative approaches that employ any type of evolutionary algorithm, whether applied on its own or in conjunction with other methods. In particular, we present a literature review on this topic, identifying the main properties of evolutionary generative models and categorising them into four different categories: *evolutionary computation without machine learning*, *evolutionary computation aided by machine learning*, *machine learning aided by evolutionary computation* and *machine learning evolved by evolutionary computation*. Therefore, we systematically analyse a selection of prominent works concerning evolutionary generative models. We conclude by addressing the most relevant challenges and open problems faced by current evolutionary generative models and discussing where the topic's future is headed.

J. Correia (✉) · F. Baeta · T. Martins

Department of Informatics Engineering, Centre for Informatics and Systems of the University of Coimbra, University of Coimbra, 3004-531 Coimbra, Portugal

e-mail: jncor@dei.uc.pt

F. Baeta

e-mail: fjrbeta@dei.uc.pt

T. Martins

e-mail: tiagofm@dei.uc.pt

10.1 Introduction

In recent years, the growth of interest in artificial intelligence research and development has sparked the development of increasingly more complex systems in many fields. In particular, generative models have seen widespread use in the fields of machine learning and evolutionary computation for their ability to generate new instances that follow the probabilistic distribution of a set of pre-existing ones. Due to their recent advancements and impressive results, generative models have become a hot topic. Their outputs can be diverse and applied to a broad range of application domains, e.g. image, music, text, engineering and game design.

Within generative machine learning, the proposal of generative adversarial neural networks (GANs) in 2014 by Goodfellow et al. [72] dramatically changed the landscape in research on generative models by proposing an effective adversarial way of training them. Consequently, GANs quickly gained acclamation to the extent of being coined as “the most interesting idea in the last 10 years in Machine Learning” by Yann LeCun. However, before the advent of GANs, evolutionary computation was the *de facto* approach employed in generative modelling [171]. With the rapid surge of interest around GANs and other deep learning generative approaches, most machine learning-based approaches surpassed what most evolutionary computation generative approaches and models have achieved. This was the case until the last few years when a paradigm shift happened propelled by the maturity of the field of evolutionary machine learning, where methods from evolutionary computation and machine learning are combined as a unified solution.

This chapter surveys the work on evolutionary generative models. We propose this term to identify generative models that use evolutionary computation in any part of the generative process. This definition of evolutionary generative model means that a substantial body of the surveyed work includes plain evolutionary computation approaches that date back to the 1980s, although authors such as Richard Dawkins and Karl Sims have never described their work as evolutionary generative models. Given the increasing adoption of evolutionary machine learning techniques by the research community as well as the rise of the popularity of generative models, the study of evolutionary generative models is ever more frequent in the literature. Despite this, to the best of our knowledge, there is still no comprehensive review focussed on the existing literature for evolutionary generative models. Therefore, in this chapter, we survey the existing literature related to this type of model and analyse existing work from both a historical and scientific point of view.

During the survey, we found that the evolutionary generative model can be divided into four different categories based on the role of machine learning in the generative process: *evolutionary computation without machine learning*, *evolutionary computation aided by machine learning*, *machine learning aided by evolutionary computation*, and *machine learning evolved by evolutionary computation*. The division along the four categories is central to the structure and narrative of this chapter.

The document outline is as follows. In Sect. 10.2, we introduce the core concepts and definitions regarding evolutionary generative models and outline the main properties of these models, thus providing a knowledge base for the chapter and

defining its scope. Then, in Sect. 10.3, we propose a taxonomy to classify evolutionary generative models. Next, in Sect. 10.4, we provide a brief historical overview to contextualise research on this type of generative model, describing the most relevant works and presenting a visual timeline. The following four Sects. 10.5, 10.6, 10.7 and 10.8, analyse the collected papers of the four defined categories. Then, in Sect. 10.9, we overview open problems in the area of evolutionary generative models and identify new research opportunities. Finally, Sect. 10.10 lays the closing remarks by enumerating the main contributions, opportunities and the conclusion of this chapter.

10.2 Fundamentals

We define an *evolutionary generative model* as a generative model that employs evolutionary computation in any part of the generative process. Although the *evolutionary* part of this definition is explicit, we believe that *generative model* may have different interpretations.

When we look at the broader concept of generative model and generative modelling, we can encounter different definitions. For instance, in machine learning, generative models are considered models capable of generating new data instances. More formally, given a set of data instances X and a set of labels Y , generative models capture the joint probability $p(X, Y)$, or just $p(X)$ if there are no labels. Therefore, a generative model describes how a dataset is generated in terms of a probabilistic model that can create instances from a distribution similar to the training data instances. In statistics, a generative model is a statistical model of the joint probability distribution $p(X, Y)$ of a given observable variable X and target variable Y . In graphic design, a generative process is concerned with the creation of an algorithm, or model that produces multiple designs that instantiate a concept. Considering these definitions and our analysis of varied works in this area, we can define a common ground in what concerns the specifications that characterise generative models.

A generative model typically implements an algorithmic process that generates outputs that serve a certain objective or concept. This way, the generated outputs tend to follow a given distribution within an objective class, thus forming a space where the outputs share common characteristics. In addition, the generative model can often receive input information that influences the generative process and consequently changes the outputs. This generative process can be implemented using evolutionary computation or enhanced by it.

During our analysis of existing evolutionary generative models, we identified five properties which we consider relevant to study in each model: application domain, evolutionary algorithm, population, representation and fitness. These properties are outlined in Fig. 10.1 and described in the following paragraphs.

Domain corresponds to the application domain of the model being analysed: *image*, *music*, *text* or *other* applications such as engineering, architecture and even less common disciplines such as dance [59] and culinary recipes [152]. It can be

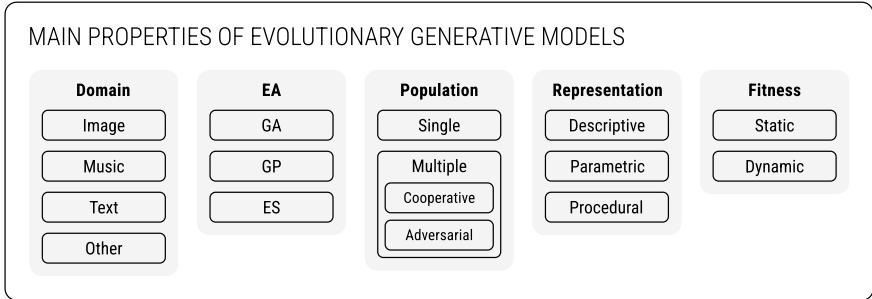


Fig. 10.1 Main properties of evolutionary generative models

argued that this property pertains to the applications of the model and not to the model itself. However, as far as evolutionary computation is concerned, we found that most algorithms follow specific representations and parameterisations according to the application domain.

Evolutionary Algorithm establishes which evolutionary paradigm is employed by the evolutionary generative model, namely, genetic algorithms, genetic programming or evolutionary strategies.

Population relates to the fact of the generative model having a *single* or *multiple* populations of individuals being evolved. Within approaches with *multiple* populations, we have two different types: *cooperative* and *adversarial*. In a *cooperative* setup, multiple populations collaborate to achieve a common goal, where typically each population specialises in a specific task or sub-problem, and their combined efforts contribute to the overall solution. On the other hand, in an *adversarial* setup, different populations are evolved and compete with each other.

Representation refers to the structure of the genotype. Based on the categorisation of evolutionary art systems by Machado [123], we generalised the categorisation and divided the representations used in the analysed works into three main types: *descriptive*, *parametric* and *procedural*. In a *descriptive* representation, we evolve values that directly define the properties of each individual. In a *parametric* representation, we also evolve values, but they are used as input parameters for a model that generates each individual. This way, individuals are defined explicitly by the parametric model and the properties of these individuals are controlled by varying the parameter values of the model, which are encoded in the genotype. In a *procedural* representation, on the other hand, we evolve a sequence of procedures or rules, which can be seen as a model that is then used to generate the individual. In summary, while *descriptive* and *parametric* representations are data oriented, as they encode values, *procedural* representations are process oriented, since the evolutionary algorithm evolves, for instance, mathematical expressions, networks or rules.

Fitness evaluation in the generative process is crucial, and according to the literature, there are several ways of doing it and it depends on many factors. However, in this chapter, we are mainly interested in whether the fitness evaluation changes throughout the evolutionary process or not. Thus, we divided the surveyed works

into two categories: *static* and *dynamic*. In the case of *static* fitness, if the individual is the same, no change should occur to its fitness throughout the evolutionary process. As for *dynamic* fitness, we consider cases where the output of the evaluation may vary during the evolutionary process, i.e. when different fitness values can be assigned to the same individual in different moments of the evolutionary process. Among the approaches that implement dynamic fitness, we can find, for instance, those that employ user feedback to guide the evolutionary process, i.e. interactive evolutionary computation approaches. We consider that fitness assignment based on user feedback, i.e. interactive evolutionary computation, is *dynamic*.

All these properties helped to structure and sort the analysed works, as can be seen later in this chapter, in Tables 10.1 and 10.4, which contain the analysed evolutionary generative models.

Table 10.1 Examples of evolutionary generative models categorised as *evolutionary computation without machine learning*

Domain	Year	Authors	EA	Population	Represent.	Fitness
Image	1986	[52] Dawkins	GA	Single	Procedural	Dynamic
Image	1991	[185] Sims	GP	Single	Procedural	Dynamic
Image	1991	[187] Smith	GA	Single	Procedural	Dynamic
Image	1992	[199] Todd and Latham	GP	Single	Procedural	Dynamic
Image	1993	[11] Baker and Seltzer	GA	Single	Descriptive	Dynamic
Image	1993	[186] Sims	GP	Single	Procedural	Dynamic
Image	1995	[74] Graf and Banzhaf	GA	Single	Descriptive	Dynamic
Image	1996	[5] Angeline	GA	Single	Parametric	Dynamic
Image	1996	[214] World	GP	Single	Procedural	Dynamic
Image	1997	[157] Nishio et al.	GA	Single	Parametric	Dynamic
Image	1998	[76] Greenfield	GP	Single	Procedural	Dynamic
Image	1999	[14] Bentley and Kumar	GA/GP	Single	Descrip./Proced.	Static
Image	1999	[207] Unemi	GP	Single	Procedural	Dynamic
Image	2000	[77] Greenfield	GP	Single	Procedural	Dynamic
Image	2000	[104] Lewis	GA	Single	Procedural	Dynamic
Image	2000	[124] Machado and Cardoso	GP	Single	Procedural	Dynamic
Image	2001	[31] Chapuis and Lutton	GP	Single	Procedural	Dynamic
Image	2002	[125] Machado and Cardoso	GP	Single	Procedural	Dynamic
Image	2002	[172] Rooke	GP	Single	Procedural	Dynamic
Image	2003	[7] Aupetit et al.	GA	Single	Parametric	Dynamic
Image	2004	[78] Greenfield	GP	Multi. (advers.)	Procedural	Dynamic
Image	2005	[55] DiPaola	GP	Single	Procedural	Static
Image	2005	[58] Draves	GA	Single	Procedural	Dynamic
Image	2005	[79] Greenfield	GA	Single	Parametric	Static
Image	2006	[6] Ashlock	ES	Single	Parametric	Static
Image	2007	[56] DiPaola and Gabora	GP	Single	Procedural	Static
Image	2007	[80] Hart	GP	Single	Procedural	Dynamic/Static

(continued)

Table 10.1 (continued)

Domain	Year	Authors	EA	Population	Represent.	Fitness
Image	2008	[35] Collomosse	GA	Single	Parametric	Dynamic
Image	2008	[129] Machado and Graça	GP	Single	Procedural	Dynamic
Image	2008	[154] Neufeld et al.	GP	Single	Procedural	Static
Image	2008	[208] Ventrella	GA	Single	Procedural	Dynamic
Image	2010	[131] Machado et al.	GP	Single	Procedural	Dynamic/Static
Image	2011	[24] Brown et al.	GA	Single	Parametric	Static
Image	2011	[169] Reynolds	GP	Single	Procedural	Static
Image	2012	[15] Bergen and Ross	GA	Single	Descriptive	Dynamic/Static
Image	2012	[37] Colton	GA	Single	Parametric	Static
Image	2012	[54] den Heijer and Eiben	GP	Single	Procedural	Static
Image	2012	[75] Greenfield	GP	Single	Procedural	Static
Image	2012	[132] Machado and Pereira	GA	Single	Parametric	Dynamic
Image	2013	[16] Bergen and Ross	GP	Single	Procedural	Static
Image	2014	[126] Machado and Correia	GP	Single	Procedural	Dynamic/Static
Image	2016	[170] Rodrigues et al.	GA	Single	Parametric	Dynamic
Image	2018	[121] Maçãs et al.	GA	Single	Parametric	Dynamic
Image	2018	[217] Zoric and Gambäck	GA	Single	Procedural	Static
Image	2020	[50] Cunha et al.	GA	Single	Parametric	Dynamic
Image	2020	[161] Parente et al.	GA	Single	Parametric	Dynamic
Image	2020	[167] Rebelo et al.	GA	Single	Parametric	Static
Image	2021	[9] Baeta et al.	GP	Single	Procedural	Static
Music	1991	[91] Horner and Goldberg	GA	Single	Procedural	Static
Music	1994	[18] Biles	GA	Multi. (colab.)	Descriptive	Dynamic
Music	1994	[102] Laine and Kuuskankare	GP	Single	Procedural	Static
Music	1994	[92] Horowitz	GA	Single	Parametric	Dynamic
Music	1994	[188] Spector and Alpern	GP	Single	Procedural	Static
Music	1995	[95] Jacob	GA	Single	Descriptive	Dynamic
Music	1999	[196] Thywissen	GA	Single	Parametric	Dynamic
Music	2000	[151] Moroni et al.	GA	Single	Descriptive	Static
Music	2002	[53] de la Puente et al.	GP	Single	Procedural	Static
Music	2003	[69] Gartland-Jones	GA	Single	Parametric	Static
Music	2003	[138] Manaris et al.	GP	Single	Descriptive	Static
Music	2007	[19] Bilotta et al.	GA	Single	Parametric	Static
Music	2007	[98] Khalifa et al.	GA	Single	Descriptive	Static
Music	2009	[168] Reddin et al.	GP	Single	Procedural	Static
Music	2010	[165] Prisco et al.	GA	Single	Descriptive	Static
Music	2010	[184] Shao et al.	GP	Single	Procedural	Dynamic
Music	2011	[57] Donnelly and Sheppard	GA	Single	Descriptive	Static
Music	2011	[147] McDermott and O'Reilly	GP	Single	Procedural	Static
Music	2012	[61] Eigenfeldt and Pasquier	GA	Single	Descriptive	Dynamic
Music	2012	[97] Kaliakatsos-Papakostas et al.	GP	Single	Procedural	Dynamic

(continued)

Table 10.1 (continued)

Domain	Year	Authors	EA	Population	Represent.	Fitness
Music	2012	[122] MacCallum et al.	GP	Single	Procedural	Dynamic
Music	2012	[164] Pirnia and McCormack	GP	Single	Procedural	Static
Music	2015	[85] Hofmann	GP	Single	Procedural	Static
Music	2015	[101] Kunimatsu et al.	GP	Single	Procedural	Static
Music	2015	[115] Loughran et al.	GP	Single	Procedural	Static
Music	2015	[193] Sulyok et al.	GP	Single	Procedural	Static
Music	2016	[84] Hickinbotham and Stepney	GP	Single	Procedural	Dynamic
Music	2016	[116] Loughran et al.	GP	Single	Procedural	Static
Music	2016	[119] Loughran and O'Neill	GP	Single	Procedural	Static
Music	2016	[180] Scirea et al.	GA	Multi. (colab.)	Descriptive	Static
Music	2017	[117] Loughran and O'Neill	GP	Single	Procedural	Dynamic
Music	2017	[197] Ting et al.	GA	Single	Descriptive	Static
Music	2018	[159] Olseng and Gambäck	GA	Single	Descriptive	Static
Music	2020	[4] Alvarado et al.	GA	Single	Descriptive	Static
Music	2020	[166] Prisco et al.	GA	Single	Descriptive	Static
Music	2021	[3] Albaracín-Molina et al.	GP	Single	Procedural	Static
Music	2021	[174] Santos et al.	GA	Single	Parametric	Static
Music	2021	[93] Hui Yap et al.	GA	Single	Descriptive	Static
Music	2022	[142] Martínez-Rodríguez	ES	Single	Descriptive	Static
Text	2000	[140] Manurung et al.	GP	Single	Procedural	Static
Text	2003	[139] Manurung	GP	Single	Procedural	Static
Text	2006	[150] Montero and Araki	GA	Single	Descriptive	Dynamic
Text	2007	[82] Hervás et al.	GA	Single	Procedural	Static
Text	2012	[141] Manurung et al.	GA	Single	Procedural	Static
Text	2013	[70] Gervás	GP	Single	Procedural	Static
Text	2015	[202] Tomasic et al.	GA	Single	Procedural	Static
Other	1996	[177] Schnier and Gero	GA	Single	Procedural	Static
Other	2008	[62] Ekárt	GP	Single	Procedural	Static
Other	2008	[66] Frade et al.	GP	Single	Procedural	Dynamic
Other	2010	[26] Browne and Maire	GP	Single	Procedural	Static
Other	2010	[28] Byrne et al.	GP	Single	Procedural	Dynamic
Other	2012	[183] Shaker et al.	GP	Single	Procedural	Static
Other	2014	[29] Cardona et al.	ES	Single	Descriptive	Static
Other	2015	[87] Hoover et al.	GA	Single	Procedural	Dynamic
Other	2016	[114] Lopes et al.	GA	Single	Descriptive	Static
Other	2017	[40] Cook et al.	GA/GP	Multi. (colab.)	Descrip./Proced.	Static
Other	2017	[120] Lucas and Martinho	GA	Single	Descriptive	Static
Other	2017	[153] Muehlbauer et al.	GP	Single	Procedural	Static
Other	2018	[39] Cook and Colton	GA/GP	Multi. (colab.)	Descrip./Proced.	Static

10.3 Taxonomy

We propose a taxonomy to classify evolutionary generative models into four categories according to the existence and role of machine learning in the generative model. The proposed categories are described next.

Evolutionary computation without machine learning—systems where the generative model only uses evolutionary computation with no intervention of any kind of machine learning technique. The modelling is purely based on evolutionary computation and the evaluation typically involves user guidance, hardwired fitness functions or a set of rules and conditions.

Evolutionary computation aided by machine learning—systems that combine evolutionary computation and machine learning, where the utilisation of machine learning enhances the capability of the evolutionary generative model. For instance, using the activation of a machine learning classifier to assign fitness and guide evolution towards outputs that maximise the classifier activation or using a clustering technique to aid in the selection of evolved individuals.

Machine learning aided by evolutionary computation—systems which are machine learning-based and where evolutionary computation is employed to enhance the capability of the machine learning part. For instance, neuroevolution approaches fall into this category; the same applies to approaches that explore the latent space of a GAN to generate suitable outputs or use evolutionary computation instead of gradient optimisation to train or fine-tune generative machine learning models.

Machine learning evolved by evolutionary computation—systems in which evolutionary computation evolves entire machine learning generative models or entire machine learning components. For instance, neuroevolution of augmenting topologies (NEAT) evolves artificial neural networks and their components using a genetic algorithm or the evolution of populations of GANs.

10.4 Historical Overview

This section provides a brief historical overview of key publications in each of the four proposed categories for evolutionary generative models. In particular, in each of the following subsections, we summarise the publication timeline of each category while presenting a table containing several examples and their main properties. Figure 10.2 presents a timeline for the four proposed categories.

10.4.1 Evolutionary Computation Without Machine Learning

This category of evolutionary generative models is the one with the highest volume of publications (please refer to Fig. 10.2), which is explained by the fact that before the popularisation of GANs and neuroevolution, evolutionary computation approaches represented the early instances of evolutionary generative models.

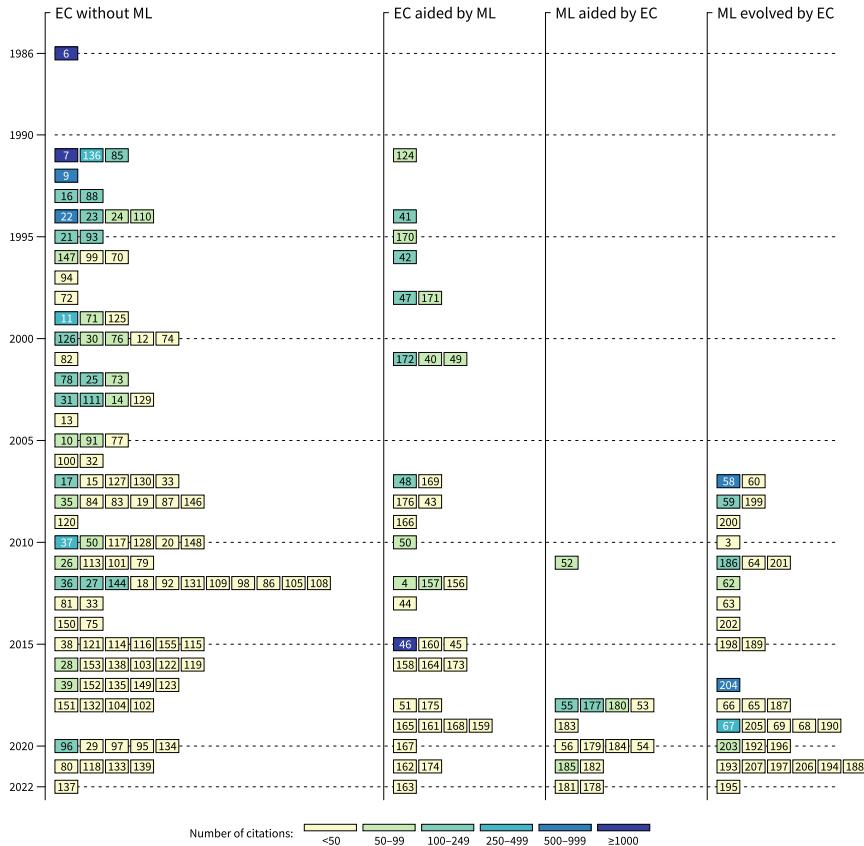


Fig. 10.2 Timeline showing all publications on evolutionary generative models cited in this chapter. Publications are ordered chronologically (oldest at the top and most recent at the bottom) and divided horizontally by category (from left to right: *evolutionary computation without machine learning*, *evolutionary computation aided by machine learning*, *machine learning aided by evolutionary computation* and *machine learning evolved by evolutionary computation*). Each publication is represented with a rectangle containing the corresponding reference number. The colour of each rectangle indicates the number of citations of the corresponding publication according to the scale shown at the bottom of the figure (darker colours indicate a greater number of citations). The number of citations was queried to Google Scholar on June 7, 2023

The biomorphs presented by Dawkins [52] in his book *The Blind Watchmaker*, in 1986, were one of the earliest examples of using evolutionary computation to create images using user interaction, basically setting the start for interactive evolutionary computation. Afterwards, in 1991, Sims [185] employed user-guided evolution to create images, solid textures and animations encoded as Lisp-based expressions of mathematical functions, variables and constants, also referred to as symbolic expressions. The idea of using symbolic expressions in evolutionary algorithms had previously been explored by Koza [100] in 1990 and would eventually give rise to

the nowadays prolific field of genetic programming. Despite gaining popularity later on, the approach of using evolutionary computation for image generation was not yet common in the computer science community at this point in time (see Fig. 10.2). That would change, however, in 1992 with the work by Todd and Latham [199], who popularised evolutionary art in general with their work on the generation of 2D and 3D images and by making it available to the general public. In line with this idea, in 1999, Draves [58] created the collaborative abstract artwork “Electric Sheep” that evolved fractal flames, a generalisation and refinement of the iterative function systems. Several researchers were experimenting with varied representations for evolutionary algorithms. Bentley and Kumar [14] explored the importance of different embryogenesis processes, i.e. the mapping from the genotype representation to phenotype, in the same generative design problem demonstrating the impact of having a direct genotype on phenotype mapping on the final solutions and evolution with distinct solutions using a genetic algorithm or genetic programming.

Given the impact and amount of publications regarding evolutionary computation for image generation, we can conclude that the first half of the 1990s sparked more interest in evolutionary art in general following its appearance. This interest was eventually picked back up around the turn of the millennium, with many works in evolutionary computation being published (see Fig. 10.2). Within this period, it is worth mentioning the work by Lewis [104] concerning aesthetic evolutionary design with network data flows, a direct acyclic graph representation of transformations on geometric primitives for generating specific types of images such as cartoon faces and human figure geometry. Greenfield [78] used the image generation approach of Sims [185] with a bio-inspired coevolutionary fitness scheme, inspired by hosts and parasites dynamics, to instantiate an evolutionary simulation for the generation of aesthetic image tiles. Aupetit et al. [7] created an interactive genetic algorithm to evolve parameters for an ant colony optimisation approach to generate paintings. Hart [80] proposed new mutation operators for expression-based evolutionary image generation and tree alignments schemes for improving the morphing between two individuals, thus enabling a smoother animation process and expanding the possible solutions for the cross-dissolve process between individuals [186]. Genetic programming was used extensively around this time, with a few more notable examples to be mentioned. Di Paola and Gabora [56] tried to evolve a portrait of Charles Darwin using genetic programming, an endeavour that presented many hurdles, such as handling fitness values plateaus which proved difficult to overcome. The work by den Heijer and Eiben [54] used a scalable vector graphics format as a genetic representation of their genetic programming approach for the evolution of abstract images. Outside of expression-based image generation, we should mention the work by Collomesse [35], which performed non-photorealistic rendering by generating images with pre-processed and parameterised brush strokes based on a starting image. To the best of our knowledge, although many genetic programming approaches have done it before [185], this work represents one of the first efforts in using genetic algorithms to evolve filters from existing images. In 2010, Machado et al. [131] presented an evolutionary engine that allows the evolution of context-free grammars using a graph representation and operators.

The early use of genetic algorithms in the generation and evolution of music was explored by Jacob [95], who employed an interactive system to assign fitness in music composition, and by Biles [18], who introduced the framework Genjam to generate Jazz solos. It is important to note that these first studies in music generation mainly focussed on the evolution of melodic progressions. One of the first studies specifically targeting metric and rhythm was performed by Horowitz [92] in 1994. Furthermore, around this period, the use of expression-based methods to create music was introduced by Spector and Alpern [188] with a system to create Bebop music using genetic programming. These first efforts were later picked up and extended upon, which saw the introduction of new frameworks and techniques. It is also worth mentioning the work by de la Puente et al. [53] in the early 2000s, which, to the best of our knowledge, corresponds to the first application of a grammatical evolution algorithm to music composition. Moreover, Donnelly and Sheppard [57] proposed the use of a genetic algorithm to evolve four-part melodic harmonies, and frameworks such as DarwinTunes [122] were introduced using genetic programming tree-like representation to represent full songs. In addition to this publication, it is important to mention MetaCompose [180], a framework still used nowadays, consisting of a graph-based chord sequence generator, as well as a melody and accompaniment generator. More recently, a system that has been gaining interest is Evocomposer [166]. The novelty of this framework is the hybrid evaluation function capable of multi-objective optimisation to solve a four-voice harmonisation problem, i.e. given a music line as input, the algorithm must generate the remaining three accompanying voices.

Concerning the domain of text generation, the body of existing research seems to be more sparse. At the beginning of the millennium, Manurung et al. [140] presented the first efforts in applying evolutionary techniques to the development of an English poetry generation system using genetic programming. Manurung [139] would later extend upon this research in poetry generation, this time employing grammatical evolution. Although poetry seems to be the preferred text type for generative systems, it is worth highlighting pioneers in other genres, such as the work by Montero and Araki [150] in 2006, who has worked on the generation of dialogue in natural language. Also in the mid-2000s, Hervás et al. [82] proposed an evolutionary approach to assist in the generation of alliterative text that preserves the meaning of a given input sentence. Gervás [70] used an evolutionary algorithm coupled with his “wishful automatic Spanish poet” (WASP) system for poetry generation.

Lastly, beyond the image and music domains, we should also note the integration of evolutionary techniques with other types of domains of applications throughout the timeline. It is the case of the work by Frade et al. [66], which uses genetic programming for the evolution of realistic video game terrain. Additionally, the work of Shaker et al. [183] used grammatical evolution for the generation of *Super Mario Bros.* levels. Aside from the efforts of creating gaming assets, Browne and Maire [26] presented an approach to evolve interesting game designs from a human perspective with the Ludi framework. Still, in the domain of game design, the work by Hoover et al. [87] represents one of the first systems using evolutionary techniques

to develop multiple aspects of a level, including visuals, audio and gameplay, along with the work by Cook et al. [40] with the ANGELINA framework.

10.4.2 Evolutionary Computation Aided by Machine Learning

After summarising the timeline of evolutionary generative models that use solely evolutionary computation, we now move on to evolutionary generative models that, in addition to evolutionary computation, also use machine learning on any part of the generative process.

The machine learning approaches have aided evolutionary generative models mostly in two aspects: fitness assignment or auxiliary representations for selecting individuals. Starting with the latter, Saunders and Gero [176], in the early 2000s, proposed a system that used a self-organising map to categorise each generated image and acted as a short-term memory of their agent-based and evolutionary generative model. This is one of the first examples of using an auxiliary representation to help the evolutionary process. Nevertheless, based on our collection of works, most of the work done on evolutionary generative models aided by machine learning mostly uses machine learning to evaluate or assist in evaluating generated individuals. Until the early 1990s, the evolution of images had been done by resorting to user interaction to assign fitness to individuals, i.e. an interactive evolutionary computation approach. Albeit effective, interactive fitness assignment is tied to problems such as user fatigue or inconsistent evaluation, and it is hard to scale. The work by Baluja et al. [12] aimed to change this paradigm by proposing an automatic fitness assignment scheme based on the evaluation of images using machine learning, more precisely, an artificial neural network, to automate the evolutionary process. However, this approach only had partial success, the main problem being the employed artificial neural network was not able to generalise and distinguish between different visually striking image features and properly guide the evolutionary process. Most works at that time tended to use interactive fitness assignment schemes or opted for a semi-automatic form with the help of machine learning approaches, as is the case of the incorporation of an artificial neural network to aid fitness assignment in the enhanced GenJam framework proposed by Biles et al. [17].

In 2008, Machado et al. [133] experimented with automatic fitness assignment using artificial neural networks, similar to the work by Baluja et al. but promoting an adversarial scenario between an artificial neural network and the evolutionary computation approach. The artificial neural network was trained to discriminate among images previously generated by the evolutionary approach and famous paintings. Then the evolutionary approach was used to generate images that the artificial neural network classifies as paintings. The images created throughout the evolutionary run were added to the training set, and the process was repeated. The iterative process led to the refinement of the classifier forcing the evolutionary approach to explore different types of images after each refinement. Based on this work, Correia et al. [41] used

a machine learning approach trained with multiple object detectors to evolve figurative images consistently, i.e. images that contain a certain object according to the machine learning approach. Although different types of images generated resemble the object, some of the images evaluated by the machine learning approaches yielded high confidence but did not resemble it or were exploits of the machine learning approaches that did not correspond to the intended object. The work was later extended by Machado et al. [134], generating images that were ambiguous from a human and machine learning approach perspective. Following up on the idea of having fitness assignment schemes with machine learning approaches and their exploits, the work by Nguyen et al. [156] demonstrated how easily state-of-the-art machine learning approaches based on deep artificial neural networks can be fooled, i.e. classify images with high confidence of belonging to a certain class, when in reality the images do not resemble that class from a human perspective. The authors have shown that it is possible while using a simple genetic algorithm with a direct encoding or using compositional pattern-producing networks (CPPNs) to generate images that were unrecognisable by humans but recognised by deep neural networks with high confidence. Despite its exploits, machine learning intervention in fitness assignment schemes proliferated and is still one of the most used approaches for automating the fitness assignment process.

The idea of using machine learning to assist with fitness assignments was also used in other types of applications. For sound generation, the work by Johanson and Poli [96] presented the generation of music with automatic fitness using an artificial neural network to rate the generated music. Manaris et al. [137] proposed a hybrid approach that saw the combination of evolutionary techniques along with machine learning to compose and analyse music. In terms of text generation, akin to Manurung et al. [140], Levy [103] explored evolutionary techniques coupled with machine learning for poetry generation, generation of stories with plot induction by McIntyre and Lapata [148] and Al-Najjar and Hämäläinen [2] worked with the generation of culturally satirical movie titles. As an example of an evolutionary system employing machine learning for other applications, we include the work by Morris et al. [152] concerning evolutionary food recipes using a genetic algorithm and an artificial neural network trained on a starting set of recipes.

10.4.3 Machine Learning Aided by Evolutionary Computation

In this subsection, we shift our focus to evolutionary generative models that are machine learning-based and where evolutionary computation is employed to improve the capability of the machine learning part.

As can be seen in Fig. 10.2, this is the section with the least entries among all the categories. This can be mostly related to the fact that the generative models based on machine learning gained traction around 2014 with the advent of GANs, and most of the researchers working on evolutionary approaches started to explore models that

combined the machine learning models with the evolutionary ones. Before 2014, the work by Manaris et al. [136] introduced in 2011 the Monterey Mirror, a system that uses a genetic algorithm to evolve musical phrases provided by a Markov model.

After 2014, in this category of evolutionary generative models, most publications pertain to latent variable evolution, a term initially coined by Bontrager et al. [22] to refer to the evolutionary exploration of a given latent space. Latent spaces in generative machine learning are intermediate compressed versions of the original inputs, e.g. bottleneck layer of auto-encoders. Thus, these compressed versions, which are typically composed of numerical vectors, encode original samples and by sampling the latent space and decoding via the generative model, we can explore samples that were learnt by the generative model. This exploration by sampling and decoding vectors of the latent space holds potential that has been researched until today. For instance, in the work of Bontrager et al. [22], the authors trained a GAN to generate fingerprints capable of matching a high percentage of users and afterwards explored the generated latent space of the model using covariance matrix adaptation evolution strategy (CMA-ES). In a different setting, Fernandes et al. [64] explored the latent space to evolve groups of diverse images using genetic algorithms and multi-dimensional archive of phenotypic elites (MAP-elites). In the context of a procedural content generation scenario and following similar procedures, Volz et al. [210] generated game levels using latent variable evolution with CMA-ES and a GAN. Moreover, in the work by Schrum et al. [178], instead of directly evolving latent vectors, it evolves parameters for CPPNs that will, in turn, generate latent vectors for GANs.

10.4.4 Machine Learning Evolved by Evolutionary Computation

Lastly, we overview the timeline of the last category of evolutionary generative models, in which evolutionary computation is used to evolve entire machine learning models or entire machine learning components.

A significant instance in the literature of machine learning models evolved by evolutionary computation for generative purposes would be the NEAT framework in 2002 by Stanley and Miikkulainen [191]. NEAT represented one of the first neuroevolution approaches capable of automatically evolving and adjusting the topology and weights of an artificial neural network. Stanley [190] would later extend upon his work by proposing CPPNs and evolving these networks with the NEAT method. Thus, in 2008, the CPPN-NEAT approach was made publicly available in the form of an online evolutionary tool named Picbreeder [181], capable of guiding evolution using an interactive fitness assignment scheme. Furthermore, Dubbin and Stanley [59] generated dance moves using the CPPN-NEAT model.

Around the same time, Togelius et al. [200] explored a competitive coevolutionary generative model for racing car controllers. These early studies regarding controllers led the way to the evolutionary exploration of game design features, which was

mostly set in motion by Togelius et al. [201] with the evolution of controllers to play *Super Mario Bros*. Other works followed, such as the generation of shapes in video games by Liapis et al. [111] and the evolution of gameplay in the work of Cardona et al. [30], using competitive coevolution to evolve agents for both the player and the non-player characters in the classic game *Pac-Man*.

In the realm of music, we mention the work by Bell [13] in the generation of musical compositions by evolving populations of Markov models with a genetic algorithm. Despite this, and perhaps motivated by a saturation of research concerning traditional machine learning methods to improve the performance of GANs, the end of the last decade saw an increase in evolutionary machine learning research mostly based on coevolution. Aside from neuroevolution being central in the same timeline, evolutionary approaches that operate to evolve entire generative models, such as GANs, have a high computational cost. Here the advantages of the optimisation aspect of evolutionary computation were to be explored while minimising the computational cost of an already demanding training process of the state-of-the-art generative machine learning models.

Extending upon the GAN scheme, in 2018, Garciarena et al. [68] presented a method for generating Pareto set approximations of generators and discriminators. In this method, the authors resort to the coevolution of generators and discriminators where a single individual represents both components. In the same year, the Lippizanner framework was proposed by Al-Dujaili et al. [1] as the first system encompassing a spatial approach to coevolution where a grid of cells was used for evolving groups of GANs. Moreover, in 2019, Wang et al. [211] proposed the evolutionary GAN (E-GAN) model, which consisted of the evolution of a population of generators within a GAN model. This approach was later enhanced by Costa et al. [49] with the proposal of coevolutionary GAN, a model based on coevolution where an additional population of discriminators is evolved alongside the population of generators. Additionally, Toutouh et al. [203] extended the Lippizanner framework by incorporating the generator training method from the E-GAN model, thus effectively combining variation operators and population-level approaches to improve the diversity of GANs.

10.5 Evolutionary Computation Without Machine Learning

In the last section, we provided a historical overview and timeline of the works in each category. In this section, as well as in the three following ones, we will cover the collected works on evolutionary generative models. We start by analysing the works that only use evolutionary algorithms. Table 10.1 enumerates a series of evolutionary generative models which we consider to belong to the category *evolutionary computation without machine learning*. The listed publications are grouped by application domain (first column) and then sorted by year (second column) and by authors (third column).

Our analysis begins with applications in the image domain, including shape generation, 3D modelling and video generation, among others. Afterwards, we delve into sound applications, covering everything from sound effects to full-fledged music composition. Next, the most relevant works in text generation are analysed, and we finish the section with other applications which do not fit into the mentioned application domains. Each of the works analysed in each domain is further subdivided by the type of evolutionary algorithm used.

When it comes to using evolutionary computation as an image generative model, the two most common approaches are by far genetic programming and genetic algorithm, with genetic programming taking a slight preference within the literature analysed (see Table 10.1). Most of the approaches that employ genetic programming do so by evolving symbolic expressions that generate images through the process of feeding coordinates as input. This way, to evaluate an individual, each pixel coordinate is fed into the expression that will generate a colour (typically a tuple of values representing RGB channels) of which the output image, i.e. the individual, will be composed of. These approaches have been applied to the generation of a diverse range of image types, including aesthetic imagery [76–78, 80, 126, 172, 185, 186, 207, 214], fractals as iterated function systems [58] or the generation of specific image classes [9, 16, 55, 56, 124, 125, 169]. Other works define the features of an output image iteratively: vector graphics [54], stochastic sequencing of points to generate iterated function systems [31], evolving image filters [154] and the evolution of 2D renders of 3D compositions [129]. Regarding the type of fitness function, for the purposes of image generation with genetic programming, we observe that there are more static than dynamic (see Table 10.1), mostly due to automation of the generative model via hardwired fitness functions.

Akin to genetic programming, genetic algorithms were also used to generate abstract imagery [37, 52, 187]. Aside from image generation, genetic algorithm-based generative systems were used to generate drawings of a specific category by evolving properties of primitives such as coordinates, rotation and size. Line drawings are perhaps the simpler examples of these approaches. Within fractals, Ventrella [208] used a genetic algorithm to explore different configurations of the Mandelbrot set. Baker and Seltzer [11] evolved lines either using line drawings coordinates or by evolving agents that live on a canvas and spread their raster until they die. This aspect of agents having an energy limit to operate was a typical aspect of agent-based approaches [79, 145, 175].

The evolution of vector graphics is also explored in genetic algorithms and evolutionary strategies for image generation [15]. Aside from vectors, similar works made use of other primitives. Examples include examples of evolving images directly bitmaps via transformations and distortions [74], the generation of cartoon faces [104, 157], emojis [50], letterings [161], posters [167], the evolution of renderings based on existing images (non-photorealistic rendering) [35, 132] and the evolution of context-free grammars [131]. Animation is another field where genetic algorithms stand out, with some works evolving fractals [5, 6, 24] and screensavers [58].

Other works demonstrated the interplay between the image and music domains by generating images based on musical input directly [217] or based on systems evolved

using music [170]. Other authors developed meta-models by evolving the parameters of these algorithms [7, 121] or resorted to an indirect way of generating the outputs, e.g. Greenfield [75], although not concerning image generation directly, produced controllers for drawing robots that, in turn, created visually appealing imagery.

Readers interested in a thorough analysis of evolutionary computation for the purposes of image generation are encouraged to read the works by Galanter [67], Romero and Machado [171] as well as Lewis [105].

Akin to image generation, genetic algorithms and genetic programming are again the most prolific techniques in the generation of sound, with genetic algorithms taking a slight preference within the literature reviewed. Concerning genetic programming, many works deal with the generation of musical melodies [97, 164], some of them of a particular style such as Jazz [188]. However, the largest body of research includes both the generation of rhythm along with accompanying melodies [51, 85, 101, 102, 122, 138, 147, 193]. Some of these are music guided towards specific properties of sound and musical score [51], while others focus on the methods behind the compositional process [193]. Other works resort to grammatical evolution for the generation of music that has been applied mainly to musical composition [3, 184], including melody formation [53, 115, 116, 119, 168] as well as live coding [117].

Regarding genetic algorithms, a number of works dealt with the generation of specific parts of music such as the melody [19, 71, 151, 196] and the rhythm [92]. Some studies delved into the problem of harmony production, i.e. given a musical line/voice, the system is tasked with the creation of accompanying lines/voices to have a complete musical piece with chords for each input note [57, 165, 166], with the harmonisation of bass lines being a recurring theme, in particular [165].

However, most works in this section tackle the broader problem of automated musical composition [61, 69, 93, 98, 159, 180], some of these deal with the generation of responses to given musical phrases instead of generating sound from the ground up [18]. It is worth noting that in musical composition, many approaches tackle different problems, such as generating variations from existing pieces, thematic bridging and live coding. The generation of variations deals with the composition of pieces given an existing source [4, 95, 197]. On the other hand, thematic bridging concerns the generation of lines capable of seamlessly transitioning between two given musical phrases [91]. More recently, thematic bridging was also explored using evolutionary strategies in the work by Martínez-Rodríguez [142], who proposed fitness as a weighted neighbourhood average distance metric between a solution and the target. More applications include live coding, which tackles the use of domain-specific languages for generating new musical patterns in a live concert setting [84].

Generally, we can observe that most systems for sound generation based on genetic algorithms and genetic programming implement static fitness assignments, with user-defined fitness being a minority. Still, in music composition, more recent works try to gather inspiration from different domains. An example of this is the work by Santos et al. [174], which bridges the fields of image and music generation by using images as a source of inspiration for the compositional process.

The above-mentioned works illustrate examples from the literature for evolutionary computation in music. Readers who seek a complete analysis of the appli-

cation of evolutionary computation to music should refer to the following surveys by Todd and Werner [198], McCormack [146], Loughran and O’Neill [118] and Eigenfeldt et al. [60].

The body of work analysed regarding the application of evolutionary computation for text generation is rather lacking compared to the domains of image and sound. Nevertheless, the genetic algorithm is by far the most used approach, with applications to the generation of poetry [70, 139–141], dialogue (i.e. natural language generation) [150], as well as alliterative text [82]. In particular, it is worth mentioning the work by Manurung [139], where the evolution of grammars was used to generate poetry. For the more curious reader, a survey by Oliveira [158] analyses automated poetry, which goes beyond the scope of evolutionary computation.

Finally, we list works that one way or another concern the generation of artefacts that do not fit the categories of either images, sound or text. Here it is worth mentioning the automatic evolution of designs for Lace Knitting Stitch Patterns by Ekárt [62], and the modelling of terrain maps by Frade et al. [66], both resorting to genetic programming techniques. Moreover, the works dealing with 3D [28, 177] and shape design [153] are worth noting. Similar to what was verified for sound, text and other applications, most of these applications implement fixed fitness assignment as well. Lastly, another avenue where evolutionary computation has been applied is to aid game design. In this context, we include Ludi, a general game system introduced by Browne and Maire [26] to synthesise and evaluate new games capable of captivating the interest of human players, as well as the use of a (1 + 1) evolutionary strategy for the generation of decks of cards by Cardona et al. [29]. Regarding genetic algorithms, we can mention the works by Cook et al. [40] and Cook and Colton [39] in the automated and cooperative coevolution of game design. Still pertaining to game design, some other examples consist in the co-generation of game levels alongside a human designer [120] as well as the evolution of level design to increase the feeling of in-game tension [114]. Refer to the survey of Liapis [109] for a thorough analysis of the applications of evolutionary computation and procedural content generation, among other techniques to the generation of game design.

In terms of representation used in the analysed evolutionary generative models, we found instances of the different types of representation, with a clear predominance of procedural representations (70 procedural, 23 descriptive and 19 parametric representations).

10.6 Evolutionary Computation Aided by Machine Learning

This section includes approaches where models are inherently evolutionary, but their functionality or performance is improved with machine learning approaches. A common example is an approach where the evolutionary model uses machine learning in

their fitness assignment or one of the components of the evolutionary computation model is replaced by machine learning. Table 10.2 presents examples of evolutionary generative models, which we consider to belong to the category *evolutionary computation aided by machine learning*. The listed publications are grouped by application

Table 10.2 Examples of evolutionary generative models categorised as *evolutionary computation aided by machine learning*

Domain	Year	Authors	EA	Population	Represent.	Fitness
Image	1994	[12] Baluja et al.	GP	Single	Procedural	Static
Image	1994	[176] Saunders and Gero	GP	Multi. (colab.)	Procedural	Dynamic/Static
Image	2008	[36] Colton	GA	Single	Parametric	Dynamic
Image	2008	[133] Machado et al.	GP	Single	Procedural	Dynamic
Image	2009	[32] Chen et al.	GA	Single	Parametric	Static
Image	2012	[127] Machado et al.	GP	Single	Procedural	Static
Image	2012	[128] Machado et al.	GP	Single	Procedural	Dynamic
Image	2013	[41] Correia et al.	GP	Single	Procedural	Static
Image	2015	[134] Machado et al.	GP	Single	Procedural	Static
Image	2015	[143] Martins et al.	GA	Multi. (colab.)	Descriptive	Static
Image	2015	[156] Nguyen et al.	GA	Single	Procedural	Static
Image	2016	[44] Correia et al.	GA	Single	Parametric	Static
Image	2016	[209] Vinhas et al.	GP	Single	Procedural	Static
Image	2019	[42] Correia et al.	GP	Single	Procedural	Dynamic
Image	2019	[43] Correia et al.	GA	Single	Parametric	Static
Image	2019	[144] Martins et al.	GA	Single	Procedural	Static
Image	2019	[195] Tanjil and Ross	GP	Single	Procedural	Static
Image	2020	[113] Lopes et al.	GA	Single	Procedural	Dynamic/Static
Image	2021	[45] Correia et al.	GP	Single	Procedural	Static
Image	2022	[10] Baeta et al.	GP	Single	Procedural	Dynamic
Music	1991	[71] Gibson and Byrne	GA	Single	Descriptive	Static
Music	1995	[189] Spector and Alpern	GP	Single	Procedural	Static
Music	1996	[17] Biles et al.	GA	Single	Descriptive	Static
Music	1998	[27] Burton and Vladimirova	GA	Single	Descriptive	Dynamic
Music	1998	[96] Johanson and Poli	GP	Single	Procedural	Dynamic/Static
Music	2001	[162] Pearce and Wiggins	GA	Single	Descriptive	Static
Music	2007	[137] Manaris et al.	GP	Single	Procedural	Dynamic
Music	2007	[163] Phon-Ammuisuk et al.	GP	Single	Procedural	Static
Music	2016	[94] Ianigro and Bown	GA	Single	Procedural	Dynamic
Text	2001	[103] Levy	GA	Single	Descriptive	Static
Text	2010	[148] McIntyre and Lapata	GA	Single	Procedural	Static
Text	2018	[2] Al-Najjar and Hämäläinen	ES	Single	Descriptive	Static
Text	2021	[213] Winters and Delobelle	GA	Single	Descriptive	Static
Other	2012	[152] Morris et al.	GA	Single	Procedural	Static
Other	2018	[108] Liapis	GA	Single	Parametric	Static

domain (first column) and then sorted by year (second column) and by authors (third column).

Similar to Sect. 10.5, the use of genetic programming in this section mainly pertains to the generation of specific image classes/types [41, 127, 128, 134, 143, 195, 209] or to aesthetic imagery [12, 42]. In addition to this, genetic programming methods for the transformation of existing images have also been presented, such as the evolution of image pipelines by Correia et al. [45]. Machado et al. [133] and Correia et al. [42] explored the stylistic change in evolutionary art in a framework similar to GANs using an adaptive classifier along with a genetic programming engine. Similarly, Baeta et al. [10] proposed another example of a system having a competition between isolated components, where the generator of a standard deep convolutional GAN is replaced by a genetic programming approach.

Regarding genetic algorithms for the image domain, examples of machine learning for the fitness assignment phase include the use of artificial neural networks either by evolving false positive images [156], false negatives [44] or even to perform data augmentation [32, 43]. Lopes et al. [113] and Martins et al. [144] also employed a genetic algorithm guided by a machine learning model to evolve vector graphics, and stencils for typefaces, respectively. Similar to the work by Bergen and Ross [15] (see previous section), both modalities of fitness function (static and dynamic) were explored, the latter as a separate experiment and the former by mixing a static target function with user-defined options. Graph evolution is another example of the application of genetic programming to image generation, as exemplified by Vinhas et al. [209].

Concerning genetic programming, there are examples for the generation of melodies [96, 163], including Jazz [188], and the generation of rhythm with accompanying melodic tracks, explored by Manaris et al. [137] as well as Spector and Alpern [189]. Using genetic algorithms, many works dealt with the generation of specific parts of music such as melody [71] and rhythm [27]. Other works focussed solely on harmony generation [162], i.e. musical responses, instead of generating sound from scratch. Like in the category *evolutionary computation without machine learning*, automated music composition is also addressed in this section with the work of Biles et al. [17], who used artificial neural networks for fitness assignment. Although almost all examples of the application of genetic algorithms to music generation dealt with music, the broader exploration of audio, in general, was addressed by Ianigo and Bown [94].

Relating to text generation, a genetic algorithm was used by Levy [103] for the purposes of poetry generation with an artificial neural network as a measure of adaptive fitness. Other examples include the study of satire generation by Winters and Delobelle [213] and by Al-Najjar and Hämäläinen [2]. Finally, examples of other applications include the works by Liapis [108] for the evolution of colour palettes, the evolution of scene elements by Colton [36] and the introduction of a system for generating novel culinary recipes by Morris et al. [152].

In this section, most of the analysed works in terms of representation have representativity of the three types of representation but with a prevalence of the procedural ones (20 procedural, 9 descriptive and 6 parametric representations).

10.7 Machine Learning Aided by Evolutionary Computation

In this section, we include the approaches where evolutionary computation is used to enhance the functionality or performance of the machine learning model. As an example, latent variable evolution belongs in this section because these approaches use evolutionary computation to explore a latent space previously built using a machine learning model. Table 10.3 contains examples of evolutionary generative models which we consider to belong to the category *machine learning aided by evolutionary computation*. The listed publications are grouped by application domain (first column) and then sorted by year (second column) and by authors (third column).

The idea of latent variable evolution was introduced by Bontrager et al. [22, 23] to explore the latent space of inputs to the generator network of a GAN using a CMA-ES approach to generate fingerprints that maximise the number of imposter matches. Concerning genetic algorithms, Schrum et al. [178] evolved parameters for CPPNs, which in turn generated a variety of different latent vectors for GANs.

Other examples concerning the application of genetic algorithms to latent variable evolution include the work by Grabe et al. [73], who used Gaussian mixture models to explore possible latent vectors to evolve distinct images, along with interactive methods to perform the evolutionary search by Zaltron et al. [215], among others [21, 64, 135]. Outside of latent variable evolution but still pertaining to image generation, Colton [38] also proposed a system to generate image filters using a genetic algorithm with MAP-elites or the work by Korde et al. [99], which uses an evolutionary algorithm to train in the initial iterations to stabilise the weights before using normal optimisation techniques to train GANs.

Table 10.3 Examples of evolutionary generative models categorised as *machine learning aided by evolutionary computation*

Domain	Year	Authors	EA	Population	Represent.	Fitness
Image	2018	[22] Bontrager et al.	ES	Single	Parametric	Static
Image	2018	[21] Bontrager et al.	GA	Single	Parametric	Dynamic
Image	2018	[23] Bontrager et al.	ES	Single	Parametric	Static
Image	2019	[99] Korde et al.	GA	Single	Parametric	Dynamic
Image	2020	[64] Fernandes et al.	GA	Single	Parametric	Static
Image	2020	[178] Schrum et al.	GA	Single	Parametric	Static
Image	2020	[215] Zaltron et al.	GA	Single	Parametric	Dynamic
Image	2021	[38] Colton	GA	Single	Parametric	Static
Image	2021	[110] Liapis et al.	GA	Multi. (colab.)	Procedural	Static
Image	2022	[73] Grabe et al.	GA	Single	Parametric	Static
Image	2022	[135] Machín et al.	GA	Single	Parametric	Static
Music	2011	[136] Manaris et al.	GA	Single	Procedural	Static
Other	2018	[210] Volz et al.	ES	Single	Parametric	Dynamic/Static
Other	2020	[160] O'Reilly et al.	GP	Single	Procedural	Dynamic

Concerning other applications, an approach combining genetic algorithms and Markov chains was proposed by Manaris et al. [136] to evolve musical phrases. Volz et al. [210] worked on the generation of game levels using latent variable evolution with adversarial training. Moreover, the work by O'Reilly et al. [160] in cyber-attacks and defence mechanisms using genetic programming in an adversarial setting, among other techniques, explored generating and evolving executable adversarial programs. Lastly, we mention the work of Liapis et al. [110] for the creation of suitable spaceships for arcade-style games using CPPN-NEAT and novelty search.

One of the most noticeable aspects of this section lies in the fact that, when compared to other ones, the body of work concerned with aiding machine learning with evolutionary computation is significantly less, especially when compared to Sect. 10.5. Moreover, in terms of representation, we can observe that most approaches are parametric, with just a few being procedural (in total, there are 3 procedural, 11 parametric and no descriptive representations). This is because the analysed works mostly use latent variable evolution, which evolves parameters to generative machine learning models.

10.8 Machine Learning Evolved by Evolutionary Computation

This section addresses approaches where evolutionary computation is used to evolve one or more populations of generative machine learning models. We present direct applications of evolutionary computation to machine learning or parts thereof. These applications are direct in the sense that the evolutionary computation techniques are applied as is, i.e. without modifications, over the set of machine learning individuals. As an example, here we include the NEAT-like approaches because the NEAT algorithm evolves a population of artificial neural networks. Table 10.4 presents examples of evolutionary generative models which we consider to belong to the category *machine learning evolved by evolutionary computation*. The listed publications are grouped by application domain (first column) and then sorted by year (second column) and by authors (third column).

Most of the research within the literature concerning machine learning models evolved by evolutionary computation has to do with the evolution of artificial neural networks and CPPNs [190], and with the use of neuroevolution techniques such as NEAT [191]. Because most fixed-topology neuroevolution models typically generate artificial neural networks to be used for classification tasks, such approaches are not within the scope of this section. For literature regarding the use of evolutionary approaches in supervised and unsupervised learning, see Sects. 1.6 and 2.5, respectively.

There have been a number of works inspired by the CPPN-NEAT framework proposed by Stanley [190] to generate images by augmenting and evolving a CPPN [63, 181, 182, 206, 216]. It is worth noting that some of the mentioned works have been applied to the GAN model, as is the case with the replacement of a standard

Table 10.4 Examples of evolutionary generative models categorised as *machine learning evolved by evolutionary computation*

Domain	Year	Authors	EA	Population	Represent.	Fitness
Image	2007	[190] Stanley	GA	Single	Procedural	Dynamic
Image	2008	[181] Secretan et al.	GA	Single	Procedural	Dynamic
Image	2011	[182] Secretan et al.	GA	Single	Procedural	Dynamic
Image	2015	[216] Zhang et al.	GA	Single	Procedural	Dynamic
Image	2017	[192] Suganuma et al.	GP	Single	Procedural	Dynamic
Image	2018	[1] Al-Dujaili et al.	GA	Multi. (advers.)	Procedural	Dynamic
Image	2018	[68] Garcíarena et al.	GA	Single	Procedural	Dynamic
Image	2018	[206] Turhan and Bilge	GA	Single	Procedural	Dynamic
Image	2019	[34] Cho and Kim	GA	Single	Parametric	Dynamic
Image	2019	[46] Costa et al.	GA	Multi. (colab.)	Procedural	Dynamic
Image	2019	[49] Costa et al.	GA	Multi. (colab.)	Procedural	Dynamic
Image	2019	[203] Toutouh et al.	GA	Multi. (advers.)	Procedural	Dynamic
Image	2019	[211] Wang et al.	GA	Single	Procedural	Dynamic
Image	2020	[47] Costa et al.	GA	Multi. (colab.)	Procedural	Dynamic
Image	2020	[204] Toutouh et al.	GA	Multi. (advers.)	Procedural	Dynamic
Image	2021	[33] Chen et al.	GA	Multi. (colab.)	Parametric	Dynamic
Image	2021	[63] Ekern and Gambäck	GA	Single	Procedural	Dynamic
Image	2021	[81] Hemberg et al.	GA	Multi. (advers.)	Procedural	Dynamic
Image	2021	[107] Li et al.	GA	Single	Procedural	Dynamic
Image	2021	[205] Toutouh and O'Reilly	GA	Multi. (advers.)	Procedural	Dynamic
Image	2022	[65] Flores et al.	GA	Multi. (advers.)	Procedural	Dynamic
Music	2008	[88] Hoover et al.	GA	Single	Procedural	Dynamic
Music	2009	[89] Hoover and Stanley	GA	Single	Procedural	Dynamic
Music	2011	[13] Bell	GA	Single	Procedural	Dynamic
Music	2011	[86] Hoover et al.	GA	Single	Procedural	Dynamic
Music	2014	[90] Hoover et al.	GA	Single	Procedural	Dynamic
Music	2015	[179] Scirea et al.	GA	Single	Procedural	Dynamic
Text	2020	[112] Liu et al.	GA	Single	Procedural	Dynamic
Other	2007	[200] Togelius et al.	ES	Multi. (advers.)	Procedural	Dynamic
Other	2009	[201] Togelius et al.	GP	Single	Procedural	Static
Other	2010	[59] Dubbin and Stanley	GA	Single	Procedural	Dynamic
Other	2012	[111] Liapis et al.	GA	Single	Procedural	Dynamic
Other	2013	[30] Cardona et al.	ES	Multi. (advers.)	Procedural	Dynamic

deep convolutional GAN generator with a CPPN evolved with NEAT, proposed by Ekern and Gambäck [63]. Another example related to GANs is the work by Turhan and Bilge [206], which combined a variational auto-encoder with the pixel-wise generative capability of CPPNs to create a GAN-variational auto-encoder model, i.e. a GAN with an inference mechanism in the form of an extra encoder network. Similar to Sect. 10.6, despite most models in this section not exhibiting competition,

these last two studies can be identified as having a competition between isolated components [63, 206].

Apart from NEAT approaches, but still within the context of genetic algorithms, Wang et al. [211] proposed E-GANs, which consisted in the evolution of populations of generators. Similar works also used GANs to evolve samples from the generator [34]. Costa et al. [49] worked on the coevolution of sub-populations of generators and discriminators based on CoDeepNEAT [149] with multiple variants, e.g. one based on novelty search with local competition [47]. The technique of evolving both the generator and the discriminator was applied in other works, both with different sub-populations for each component [33] and in a single population where each individual represents a tuple consisting of both components [68].

Regarding coevolutionary approaches, we can identify a sub-category of models dealing with spatial coevolution. This idea was first proposed by Al-Dujaili et al. [1] with the introduction of the Lipizzaner framework using a grid of cells where GANs are allowed to evolve and communicate. Other works expanded on Lipizzaner by hybridising it with E-GAN and Lipizzaner to increase solution diversity [203] and proposing a ring topology for the spatial cell grid instead of a toroidal one [205]. Other works apply the same idea by scaling the framework in the context of a high-performance computing setting for the medical domain [65] or by further exploring the features of Lipizzaner [81, 204]. Lastly, although during our literature review, we did not find a substantial body of work in this section outside of the image domain, it is interesting to note the evolution of Markov chains in the realm of music using genetic algorithms [13], an approach later explored further by Scirea [179] for the generation of music from scientific papers.

In the realm of music generation, many of the studies analysed here also employ NEAT to evolve CPPNs. These approaches typically use functional scaffolding, a technique that exploits the fact that a pattern of notes and rhythms in different instrumental parts of the same song is functionally related. These studies were initially performed by Hoover et al. [88] and Hoover and Stanley [89], who started by presenting a framework called NEAT Drummer for the generation of drum patterns, which was later extended to the application of full-fledged music composition [86, 90]. Concerning text generation, we point out the work of Liu et al. [112] pertaining to the creation of a category-aware model employing a genetic algorithm to evolve a population of GANs.

Albeit not many, there are a few noteworthy works in the scope of this section that are concerned with the generation of artefacts not pertaining to imagery or music. It is the case with the generation of shapes in video games by Liapis et al. [111], as well as other work exploring the coevolution of controllers [30, 200]. Furthermore, Dubbin and Stanley [59] generated dance moves using the CPPN-NEAT model.

As shown in Table 10.4, genetic algorithms account for the vast majority of approaches in this section (31 out of 32 papers), with only one using genetic programming to design both the structure and connectivity of the convolutional neural networks [192]. As a matter of fact, the standard NEAT framework representation is used in almost half of the models analysed herein and it is procedural by definition (in total there are 33 procedural, 2 parametric and no descriptive

representations). Lastly, regarding the fitness assignment, we conclude that most approaches are dynamic, especially the ones in an adversarial setting.

10.9 Open Problems and Challenges

Throughout the history of evolutionary generative models, several problems have been tackled progressively. It is the case with the fitness assignment of early evolutionary computation as discussed in Sect. 10.4. The first explorations of the generative capabilities of evolutionary computation models required the user to select the outputs for the evolutionary generative model. For many years, and especially in the fields of evolutionary art and music generation, user fitness assignment was the *de facto* technique. However, researchers noted that having the user in the loop with the system presented an enormous bottleneck that had to be addressed for the field to progress, and thus some papers started tackling this issue [12]. Nowadays, even though interactive systems are still very prolific, the most standard way of dealing with fitness assignment is to have some automatic algorithm, be it analytical, statistical (such as Markov models and classifiers) or indirectly informed in some other way. This section addresses some of these problems along with the strides that have recently been taken in addressing them.

Throughout this survey, we have identified the following challenges when dealing with evolutionary generative models.

10.9.1 *How to Represent the Space of Solutions Generated by the Model?*

GANs and other generative machine learning approaches are characterised by having a latent space, from which one selects a sample from that space, inputs it into the model and generates the output solution. Moreover, GANs allow for the conditional exploration of solutions through algebraic vector operations such as addition, subtraction and interpolation, to mention a few, which will, in turn, ease the exploration of such solutions. However, most evolutionary generative models, especially those that apply evolutionary computation directly (see Sect. 10.5), do not aggregate solutions in such an organised latent space. Therefore, a way to better represent the space of generated solutions is oftentimes needed. In this scope, several viable alternatives are considered throughout the literature. Although not latent spaces, spatial representations of the solutions have been constructed by resorting to an archive that stores solutions according to their novelty [42, 144] or by employing manifold learning algorithms to aggregate solutions on a space with lower dimensionality with techniques such as t-distributed stochastic neighbour embedding (t-SNE) [48]. Another trending solution is to build a latent space by combining a variational auto-encoder into the model by training an additional network that learns to encode

an image into a smaller latent vector. These are called the GAN-variational auto-encoder approaches [206]. At the moment, this challenge differentiates *evolutionary computation without machine learning* and *evolutionary computation aided by machine learning* approaches from *machine learning aided by evolutionary computation* and *machine learning evolved by evolutionary computation* because by using machine learning as a generative approach, typically the machine learning model will have the latent space with a solution space to be drawn. In the case of evolutionary computation-based approaches, we have solutions being evolved but they are independent solutions, unstructured, and not contained in a defined space, which can be seen as a drawback.

10.9.2 How to Navigate the Generated Space of Solutions?

This challenge concerns the exploration of all feasible solutions that can be generated by the model. Within the models that organise their solutions in a latent space, an effective way of exploring these solutions is to perform latent variable evolution directly over this space. Because a latent vector is the genotype representation of latent variable evolution, the types of algorithms used are mostly genetic algorithms [64, 73], and evolutionary strategies in some cases [22] (see Sect. 10.7 for a thorough analysis). For models where a latent space does not exist, the exploration of the solutions space cannot be done directly, and thus other methods need to be employed to aid solution exploration during evolution, e.g. exploration with variation operators or novelty search [110, 209]. Such methods typically involve hand-crafting specific variation operators to fence off the bias of the models. This can be seen, for instance, in the work by Li et al. [107] with the proposal of a new crossover operator for the E-GAN model and in the implementation of a mutation operator with features tailored to ease the process of interactive 3D design by Byrne et al. [28].

10.9.3 How to Evaluate the Generative Capabilities of a Model?

In this challenge, we are concerned with evaluating the generative capabilities of the model by itself without comparing it to similar generative models, whether evolutionary or not. This problem is often difficult to tackle as it depends a lot on the desired output of the model and the process itself. For instance, when it comes to music composition, Ng and Jordan [155] used genetic programming to examine evolving compositional processes instead of a musical piece. As further explained by Loughran and O'Neill [118], it is imperative to look beyond the generated output of a model when investigating the ability of the system to compose. This way, as analysed by Li et al. [106], there are two ways to evaluate a music composer: by evaluating the music it composes or by evaluating the composition process. This

idea can be extended outside the specific domain of music generation. Given the difficulties in evaluating isolated solutions, we might see more works where models evaluate the generative process itself and not only the output solutions.

10.9.4 How to Compare Evolutionary and Non-evolutionary Models?

Following the challenge of isolated evaluation, we turn our attention to the comparison of performance between different generative models, namely, between generative models with and without evolutionary computation. It is demonstrated that in adversarial generative models, the loss function is not stable enough to be taken as a comparative measure of generative performance [46]. In fact, it is shown that even simple metrics such as the inception score, which applies the Inception Model [194] to every generated image in order to get the conditional probability distribution of classes in the original data [173], fails to capture similarities between generated images adequately. For this reason, the Frechét inception distance metric [83] has been proposed. Although the Frechét inception distance was specifically developed to evaluate the performance of GANs, it can be applied to any image generation model. The Frechét inception distance directly compares the statistics of real samples with the statistics of generated ones, allowing for the capture of features used for a collection of both real and generated images gathered from training the network on image classification tasks. One major shortcoming of the inception score and Frechét inception distance scores is that they fail to determine whether the proposed model is overfitting or underfitting [20]. This problem pertains not only to evolutionary generative models but also to standard generative models. To overcome this hurdle, metrics such as the Reconstruction Error [212] and the Kernel Inception Distance were introduced [20]. Even though applying these metrics in evolutionary generative models still seems very rare, we posit that using these more advanced metrics will become more widespread in the study of evolutionary generative models as research in the field keeps taking off.

10.9.5 How to Improve the Computational Efficiency and Scalability of Evolutionary Generative Models?

Generally speaking, when it comes to improving computational efficiency, there are two ways a model can be optimised: through macro-optimisations, i.e. algorithmic improvements in our case, or through micro-optimisations, which in computer science typically means small statement-level optimisations to the source code. However, by micro-optimisations, we refer to the adaption of the model to run in different hardware according to the demands of operations in its algorithm. As an example, the work by Turhan and Bilge [206] tackles the problem of efficiently

generating high-resolution versions of handwritten digits, which is a slow task in conventional GAN-variational auto-encoder models. The proposed model converges more efficiently and thus faster than the baseline models by allowing the feature-wise reconstruction of a GAN-variational auto-encoder hybrid model instead of the standard pixel-wise reconstruction approach. This is a typical example of an algorithmic improvement. Another noteworthy example in the category of computational performance is TensorGP [9]. TensorGP is a genetic programming engine that not only performs algorithmic optimisations by changing the internal representation of symbolic expressions from the traditional tree graph to a directed acyclic graph, thus avoiding code re-execution but also has the option to evaluate expressions in dedicated hardware if available. This way, while most of the evolutionary stages such as variation operators, selection and other control mechanisms run on the central processing unit (CPU), the evaluation can take place in graphics processing units (GPUs) or tensor processing units (TPUs), which are capable of vectorising the pipeline of genetic programming operations, something that is still not implemented in many off-the-shelf genetic programming engines [8]. With the continued slowing down of Moore's law and GPU computing still on the rise, it makes sense to expect an increase in the number of frameworks using parallel capable hardware not only in evolutionary generative models but in generative models.

Coupled with the problem of computational efficiency, there is the problem of complexity associated with the generative process. The early works by Sims [185] in image generation using symbolic expressions are straightforward examples of low computational cost in terms of the generative process. As a counter-example, the work by Correia et al. [45] in enhancing image pipelines is fairly heavier than most generative processes because the evolutionary generative model encompasses more transformation steps. In this work, genetic programming was used to evolve image filters that, aside from requiring the traditional evaluation of the individual tree graphs, are then applied to a predefined image, which is, in turn, evaluated using an external classifier for fitness assignment. Overall, it seems that as time passes by, generative models tend to complexify, demanding more computational resources and thus becoming harder to scale and deal with.

10.9.6 How to Improve the Interaction of Evolutionary Generative Models?

Another aspect that we believe to be extremely relevant is the way we interact with evolutionary generative models. Not all generative models are easy to use for the average user, offering a poor user experience which will probably limit their reach to a broad audience. Therefore, it is essential to explore effective ways to control and visualise the different elements of an evolutionary generative model. This goes beyond facilitating a given action. Interaction can help to understand a certain functionality or aspect of the models that are often very technical.

For instance, although interfaces for large language models such as ChatGPT¹ have been trending as of late, the generative pre-trained transformer (GPT) model from which ChatGPT was built is not widely known despite having been created earlier [25]. The main reason for this lies in the fact that ChatGPT has a simple, natural and inviting user interface. Likewise, Midjourney,² a system which offers a model for the generation of images from natural language descriptions in the form of text prompts, is another example of an increasingly popular artificial intelligence service. Both examples have the benefit of being easily accessible: ChatGPT through the OpenAI website and Midjourney through a server hosted on Discord, a popular chat application.

Historically, interfaces in evolutionary generative models were created to aid the user to interact with them, control the evolutionary process, preview individuals and assign fitness [124, 181]. Other systems allow the designing of fitness functions [130, 144], a process that poses challenges in terms of interpretability and explainability to its design. There are other systems that allow for inspecting and navigation of the outputs [80, 143, 182], which also poses challenges from the implementation and user interaction perspectives.

However, the majority of evolutionary generative models possess neither this level of user-friendliness nor accessibility. In fact, most evolutionary generative model frameworks use their own interfaces, with different layouts and sometimes unintuitive functionality, lacking both generalisation and scalability.

We believe that the interaction with these models can be planned not only with human users in mind but also with other computational systems with which they can be integrated, following the example of nowadays machine learning-based systems. This can be done by developing APIs that allow for the development of new systems while promoting their modularity. The solution of using APIs solves the decoupling of the generative model, however, does not solve the part of adding a standardised experience for the interaction with the generative model which can be seen as an open problem.

10.9.7 How to Increase the Availability of Evolutionary Generative Model Systems?

As a final open challenge, we address the availability of the implemented systems to the general public. Within the surveyed literature, the number of papers with publicly source-code repositories is relatively scarce. Truth be told, this is not a problem specific to the field of generative models or even to artificial intelligence but one that is common in most research fields. Albeit a big challenge, websites such as huggingface.com and paperswithcode.com are a good step forward as they help disseminate and organise machine learning and artificial intelligence papers alike,

¹ <https://chat.openai.com/>.

² <https://docs.midjourney.com/>.

along with their respective code. Moreover, we believe that having open access to new methods and their respective implementations is especially important to advance the field of evolutionary generative models.

Having identified the core challenges and future trends in evolutionary generative models, it is clear that some are inherently more difficult to tackle than others. Challenges such as the availability of implemented evolutionary generative model systems are more of an ongoing problem that, arguably, is not expected to be solved so soon as it is pervasive in many other fields. Similarly, despite recent breakthroughs, technical challenges such as the construction of organised solution spaces or the evaluation of the generative capabilities of a model are non-obvious challenges that, due to being very dependent on the type of evolutionary generative model, will likely keep posing challenges as the topic of evolutionary generative models evolves, and new approaches are proposed. In contrast, challenges such as bottleneck mitigation and overhead reduction can arguably be easier to implement in the sense that they typically consist of general techniques that can be applied to a wide range of models. For instance, with the continuous rise in parallel computing, techniques as well as research and development investments in hardware with vectorisation technologies are increasing. Nonetheless, instead of dwelling on the many hurdles surrounding this topic, it is also worth praising the number of positive efforts being put in place to help advance and develop research on evolutionary generative models.

10.10 Conclusions

In this chapter, we presented a comprehensive survey on evolutionary generative models, analysing their main properties, along with a taxonomy to categorise these models. The lack of a well-established categorisation of these models in the literature led us to identify four main categories of evolutionary generative models: *evolutionary computation without machine learning*, *evolutionary computation aided by machine learning*, *machine learning aided by evolutionary computation* and *machine learning evolved by evolutionary computation*.

The importance of this contribution is twofold. First, it aims to standardise or at least provide a basis for further classifications of evolutionary generative models while easing the process of analysing the existing body of research. Second, we collected and categorised some of the most prominent literature concerning evolutionary generative models. We are aware that the work in evolutionary generative models is extensive, and therefore it is not feasible to carry out an exhaustive survey of all existing approaches.

Early instances of evolutionary generative models started as plain evolutionary computation approaches. With the growth in the research and adoption of machine learning seen within the past decades, the evolutionary computation approaches evolved into models incorporating machine learning techniques. For this reason, we have performed an in-depth listing of applications, surveys, position papers and

other research pertaining to and spanning from the beginnings of the use of evolutionary computation as a generative model to the state-of-the-art approaches. Overall, more than 200 publications ranging from plain evolutionary computation systems to evolutionary generative machine learning for the generation of images, music, text, as well as other domains were explored and classified. Each publication was classified according to the main properties that we have identified, namely, application domain, evolutionary algorithm, population, representation and fitness.

Furthermore, the extent of the present study made it possible to identify open problems and challenges for evolutionary generative models. Namely, challenges such as the need for adequate metrics for evaluating and validating generative performance, the aggregation of the generated outputs in a single, organised solution space as well as the search for well-suited operators capable of efficiently navigating through the associated solutions space is at the forefront of current open problems. It is worth noting that the growth of evolutionary generative models is tied to the maturity of the field of evolutionary machine learning, which, as an emerging field, is primarily propelled by the transparency and availability of the implemented systems to the general public. This challenge is in no way less relevant than the above-mentioned ones.

With the emergence of more promising methods such as GANs and other deep learning techniques, the use of evolutionary computation in the context of generative models might seem lost at first sight. However, based on recent trends demonstrated throughout this chapter, the future of the topic seems to point towards the symbiosis between both fields, where the power of machine learning interplays with the representation flexibility of evolutionary computation to improve the generative performance and computational efficiency of current and future evolutionary generative models.

Acknowledgements This work is funded by the FCT—Foundation for Science and Technology, I.P./MCTES through national funds (PIDDAC), within the scope of CISUC R&D Unit-UIDB/00326/2020 or project code UIDP/00326/2020 and by European Social Fund, through the Regional Operational Program Centro 2020, under the FCT grant SFRH/BD/08254/2021.

References

1. Al-Dujaili, A., Schmiedlechner, T., O'Reilly, U.M.: Towards distributed coevolutionary GANs. In: Association for the Advancement of Artificial Intelligence (AAAI) 2018 Fall Symposium (2018)
2. Al-Najjar, K., Hämäläinen, M.: A master-apprentice approach to automatic creation of culturally satirical movie titles. In: Krahmer, E., Gatt, A., Goudbeek, M. (eds.) Proceedings of the 11th International Conference on Natural Language Generation, Tilburg University, The Netherlands, pp. 274–283. Association for Computational Linguistics (2018). Accessed from 5–8 Nov 2018
3. Albarracín-Molina, D.D., Raglio, A., Rivas-Ruiz, F., Vico, F.J.: Using formal grammars as musical genome. *Appl. Sci.* **11**(9) (2021)
4. Alvarado, F.H.C., Lee, W.H., Huang, Y.H., Chen, Y.S.: Melody similarity and tempo diversity as evolutionary factors for music variations by genetic algorithms. In: Cardoso, F.A., Machado, P., Veale, T., Cunha, J.M. (eds.) Proceedings of the Eleventh International Confer-

- ence on Computational Creativity, ICCC 2020, Coimbra, Portugal, pp. 251–254. Association for Computational Creativity (ACC) (2020). 7–11 Sept 2020
- 5. Angeline, P.J.: Evolving fractal movies. In: Proceedings of the 1st Annual Conference on Genetic Programming, pp. 503–511. MIT Press (1996)
 - 6. Ashlock, D.A.: Evolutionary exploration of the mandelbrot set. In: IEEE International Conference on Evolutionary Computation, CEC 2006, part of WCCI 2006, Vancouver, BC, Canada, pp. 2079–2086. IEEE (2006). Accessed from 16–21 July 2006
 - 7. Aupetit, S., Bordeau, V., Monmarché, N., Slimane, M., Venturini, G.S.: Interactive evolution of ant paintings. In: Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2003, Canberra, Australia, pp. 1376–1383. IEEE (2003). Accessed from 8–12 Dec 2003
 - 8. Baeta, F., Correia, J., Martins, T., Machado, P.: Speed benchmarking of genetic programming frameworks. In: Chicano, F., Krawiec, K. (eds.), GECCO '21: Genetic and Evolutionary Computation Conference, Lille, France, pp. 768–775. ACM (2021). Accessed from 10–14 July 2021
 - 9. Baeta, F., Correia, J., Martins, T., Machado, P.: Tensor GP - genetic programming engine in tensor flow. In: Castillo, P.A., Laredo, J.L.J. (eds.), Applications of Evolutionary Computation - 24th International Conference, Evo Applications 2021, Held as Part of Evo Star 2021, Virtual Event, Proceedings. Lecture Notes in Computer Science, vol. 12694, pp. 763–778. Springer (2021). Accessed from 7–9 April 2021
 - 10. Baeta, F., Correia, J., Martins, T., Machado, P.: Exploring expression-based generative adversarial networks. In: Fieldsend, J.E., Wagner, M. (eds.) GECCO '22: Genetic and Evolutionary Computation Conference, Companion Volume, Boston, Massachusetts, USA, pp. 1862–1869. ACM (2022). 9–13 July 2022
 - 11. Baker, E., Seltzer, M.: Evolving line drawings. In: Forrest, S. (ed.) Proceedings of the 5th International Conference on Genetic Algorithms, Urbana-Champaign, IL, USA, p. 627. Morgan Kaufmann (1993)
 - 12. Baluja, S., Pomerleau, D., Jochem, T.: Towards automated artificial evolution for computer-generated images. *Connect. Sci.* **6**(2–3), 325–354 (1994)
 - 13. Bell, C.: Algorithmic music composition using dynamic markov chains and genetic algorithms. *J. Comput. Sci. Coll.* **27**(2), 99–107 (2011)
 - 14. Bentley, P.J., Kumar, S.: Three ways to grow designs: a comparison of embryogenies for an evolutionary design problem. In: Banzhaf, W., Daida, J.M., Eiben, A.E., Garzon, M.H., Honavar, V.G., Jakielka, M.J., Smith, R.E. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999), Orlando, Florida, USA, pp. 35–43. Morgan Kaufmann (1999). Accessed from 13–17 July 1999
 - 15. Bergen, S., Ross, B.J.: Automatic and interactive evolution of vector graphics images with genetic algorithms. *Visual Comput.* **28**(1), 35–45 (2012)
 - 16. Bergen, S., Ross, B.J.: Aesthetic 3D model evolution. *Genetic Program. Evolvable Mach.* **14**(3), 339–367 (2013)
 - 17. Biles, J., Anderson, P., Loggi, L.: Neural Network Fitness Functions for a Musical IGA. In: RIT Scholar Works, Rochester Institute of Technology (1996)
 - 18. Biles, J.A.: GenJam: a genetic algorithm for generating Jazz Solos. In: Proceedings of the 1994 International Computer Music Conference, ICMC 1994, Aarhus, Denmark. Michigan Publishing (1994). Accessed from 12–17 Sept 1994
 - 19. Bilotta, E., Pantano, P., Cupellini, E., Rizzuti, C.: Evolutionary methods for melodic sequences generation from non-linear dynamic systems. In: Giacobini, M., Brabazon, A., Cagnoni, S., Caro, G.D., Drechsler, R., Farooq, M., Fink, A., Lutton, E., Machado, P., Minner, S., O'Neill, M., Romero, J., Rothlauf, F., Squillero, G., Takagi, H., Uyar, S., Yang, S. (eds.) Applications of Evolutionary Computing, EvoWorkshops 2007: EvoCoMnet, EvoFIN, EvoIASP, EvoINTERACTION, EvoMUSART, EvoSTOC and EvoTransLog, Valencia, Spain, Proceedings. Lecture Notes in Computer Science, vol. 4448, pp. 585–592. Springer (2007). Accessed from 11–13 Apr 2007
 - 20. Binkowski, M., Sutherland, D.J., Arbel, M., Gretton, A.: Demystifying MMD GANs. In: 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, Conference Track Proceedings (2018). Accessed from April 30–May 3 2018

21. Bontrager, P., Lin, W., Togelius, J., Risi, S.: Deep interactive evolution. In: Liapis, A., Cardalda, J.J.R., Ekárt, A. (eds.) Computational Intelligence in Music, Sound, Art and Design - 7th International Conference, EvoMUSART 2018, Parma, Italy, Proceedings. Lecture Notes in Computer Science, vol. 10783, pp. 267–282. Springer, (2018). Accessed from 4–6 April 2018
22. Bontrager, P., Roy, A., Togelius, J., Memon, N., Ross, A.: Deep master prints: generating master prints for dictionary attacks via latent variable evolution. In: 9th IEEE International Conference on Biometrics Theory, Applications and Systems, BTAS 2018, Redondo Beach, CA, USA, pp. 1–9. IEEE (2018). Accessed from 22–25 Oct 2018
23. Bontrager, P., Roy, A., Togelius, J., Memon, N., Ross, A.: Deep master prints: generating master prints for dictionary attacks via latent variable evolution. In: 9th IEEE International Conference on Biometrics Theory, Applications and Systems, BTAS 2018, Redondo Beach, CA, USA, pp. 1–9. IEEE (2018). Accessed from 22–25 Oct 2018
24. Brown, J.A., Ashlock, D., Orth, J., Houghten, S.: Autogeneration of fractal photographic mosaic images. In: Proceedings of the IEEE congress on evolutionary computation, CEC 2011, New Orleans, LA, USA, pp. 1116–1123. IEEE (2011). Accessed from 5–8 June 2011
25. Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D.M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., Amodei, D.: Language models are few-shot learners. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems, NeurIPS 2020, Virtual Event, pp. 1877–1901 (2020). Accessed from 6–12 Dec 2020
26. Browne, C., Maire, F.: Evolutionary game design. *IEEE Trans. Comput. Intell. AI in Games* **2**(1), 1–16 (2010)
27. Burton, A.R., Vladimirova, T.: Genetic algorithm utilising neural network fitness evaluation for musical composition. In: Smith, G.D., Steele, N.C., Albrecht, R.F. (eds.) Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms, ICANNGA 1997, Norwich, UK, 1997, pp. 219–223. Springer (1997)
28. Byrne, J., McDermott, J., López, E.G., O'Neill, M.: Implementing an intuitive mutation operator for interactive evolutionary 3D design. In: Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2010, Barcelona, Spain, pp. 1–7. IEEE (2010). Accessed from 18–23 July 2010
29. Cardona, A.B., Hansen, A.W., Togelius, J., Friberger, M.G.: Open trumps, a data game. In: Mateas, M., Barnes, T., Bogost, I. (eds.) Proceedings of the 9th International Conference on the Foundations of Digital Games, FDG 2014, Liberty of the Seas, Caribbean. Society for the Advancement of the Science of Digital Games (2014). Accessed from 3–7 Apr 2014
30. Cardona, A.B., Togelius, J., Nelson, M.J.: Competitive coevolution in Ms. Pac-Man. In: Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2013, Cancun, Mexico, pp. 1403–1410. IEEE (2013). Accessed from 20–23 June 2013
31. Chapuis, J., Lutton, E.: Artie-fract: interactive evolution of fractals. In: International Conference on Generative Art (2001)
32. Chen, J., Chen, X., Yang, J., Shan, S., Wang, R., Gao, W.: Optimization of a training set for more robust face detection. *Pattern Recogn.* **42**(11), 2828–2840 (2009)
33. Chen, S., Wang, W., Xia, B., You, X., Peng, Q., Cao, Z., Ding, W.: CDE-GAN: cooperative dual evolution-based generative adversarial network. *IEEE Trans. Evol. Comput.* **25**(5), 986–1000 (2021)
34. Cho, H., Kim, Y.: Stabilized training of generative adversarial networks by a genetic algorithm. In: López-Ibáñez, M., Auger, A., Stützle, T. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO 2019, Prague, Czech Republic, pp. 51–52. ACM (2019). Accessed from 13–17 July 2019
35. Collomosse, J.P.: Evolutionary search for the artistic rendering of photographs. In: Romero, J., Machado, P. (eds.) *The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music*, Natural Computing Series, pp. 39–62. Springer (2008)

36. Colton, S.: Automatic invention of fitness functions with application to scene generation. In: Giacobini, M., Brabazon, A., Cagnoni, S., Caro, G.D., Drechsler, R., Ekárt, A., Esparcia-Alcázar, A., Farooq, M., Fink, A., McCormack, J., O'Neill, M., Romero, J., Rothlauf, F., Squillero, G., Uyar, S., Yang, S. (eds.) Applications of Evolutionary Computing, EvoWorkshops 2008: EvoCOMNET, EvoFIN, EvoHOT, EvoIASP, EvoMUSART, EvoNUM, EvoSTOC, and EvoTransLog, Naples, Italy, Proceedings, Lecture Notes in Computer Science, vol. 4974, pp. 381–391. Springer (2008). Accessed from 26–28 March 2008
37. Colton, S.: Evolving a library of artistic scene descriptors. In: Machado, P., Romero, J., Carballal, A. (eds.) Evolutionary and Biologically Inspired Music, Sound, Art and Design - First International Conference, EvoMUSART 2012, Málaga, Spain, Proceedings. Lecture Notes in Computer Science, vol. 7247, pp. 35–47. Springer (2012). Accessed from 11–13 Apr 2012
38. Colton, S.: Evolving neural style transfer blends. In: Romero, J., Martins, T., Rodríguez-Fernández, N. (eds.) Artificial Intelligence in Music, Sound, Art and Design - 10th International Conference, EvoMUSART 2021, Held as Part of EvoStar 2021, Virtual Event, Proceedings. Lecture Notes in Computer Science, vol. 12693, pages 65–81. Springer (2021). Accessed from 7–9 Apr 2021
39. Cook, M., Colton, S.: redesigning computationally creative systems for continuous creation. In: Pachet, F., Jordanous, A., León, C. (eds.) Proceedings of the Ninth International Conference on Computational Creativity, ICCC 2018, Salamanca, Spain, pp. 32–39. Association for Computational Creativity (ACC) (2018). Accessed from 25–29 June 2018
40. Cook, M., Colton, S., Gow, J.: The ANGELINA videogame design system - Part I. IEEE Trans. Comput. Intell. AI in Games **9**(2), 192–203 (2017)
41. Correia, J., Machado, P., Romero, J., Carballal, A.: Evolving figurative images using expression-based evolutionary art. In: Maher, M.L., Veale, T., Saunders, R., Bown, O. (eds.) Proceedings of the Fourth International Conference on Computational Creativity, ICCC 2013, Sidney, Australia, pp. 24–31 (2013). Accessed from 12–14 June 2013
42. Correia, J., Machado, P., Romero, J., Martins, P., Cardoso, F.A.: Breaking the mould an evolutionary quest for innovation through style change. In: Veale, F.A., Cardoso, F.A. (eds.) Computational Creativity - The Philosophy and Engineering of Autonomously Creative Systems, pp. 353–398. Springer (2019)
43. Correia, J., Martins, T., Machado, P.: Evolutionary data augmentation in deep face detection. In: López-Ibáñez, M., Auger, A., Stützle, T. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO 2019, Prague, Czech Republic, pp. 163–164. ACM (2019). Accessed from 13–17 July 2019
44. Correia, J., Martins, T., Martins, P., Machado, P.: X-faces: the eXploit is out there. In: Pachet, F., Cardoso, A., Corruble, V., Ghedini, F. (eds.) Proceedings of the Seventh International Conference on Computational Creativity, ICCC 2016, UPMC, Paris, France, pp. 164–171. Sony CSL (2016). Accessed from June 27–July 1 2016
45. Correia, J., Vieira, L., Rodriguez-Fernandez, N., Romero, J., Machado, P.: Evolving image enhancement pipelines. In: Romero, J., Martins, T., Rodríguez-Fernández, N. (eds.) Artificial Intelligence in Music, Sound, Art and Design - 10th International Conference, EvoMUSART 2021, Held as Part of EvoStar 2021, Virtual Event, Proceedings. Lecture Notes in Computer Science, vol. 12693, pp. 82–97. Springer (2021). Accessed from 7–9 Apr 2021
46. Costa, V., Lourenço, N., Correia, J., Machado, P.: COEGAN: evaluating the coevolution effect in generative adversarial networks. In: Auger, A., Stützle, T. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019, Prague, Czech Republic, pp. 374–382. ACM (2019). Accessed from 13–17 July 2019
47. Costa, V., Lourenço, N., Correia, J., Machado, P.: Exploring the evolution of gans through quality diversity. In: Coello, C.A.C. (ed.) GECCO '20: Genetic and Evolutionary Computation Conference, Cancún Mexico, pp. 297–305. ACM (2020). Accessed from 8–12 July 2020
48. Costa, V., Lourenço, N., Correia, J., Machado, P.: Demonstrating the evolution of GANs through t-SNE. In: Castillo, P.A., Laredo, J.L.J. (eds.) Applications of Evolutionary Computation - 24th International Conference, EvoApplications 2021, Held as Part of EvoStar 2021,

- Virtual Event, Proceedings. Lecture Notes in Computer Science, vol. 12694, pp. 618–633. Springer (2021). Accessed from 7–9 Apr 2021
- 49. Costa, V., Lourenço, N., Machado, P.: Coevolution of generative adversarial networks. In: Kaufmann, P., Castillo, P.A. (eds.) Applications of Evolutionary Computation - 22nd International Conference, EvoApplications 2019, Held as Part of EvoStar 2019, Leipzig, Germany, Proceedings. Lecture Notes in Computer Science, vol. 11454, pp. 473–487. Springer (2019). Accessed from 24–26 Apr 2019
 - 50. Cunha, J.M., Martins, P., Lourenço, N., Machado, P.: Emojinating co-creativity: integrating self-evaluation and context-adaptation. In: Cardoso, F.A., Machado, P., Veale, T., Cunha, J.M. (eds.) Proceedings of the Eleventh International Conference on Computational Creativity, ICCC 2020, Coimbra, Portugal, pp. 85–88. Association for Computational Creativity (ACC) (2020). Accessed from 7–11 Sept 2020
 - 51. Dahlstedt, P.: Sounds Unheard of: Evolutionary Algorithms as Creative Tools for the Contemporary Composer. Ph.D. thesis, Chalmers, School of Architecture, University of Gothenburg (2004)
 - 52. Dawkins, R.: The Blind Watchmaker: Why the Evidence of Evolution Reveals a Universe Without Design. W. W. Norton & Company (1986)
 - 53. de la Puente, A.O., Alfonso, R.S., Alfonseca, M.: Automatic composition of music by means of grammatical evolution. In: Proceedings of the 2002 International Conference on APL: Array Processing Languages: Lore, Problems, and Applications, APL 2002, Madrid, Spain, pp. 148–155. ACM (2002). Accessed from 22–25 July 2002
 - 54. den Heijer, E., Eiben, A.E.: Evolving pop art using scalable vector graphics. In: Machado, P., Romero, J., Carballal, A. (eds.) Evolutionary and Biologically Inspired Music, Sound, Art and Design - First International Conference, EvoMUSART 2012, Málaga, Spain, Proceedings. Lecture Notes in Computer Science, vol. 7247 , pp. 48–59. Springer (2012). Accessed from 11–13 Apr 2012
 - 55. DiPaola, S.: Evolving creative portrait painter programs using darwinian techniques with an automatic fitness function. In: EVA 2005 London Conference (2005)
 - 56. DiPaola, S.R., Gabor, L.: Incorporating characteristics of human creativity into an evolutionary art algorithm. In: Thierens, D. (ed.) Genetic and Evolutionary Computation Conference, GECCO 2007, Proceedings, London, England, UK, Companion Material, pp. 2450–2456. ACM (2007). Accessed from 7–11 July 2007
 - 57. Donnelly, P.J., Sheppard, J.W.: Evolving four-part harmony using genetic algorithms. In: Chio, C.D., Brabazon, A., Caro, G.A.D., Drechsler, R., Farooq, M., Grahl, J., Greenfield, G., Prins, C., Romero, J., Squillero, G., Tarantino, E., Tettamanzi, A., Urquhart, N., Etaner-Uyar, A.S. (eds.) Applications of Evolutionary Computation - EvoApplications 2011: EvoCOMNET, EvoFIN, EvoHOT, EvoMUSART, EvoSTIM, and EvoTRANSLOG, Torino, Italy, Proceedings, Part II. Lecture Notes in Computer Science, vol. 6625, pp. 273–282. Springer (2011). Accessed from 27–29 April 2011
 - 58. Draves, S.: The electric sheep screen-saver: a case study in aesthetic evolution. In: Rothlauf, F., Branke, J., Cagnoni, S., Corne, D.W., Drechsler, R., Jin, Y., Machado, P., Marchiori, E., Romero, J., Smith, G.D., Squillero, G. (eds.) Applications of Evolutionary Computing, EvoWorkshops 2005: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoMUSART, and EvoSTOC, Lausanne, Switzerland, Proceedings. Lecture Notes in Computer Science, vol. 3449, pp. 458–467. Springer (2005). Accessed from March 30–April 1, 2005
 - 59. Dubbin, G.A., Stanley, K.O.: Learning to dance through interactive evolution. In: Chio, C.D., Brabazon, A., Caro, G.A.D., Ebner, M., Farooq, M., Fink, A., Grahl, J., Greenfield, G., Machado, P., O'Neill, M., Tarantino, E., Urquhart, N. (eds.) Applications of Evolutionary Computation, EvoApplications 2010: EvoCOMNET, EvoENVIRONMENT, EvoFIN, Evo-MUSART, and EvoTRANSLOG, Istanbul, Turkey, Proceedings, Part II. Lecture Notes in Computer Science, vol. 6025 , pp. 331–340. Springer (2010). Accessed from 7–9 Apr 2010
 - 60. Eigenfeldt, A., Bown, O., Brown, A.R., Gifford, T.: Flexible generation of musical form: beyond mere generation. In: Pachet, F., Cardoso, A., Corruble, V., Ghedini, F. (eds.) Proceedings of the Seventh International Conference on Computational Creativity, ICCC 2016, UPMC, Paris, France, pp. 264–271. Sony CSL (2016). Accessed from June 27–July 1, 2016

61. Eigenfeldt, A., Pasquier, P.: Populations of populations: composing with multiple evolutionary algorithms. In: Machado, P., Romero, J., Carballal, A. (eds.) Evolutionary and Biologically Inspired Music, Sound, Art and Design - First International Conference, EvoMUSART 2012, Málaga, Spain. Proceedings. Lecture Notes in Computer Science, vol. 7247, pp. 72–83. Springer (2012). Accessed from 11–13 Apr 2012
62. Ekárt, A.: Genetic programming for the design of lace knitting stitch patterns. In: Ellis, R., Allen, T., Petridis, M. (eds.) Applications and Innovations in Intelligent Systems XV, pp. 261–274. Springer, London (2008)
63. Ekern, E.G., Gambäck, B.: Interactive, efficient and creative image generation using compositional pattern-producing networks. In: Romero, J., Martins, T., Rodríguez-Fernández, N. (eds.) Artificial Intelligence in Music, Sound, Art and Design - 10th International Conference, EvoMUSART 2021, Held as Part of EvoStar 2021, Virtual Event, Proceedings. Lecture Notes in Computer Science, vol. 12693, pp. 131–146. Springer (2021). Accessed from 7–9 Apr 2021
64. Fernandes, P., Correia, J., Machado, P.: Evolutionary latent space exploration of generative adversarial networks. In: Castillo, P.A., Laredo, J.L.J., de Vega, F.F. (eds.) Applications of Evolutionary Computation - 23rd European Conference, EvoApplications 2020, Held as Part of EvoStar 2020, Seville, Spain, Proceedings. Lecture Notes in Computer Science, vol. 12104, pp. 595–609. Springer (2020). Accessed from 15–17 Apr 2020
65. Flores, D., Hemberg, E., Toutouh, J., O'Reilly, U.: Coevolutionary generative adversarial networks for medical image augmentation at scale. In: Fieldsend, J.E., Wagner, M. (eds.) GECCO '22: Genetic and Evolutionary Computation Conference, Boston, Massachusetts, USA, pp. 367–376. ACM (2022). Accessed from 9–13 July 2022
66. Frade, M., de Vega, F.F., Cotta, C.: Modelling video games' landscapes by means of genetic terrain programming - a new approach for improving users' experience. In: Giacobini, M., Brabazon, A., Cagnoni, S., Caro, G.D., Drechsler, R., Ekárt, A., Esparcia-Alcázar, A., Farooq, M., Fink, A., McCormack, J., O'Neill, M., Romero, J., Rothlauf, F., Squillero, G., Uyar, S., Yang, S. (eds.) Applications of Evolutionary Computing, EvoWorkshops 2008: EvoCOMNET, EvoFIN, EvoHOT, EvoIASP, EvoMUSART, EvoNUM, EvoSTOC, and EvoTransLog, Naples, Italy. Proceedings. Lecture Notes in Computer Science, vol. 4974, pp. 485–490. Springer (2008). Accessed from 26–28 March 2008
67. Galanter, P.: What is generative art? Complexity theory as a context for art theory. In: In GA2003 – 6th Generative Art Conference (2003)
68. Garcíarena, U., Santana, R., Mendiburu, A.: Evolved GANs for generating pareto set approximations. In: Aguirre, H.E., Takadama, H.E. (eds.) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2018, Kyoto, Japan, pp. 434–441. ACM (2018). Accessed from 15–19 July 2018
69. Gartland-Jones, A.: MusicBlox: a real-time algorithmic composition system incorporating a distributed interactive genetic algorithm. In: Raidl, G.R., Meyer, J., Middendorf, M., Cagnoni, S., Cardalda, J.J.R., Corne, D., Gottlieb, J., Guillot, A., Hart, E., Johnson, C.G., Marchiori, E. (eds.) Applications of Evolutionary Computing, EvoWorkshop 2003: EvoBIO, EvoCOP, EvoIASP, EvoMUSART, EvoROB, and EvoSTIM, Essex, UK, Proceedings. Lecture Notes in Computer Science vol. 2611, pp. 490–501. Springer (2003). Accessed from 14–16 Apr 2003
70. Gervás, P.: Evolutionary elaboration of daily news as a poetic stanza. In: IX Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados - MAEB 2013, Universidad Complutense de Madrid, Spain (2013)
71. Gibson, P., Byrne, J.: NEUROGEN, musical composition using genetic algorithms and cooperating neural networks. In: 1991 Second International Conference on Artificial Neural Networks, pp. 309–313 (1991)
72. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, Montreal, Quebec, Canada, pp. 2672–2680 (2014). Accessed from 8–13 Dec 2014

73. Grabe, I., Zhu, J., Agirrezabal, M.: Fashion style generation: evolutionary search with gaussian mixture models in the latent space. In: Martins, T., Rodríguez-Fernández, N., Rebelo, S.M. (eds.) Artificial Intelligence in Music, Sound, Art and Design - 11th International Conference, EvoMUSART 2022, Held as Part of EvoStar 2022, Madrid, Spain, Proceedings. Lecture Notes in Computer Science, vol. 13221, pp.84–100. Springer (2022). Accessed from 20–22 Apr 2022
74. Graf, J., Banzhaf, W.: Interactive evolution of images. In: McDonnell, J.R., Reynolds, R.G., Fogel, D.B. (eds.) Proceedings of the Fourth Annual Conference on Evolutionary Programming, EP 1995, San Diego, CA, USA, pp. 53–65. MIT Press (1995). Accessed from 1–3 March 1995
75. Greenfield, G.: A platform for evolving controllers for simulated drawing robots. In: Machado, P., Romero, J., Carballal, A. (eds.) Evolutionary and Biologically Inspired Music, Sound, Art and Design - First International Conference, EvoMUSART 2012, Málaga, Spain, Proceedings. Lecture Notes in Computer Science, vol. 7247, pp. 108–116. Springer (2012). Accessed from 11–13 Apr 2012
76. Greenfield, G.R.: New directions for evolving expressions. In: Bridges: Mathematical Connections in Art, Music, and Science, pp. 29–36 (1998)
77. Greenfield, G.R.: Evolving expressions and art by choice. Leonardo **33**(2), 93–99 (2000)
78. Greenfield, G.R.: Tilings of sequences of co-evolved images. In: Raidl, G.R., Cagnoni, S., Branke, J., Corne, D., Drechsler, R., Jin, Y., Johnson, C.G., Machado, P., Marchiori, E., Rothlauf, F., Smith, G.D., Squillero, G. (eds.) Applications of Evolutionary Computing, EvoWorkshops 2004: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoMUSART, and EvoSTOC, Coimbra, Portugal, Proceedings. Lecture Notes in Computer Science, vol. 3005, pp. 427–436. Springer (2004). Accessed from 5–7 Apr 2004
79. Greenfield, G.R.: Evolutionary methods for ant colony paintings. In: Raidl, G.R., Cagnoni, S., Branke, J., Corne, D., Drechsler, R., Jin, Y., Johnson, C.G., Machado, P., Marchiori, E., Rothlauf, F., Smith, G.D., Squillero, G. (eds.) Applications of Evolutionary Computing, EvoWorkshops 2005: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoMUSART, and EvoSTOC, Lausanne, Switzerland, Proceedings. Lecture Notes in Computer Science, vol. 3449, pp. 478–487. Springer (2005). Accessed from March 30–April 1 2005
80. Hart, D.A.: Toward greater artistic control for interactive evolution of images and animation. In MGiacobini, M., Brabazon, A., Cagnoni, S., Caro, G.D., Drechsler, R., Farooq, M., Fink, A., Lutton, E., Machado, P., Minner, S., O'Neill, M., Romero, J., Rothlauf, F., Squillero, G., Takagi, H., Uyar, S., Yang, S. (eds.) Applications of Evolutionary Computing, EvoWorkshops 2007: EvoCoMnet, EvoFIN, EvoIASP, EvoINTERACTION, EvoMUSART, EvoSTOC and EvoTransLog, Valencia, Spain, Proceedings. Lecture Notes in Computer Science, vol. 4448, pp. 527–536. Springer (2007). Accessed from 11–13 Apr 2007
81. Hemberg, E., Toutouh, J., Al-Dujaili, A., Schmiedlechner, T., O'Reilly, U.: Spatial coevolution for generative adversarial network training. ACM Trans. Evol. Learn. Optim. **1**(2), 6:1–6:28 (2021)
82. Hervás, R., Robinson, J., Gervás, P.: Evolutionary assistance in alliteration and allelic drivel. In: Giacobini, M., Brabazon, A., Cagnoni, S., Caro, G.D., Drechsler, R., Farooq, M., Fink, A., Lutton, E., Machado, P., Minner, S., O'Neill, M., Romero, J., Rothlauf, F., Squillero, G., Takagi, H., Uyar, S., Yang, S. (eds.) Applications of Evolutionary Computing, EvoWorkshops 2007: EvoCoMnet, EvoFIN, EvoIASP, EvoINTERACTION, EvoMUSART, EvoSTOC and EvoTransLog, Valencia, Spain, Proceedings. Lecture Notes in Computer Science, vol. 4448 pp. 537–546. Springer (2007). Accessed from 11–13 Apr 2007
83. Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S.: GANs trained by a two time-scale update rule converge to a local nash equilibrium. In: Guyon, I., von Luxburg, U., Bengio, S., Wallach, H.M., Fergus, R., Vishwanathan, S.V.N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, Long Beach, CA, USA, pp. 6626–6637 (2017). Accessed from 4–9 Dec 2017
84. Hickinbotham, S.J., Stepney, S.: Augmenting live coding with evolved patterns. In: Johnson, C.G., Ciesielski, V., Correia, J., Machado, P. (eds.,) Evolutionary and Biologically Inspired

- Music, Sound, Art and Design - 5th International Conference, EvoMUSART 2016, Porto, Portugal, Proceedings. Lecture Notes in Computer Science, vol. 9596, pp. 31–46. Springer (2016). March 30–April 1, 2016
- 85. Hofmann, D.M.: A genetic programming approach to generating musical compositions. In: Johnson, C.G., Carballal, A., Correia, J. (eds.) Evolutionary and Biologically Inspired Music, Sound, Art and Design - 4th International Conference, EvoMUSART 2015, Copenhagen, Denmark, Proceedings. Lecture Notes in Computer Science, vol. 9027, pp. 89–100. Springer (2015). Accessed from 8–10 Apr 2015
 - 86. Hoover, A., Szerlip, P., Stanley, K.: Generating musical accompaniment through functional scaffolding. In: Proceedings of the Eighth Sound and Music Computing Conference (SMC 2011), pp. 161–168 (2011)
 - 87. Hoover, A.K., Cachia, W., Liapis, A., Yannakakis, G.N.: Audio in space: exploring the creative fusion of generative audio, visuals and gameplay. In: Johnson, C.G., Carballal, A., Correia, J. (eds.), Evolutionary and Biologically Inspired Music, Sound, Art and Design - 4th International Conference, EvoMUSART 2015, Copenhagen, Denmark, Proceedings. Lecture Notes in Computer Science, vol. 9027, pp. 101–112. Springer (2015). Accessed from 8–10 Apr 2015
 - 88. Hoover, A.K., Rosario, M.P., Stanley, K.O.: Scaffolding for interactively evolving novel drum tracks for existing songs. In: Giacobini, M., Brabazon, A., Cagnoni, S., Caro, G.D., Drechsler, R., Ekárt, A., Esparcia-Alcázar, A., Farooq, M., Fink, A., McCormack, J., O'Neill, M., Romero, J., Rothlauf, F., Squillero, G., Uyar, S., Yang, S. (eds.) Applications of Evolutionary Computing, EvoWorkshops 2008: EvoCOMNET, EvoFIN, EvoHOT, EvoIASP, Evo-MUSART, EvoNUM, EvoSTOC, and EvoTransLog, Naples, Italy, Proceedings. Lecture Notes in Computer Science, vol. 4974, pp. 412–422. Springer (2008). Accessed from 26–28 March 2008
 - 89. Hoover, A.K., Stanley, K.O.: Exploiting functional relationships in musical composition. *Connect. Sci.* **21**(2&3), 227–251 (2009)
 - 90. Hoover, A.K., Szerlip, P.A., Stanley, K.O.: Functional scaffolding for composing additional musical voices. *Comput. Music J.* **38**(4), 80–99 (2014)
 - 91. Horner, A., Goldberg, D.E.: Genetic algorithms and computer-assisted music composition. In: Belew, R.K., Booker, L.B. (eds.) Proceedings of the 4th International Conference on Genetic Algorithms, San Diego, CA, USA, July 1991, pp. 437–441. Morgan Kaufmann (1991)
 - 92. Horowitz, D.: Generating rhythms with genetic algorithms. In: Hayes-Roth, B., Korf, R.E. (eds.) Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, Vol. 2, p. 1459. AAAI Press/The MIT Press (1994). Accessed from July 31–August 4, 1994
 - 93. Hui Yap, A.Y., Soong, H.-C., Hong Tse, S.S.: Real-time evolutionary music composition using JFUGUE and genetic algorithm. In: 2021 IEEE 19th Student Conference on Research and Development (SCORED), pp. 377–382 (2021)
 - 94. Ianigro, S., Bown, O.: Plecto: a low-level interactive genetic algorithm for the evolution of audio. In: Johnson, C.G., Ciesielski, V., Correia, J., Machado, P. (eds.) Evolutionary and Biologically Inspired Music, Sound, Art and Design - 5th International Conference, EvoMUSART 2016, Porto, Portugal. Proceedings. Lecture Notes in Computer Science, vol. 9596, pp. 63–78. Springer (2016). Accessed from March 30–April 1, 2016
 - 95. Jacob, B.L.: Composing with genetic algorithms. In: Proceedings of the 1995 International Computer Music Conference, ICMC 1995, Banff, AB, Canada. Michigan Publishing (1995). Accessed from 3–7 Sept 1995
 - 96. Johanson, B., Poli, R.: GP-music: an interactive genetic programming system for music generation with automated fitness raters. In: Koza, J.R., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, D.B., Garzon, M.H., Goldberg, D.E., Iba, H., Riolo, R. (eds.) Genetic Programming 1998: Proceedings of the Third Annual Conference, pp. 181–186. Morgan Kaufmann (1998)
 - 97. Kaliakatsos-Papakostas, M.A., Epitropakis, M.G., Floros, A., Vrahatis, M.N.: Interactive evolution of 8-bit melodies with genetic programming towards finding aesthetic measures

- for sound. In: Machado, P., Romero, J., Carballal, A. (eds.) *Evolutionary and Biologically Inspired Music. Sound, Art and Design*, pp. 141–152. Springer, Berlin Heidelberg (2012)
- 98. Khalifa, Y.M.A., Khan, B.K., Begovic, J., Wisdom, A., Wheeler, A.M.: Evolutionary music composer integrating formal grammar. In: Thierens, D. (ed.) *Genetic and Evolutionary Computation Conference, GECCO 2007, Proceedings, London, England, UK, Companion Material*, pp. 2519–2526. ACM (2007). Accessed from 7–11 July 2007
 - 99. Korde, C.G., K., M.R., V, M.H., N. K. Y.B.: Training of generative adversarial networks with hybrid evolutionary optimization technique. In: 2019 IEEE 16th India Council International Conference (INDICON), pp. 1–4 (2019)
 - 100. Koza, J.R.: Genetic programming as a means for programming computers by natural selection. *Stat Comput* 4(2) (1994)
 - 101. Kunitatsu, K., Ishikawa, Y., Takata, M., Joe, K.: A music composition model with genetic programming – a case study of chord progression and bassline. In: *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, pp. 256–262 (2015)
 - 102. Laine, P., Kuuskankare, M.: Genetic algorithms in musical style oriented generation. In: *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, Orlando, Florida, USA*, pp. 858–862. IEEE (1994). Accessed from 27–29 June 1994
 - 103. Levy, R.P.: A computational model of poetic creativity with neural network as measure of adaptive fitness. In: *Proceedings of the First Workshop on Creative Systems, International Conference of Case-Based Reasoning* (2001)
 - 104. Lewis, M.: Aesthetic evolutionary design with data flow networks. In: *International Conference on Generative Art* (2000)
 - 105. Lewis, M.: *Evolutionary Visual Art and Design*, pp. 3–37. Springer, Berlin Heidelberg (2008)
 - 106. Li, J., Jing, M., Lu, K., Ding, Z., Zhu, L., Huang, Z.: Leveraging the invariant side of generative zero-shot learning. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA*, pp. 7402–7411. Computer Vision Foundation/IEEE (2019). Accessed from 16–20 June 2019
 - 107. Li, J., Zhang, J., Gong, X., Lü, S.: Evolutionary generative adversarial networks with crossover based knowledge distillation. In: *International Joint Conference on Neural Networks, IJCNN 2021, Shenzhen, China*, pp. 1–8. IEEE (2021). Accessed from 18–22 July 2021
 - 108. Liapis, A.: Recomposing the Pokémon color palette. In: Sim, K., Kaufmann, P. (eds.) *Applications of Evolutionary Computation*, pp. 308–324. Springer International Publishing (2018)
 - 109. Liapis, A.: Artificial intelligence for designing games. In: Machado, P., Romero, J., Greenfield, G. (eds.) *The Handbook of Artificial Intelligence and the Arts*. Springer (2021)
 - 110. A. Liapis, H. Martínez, J. Togelius, and G. Yannakakis. Transforming Exploratory Creativity with DeLeNoX. In M. Maher, T. Veale, R. Saunders, and O. Bown, editors, *Proceedings of the 4th International Conference on Computational Creativity, ICC 2013*, pages 56–63. Faculty of Architecture, Design and Planning, The University of Sydney, 2013
 - 111. Liapis, A., Yannakakis, G.N., Togelius, J.: Adapting models of visual aesthetics for personalized content creation. *IEEE Trans. Comput. Intell. AI Games* 4(3), 213–228 (2012)
 - 112. Liu, Z., Wang, J., Liang, Z.: CatGAN: category-aware generative adversarial networks with hierarchical evolutionary learning for category text generation. In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA*, pp. 8425–8432. AAAI Press (2020). Accessed from 7–12 Feb 2020
 - 113. Lopes, D., Correia, J., Machado, P.: Adeia - evolving glyphs for aiding creativity in typeface design. In: *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion, GECCO '20*, pp. 97–98. Association for Computing Machinery (2020)
 - 114. Lopes, P., Liapis, A., Yannakakis, G.N.: Framing tension for game generation. In: Pachet, F., Cardoso, A., Corruble, V., Ghedini, F. (eds.) *Proceedings of the Seventh International Conference on Computational Creativity, ICC 2016, UPMC, Paris, France*, pp. 205–212. Sony CSL (2016). Accessed from June 27–July 1, 2016

115. Loughran, R., McDermott, J., O'Neill, M.: Tonality driven piano compositions with grammatical evolution. In: IEEE Congress on Evolutionary Computation, CEC 2015, Sendai, Japan, pp. 2168–2175. IEEE (2015). Accessed from 25–28 May 2015
116. Loughran, R., McDermott, J., O'Neill (2016) Grammatical music composition with dissimilarity driven hill climbing. In: Johnson, C.G., Ciesielski, V., Correia, J., Machado, P. (eds.) Evolutionary and Biologically Inspired Music, Sound, Art and Design - 5th International Conference, EvoMUSART 2016, Porto, Portugal, Proceedings. Lecture Notes in Computer Science, vol. 9596 , pp. 110–125. Springer (2016). Accessed from March 30–April 1, 2016
117. Loughran, R., O'Neill, M.: My little chucky: towards live-coding with grammatical evolution. In: Musical Metacreation (MuMe) (2017)
118. Loughran, R., O'Neill, M.: Evolutionary music: applying evolutionary computation to the art of creating music. *Genetic Program. Evolvable Mach.* **21**(1–2), 55–85 (2020)
119. Loughran, R., O'Neill, M.: the popular critic: evolving melodies with popularity driven fitness. In: MUME 2016 - The Fourth International Workshop on Musical Metacreation (2016)
120. Lucas, P., Martinho, C.: Stay awhile and listen to 3buddy, a co-creative level design support tool. In: Goel, A.K., Jordanous, A., Pease, A. (eds.) Proceedings of the Eighth International Conference on Computational Creativity, ICC 2017, Atlanta, Georgia, USA, pp. 205–212. Association for Computational Creativity (ACC) (2017). Accessed from 19–23 June 2017
121. Maçãs, C., Lourenço, N., Machado, P.: Interactive evolution of swarms for the visualisation of consumptions. In: Brooks, A.L., Brooks, E., Sylla, C. (eds.) Interactivity, Game Creation, Design, Learning, and Innovation - 7th EAI International Conference, ArtsIT 2018, and 3rd EAI International Conference, DLI 2018, ICTCC 2018, Braga, Portugal, Proceedings. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 265, pp. 101–110. Springer (2018). Accessed from 24–26 Oct 2018
122. MacCallum, R.M., Mauch, M., Burt, A., Leroi, A.M.: Evolution of music by public choice. *Proc. Natl. Acad. Sci.* **109**(30), 12081–12086 (2012)
123. Machado, P.: Evolutionary art and design: representation, fitness and interaction. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO '21, pp. 1002–1031, New York, NY, USA (2021). Association for Computing Machinery
124. Machado, P., Cardoso, A.: NEvAr - the assessment of an evolutionary art tool. In: Wiggins, G. (ed.) AISB'00 Symposium on Creative and Cultural Aspects and Applications of AI and Cognitive Science (2000)
125. Machado, P., Cardoso, A.: All the truth about NEvAr. *Appl. Intell.* **16**(2), 101–118 (2002)
126. Machado, P., Correia, J.: Semantic aware methods for evolutionary art. In: Arnold, D.V. (ed.) Genetic and Evolutionary Computation Conference, GECCO '14, Vancouver, BC, Canada, pp. 301–308. ACM (2014). Accessed from 12–16 July 2014
127. Machado, P., Correia, J., Romero, J.: Expression-based evolution of faces. In: Machado, P., Romero, J., Carballal, A. (eds.) Evolutionary and Biologically Inspired Music, Sound, Art and Design - First International Conference, EvoMUSART 2012, Málaga, Spain, Proceedings. Lecture Notes in Computer Science, vol. 7247 , pp. 187–198. Springer (2012). Accessed from 11–13 Apr 2012
128. Machado, P., Correia, J., Romero, J.: Improving face detection. In: Moraglio, A., Silva, S., Krawiec, K., Machado, P., Cotta, C. (eds.) Genetic Programming - 15th European Conference, EuroGP 2012, Málaga, Spain. Proceedings. Lecture Notes in Computer Science, vol. 7244, pp. 73–84. Springer (2012). Accessed from 11–13 Apr 2012
129. Machado, P., Graça, F.: Evolutionary pointillist modules: evolving assemblages of 3D objects. In: Giacobini, M., Brabazon, A., Cagnoni, S., Caro, G.D., Drechsler, R., Ekárt, A., Esparcia-Alcázar, A., Farooq, M., Fink, A., McCormack, J., O'Neill, M., Romero, J., Rothlauf, F., Squillero, G., Uyar, S., Yang, S. (eds.) Applications of Evolutionary Computing, EvoWorkshops 2008: EvoCOMNET, EvoFIN, EvoHOT, EvoIASP, EvoMUSART, EvoNUM, EvoSTOC, and EvoTransLog, Naples, Italy. Proceedings. Lecture Notes in Computer Science, vol. 4974, pp. 453–462. Springer (2008). Accessed from 26–28 March 2008
130. Machado, P., Martins, T., Amaro, H., Abreu, P.H.: An interface for fitness function design. In: Romero, J., McDermott, J., Correia, J. (eds.) Evolutionary and Biologically Inspired Music,

- Sound, Art and Design - Third European Conference, EvoMUSART 2014, Granada, Spain, Revised Selected Papers. Lecture Notes in Computer Science, vol. 8601, pp. 13–25. Springer (2014). Accessed from 23–25 Apr 2014
- 131. Machado, P., Nunes, H., Romero, J.: Graph-based evolution of visual languages. In: Chio, C.D., Brabazon, A., Caro, G.A.D., Ebner, M., Farooq, M., Fink, A., Grahl, J., Greenfield, G., Machado, P., O'Neill, M., Tarantino, E., Urquhart, N. (eds.) Applications of Evolutionary Computation, EvoApplications 2010: EvoCOMNET, EvoENVIRONMENT, EvoFIN, Evo-MUSART, and EvoTRANSLOG, Istanbul, Turkey, Proceedings, Part II. Lecture Notes in Computer Science, vol. 6025, pp. 271–280. Springer (2010). Accessed from 7–9 Apr 2010
 - 132. Machado, P., Pereira, L.: Photogrowth: non-photorealistic renderings through ant paintings. In: Soule, T., Moore, J.H. (eds.) Genetic and Evolutionary Computation Conference, GECCO '12, Philadelphia, PA, USA, pp. 233–240. ACM (2012). Accessed from 7–11 July 2012
 - 133. Machado, P., Romero, J., Manaris, B.Z.: Experiments in computational aesthetics. In: Romero, J., Machado, P. (eds.) The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music, Natural Computing Series, pp. 381–415. Springer (2008)
 - 134. Machado, P., Vinhas, A., Correia, J., Ekart, A.: Evolving ambiguous images. In: Yang, Q., Wooldridge, M.J. (eds.) Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, pp. 2473–2479. AAAI Press (2015). Accessed from 25–31 July 2015
 - 135. Machín, B., Nesmachnow, S., Toutouh, J.: Multi-target evolutionary latent space search of a generative adversarial network for human face generation. In: Fieldsend, J.E., Wagner, M. (eds.) GECCO '22: Genetic and Evolutionary Computation Conference, Companion Volume, Boston, Massachusetts, USA, pp. 1878–1886. ACM (2022). Accessed from 9–13 July 2022
 - 136. Manaris, B., Hughes, D., Vassilandonakis, Y.: Monterey mirror: combining Markov models, genetic algorithms , and power laws an experiment in interactive evolutionary music performance. In: Proceedings of 1st Workshop in Evolutionary Music, 2011 IEEE Congress on Evolutionary Computation (CEC 2011), pp. 33–40 (2011)
 - 137. Manaris, B., Roos, P., Machado, P., Krehbiel, D., Pellicoro, L., Romero, J.: A Corpus-based hybrid approach to music analysis and composition. In: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, Vancouver, British Columbia, Canada, pp. 839–845. AAAI Press (2007). Accessed from 22–26 July 2007
 - 138. Manaris, B., Vaughan, D., Wagner, C., Romero, J., Davis, R.B.: Evolutionary music and the zipf-mandelbrot law: developing fitness functions for pleasant music. In: Raidl, G.R., Meyer, J., Middendorf, M., Cagnoni, S., Cardalda, J.J.R., Corne, D., Gottlieb, J., Guillot, A., Hart, E., Johnson, C.G., Marchiori, E. (eds.) Applications of Evolutionary Computing, EvoWorkshop 2003: EvoBIO, EvoCOP, EvoIASP, EvoMUSART, EvoROB, and EvoSTIM, Essex, UK, Proceedings. Lecture Notes in Computer Science, vol. 2611 , pp. 522–534. Springer (2003). Accessed from 14–16 Apr 2003
 - 139. Manurung, H.M.: An Evolutionary Algorithm Approach to Poetry Generation . Ph.D. thesis, Institute for Communicating and Collaborative Systems, School of Informatics, University of Edinburgh (2003)
 - 140. Manurung, H.M., Ritchie, G.D., Thompson, H.S.: Towards A Computational Model Of Poetry Generation. Technical report, Institute for Communicating and Collaborative Systems, Division of Informatics, University of Edinburgh (2000)
 - 141. Manurung, R., Ritchie, G., Thompson, H.: Using genetic algorithms to create meaningful poetic text. *J. Exp. Theor. Artif. Intell.* **24**(1), 43–64 (2012)
 - 142. Martínez-Rodríguez, B.: A new fitness function for evolutionary music composition. In: Montiel, M., Agustín-Aquino, O.A., Gómez, F., Kastine, J., Lluis-Puebla, E., Milam, B. (eds.) Mathematics and Computation in Music - 8th International Conference, MCM 2022, Atlanta, GA, USA, Proceedings. Lecture Notes in Computer Science, vol. 13267, pp. 205–217. Springer (2022). Accessed from 21–24 June 2022
 - 143. Martins, T., Correia, J., Costa, E., Machado, P.: Evotype: evolutionary type design. In: Johnson, C.G., Carballal, A., Correia, J. (eds.) Evolutionary and Biologically Inspired Music, Sound, Art and Design - 4th International Conference, EvoMUSART 2015, Copenhagen, Denmark,

- Proceedings. Lecture Notes in Computer Science, vol. 9027, pp. 136–147. Springer (2015). Accessed from 8–10 Apr 2015
144. Martins, T., Correia, J., Costa, E., Machado, P.: Evolving Stencils for Typefaces: Combining Machine Learning, User's Preferences and Novelty. Complexity, 2019 (2019)
 145. McCormack, J., Bown, O.: Life's what you make: niche construction and evolutionary art. In: Giacobini, M., Brabazon, A., Cagnoni, S., Caro, G.A.D., Ekárt, A., Esparcia-Alcázar, A., Farooq, M., Fink, A., Machado, P., McCormack, J., O'Neill, M., Neri, F., Preuss, M., Rothlauf, F., Tarantino, E., Yang, S. (eds.) Applications of Evolutionary Computing, EvoWorkshops 2009: EvoCOMNET, EvoENVIRONMENT, EvoFIN, EvoGAMES, EvoHOT, EvoIASP, EvoINTERACTION, EvoMUSART, EvoNUM, EvoSTOC, EvoTRANSLOG, Tübingen, Germany. Proceedings. Lecture Notes in Computer Science, vol. 5484, pp. 528–537. Springer (2009). Accessed from 15–17 Apr 2009
 146. McCormack, J., Eldridge, A., Dorin, A., McIlwain, P.: Generative algorithms for making music: emergence, evolution, and ecosystems. In: Dean, R.T. (ed.) The Oxford Handbook of Computer Music, Oxford Handbooks. Oxford University Press (2011)
 147. McDermott, J., O'Reilly, U.: An executable graph representation for evolutionary generative music. In: Krasnogor, N., Lanzi, P.L. (eds.) 13th Annual Genetic and Evolutionary Computation Conference, GECCO 2011, Proceedings, Dublin, Ireland, pp. 403–410. ACM (2011). 12–16 July 2011
 148. McIntyre, N., Lapata, M.: Plot Induction and evolutionary search for story generation. In: Hajic, J., Carberry, S., Clark, S. (eds.) ACL 2010, Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, Uppsala, Sweden, pp. 1562–1572. The Association for Computer Linguistics (2010). Accessed from 11–16 July 2010
 149. Miikkulainen, R., Liang, J., Meyerson, E., Rawal, A., Fink, D., Francon, O., Raju, B., Shahrzad, H., Navruzyan, A., Duffy, N., Hodjat, B.: Evolving deep neural networks. In: Kozma, R., Alippi, C., Choe, Y., Morabito, F.C. (eds.) Artificial Intelligence in the Age of Neural Networks and Brain Computing, pp. 293–312. Academic (2019)
 150. Montero, C.A.S., Araki, K.: Is it correct? - Towards web-based evaluation of automatic natural language phrase generation. In: Calzolari, N., Cardie, C., Isabelle, P. (eds.) ACL 2006, 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics, Proceedings of the Conference, Sydney, Australia. The Association for Computer Linguistics (2006). Accessed from 17–21 July 2006
 151. Moroni, A., Manzolli, J., von Zuben, F., Gudwin, R.: Vox Populi: an interactive evolutionary system for algorithmic music composition. Leonardo Music J. **10**, 49–54 (2000)
 152. Morris, R.G., Burton, S.H., Bodily, P.M., Ventura, D.: Soup over bean of pure joy: culinary ruminations of an artificial chef. In: Maher, M.L., Hammond, K.J., Pease, A., y Pérez, R.P., Ventura, D., Wiggins, G.A. (eds.) Proceedings of the Third International Conference on Computational Creativity, ICCC 2012, Dublin, Ireland, pp. 119–125 (2012). Accessed from May 30 - June 1, 2012
 153. Muehlbauer, M., Burry, J., Song, A.: automated shape design by grammatical evolution. In: Correia, J., Ciesielski, V., Liapis, A. (eds.) Computational Intelligence in Music, Sound, Art and Design - 6th International Conference, EvoMUSART 2017, Amsterdam, The Netherlands, Proceedings. Lecture Notes in Computer Science, vol. 10198, pp. 217–229 (2017). Accessed from 19–21 Apr 2017
 154. Neufeld, C., Ross, B.J., Ralph, W.: The evolution of artistic filters. In: Romero, J., Machado, P. (eds.) The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music, Natural Computing Series, pp. 335–356. Springer (2008)
 155. Ng, A., Jordan, M.: On discriminative vs. generative classifiers: a comparison of logistic regression and Naive Bayes. In: Dietterich, T.G., Becker, S., Ghahramani, Z. (eds.) Advances in Neural Information Processing Systems 14, NIPS 2001, Vancouver, British Columbia, Canada, December 3–8, 2001, pp. 841–848. MIT Press (2001)
 156. Nguyen, A., Yosinski, J., Clune, J.: Deep neural networks are easily fooled: high confidence predictions for unrecognizable images. In: IEEE Conference on Computer Vision and Pattern

- Recognition, CVPR 2015, Boston, MA, USA, June 7–12, 2015, pp. 427–436. IEEE Computer Society (2015)
- 157. Nishio, K., Murakami, M., Mizutani, E., Honda, N.: Fuzzy Fitness Assignment in an Interactive Genetic Algorithm for a Cartoon Face Search, pp. 175–192. World Scientific (1997)
 - 158. Oliveira, H.G.: Automatic generation of poetry: an overview. In: 1st Seminar of Art, Music, Creativity and Artificial Intelligence (2009)
 - 159. Olseng, O., Gambäck, B.: Co-evolving melodies and harmonization in evolutionary music composition. In: Liapis, A., Cardalda, J.J.R., Ekárt, A. (eds.) Computational Intelligence in Music, Sound, Art and Design - 7th International Conference, EvoMUSART 2018, Parma, Italy, Proceedings. Lecture Notes in Computer Science, vol. 10783, pp. 239–255. Springer (2018). Accessed from 4–6 Apr 2018
 - 160. O'Reilly, U., Toutouh, J., Perttierra, M.A., Sanchez, D.P., Garcia, D., Lugo, A.E., Kelly, J., Hemberg, E.: Adversarial genetic programming for cyber security: a rising application domain where GP matters. *Genetic Program. Evolvable Mach.* **21**(1–2), 219–250 (2020)
 - 161. Parente, J., Martins, T., Bicker, J., Machado, P.: Which type is your type? In: Cardoso, F.A., Machado, P., Veale, T., Cunha, J.M. (eds.) Proceedings of the Eleventh International Conference on Computational Creativity, ICCC 2020, Coimbra, Portugal, pp. 476–483. Association for Computational Creativity (ACC) (2020). Accessed from 7–11 Sept 2020
 - 162. Pearce, M., Wiggins, G.A.: Towards a framework for the evaluation of machine compositions. In: Proceedings of the AISB'01 Symposium on Artificial Intelligence and Creativity in the Arts and Sciences, pp. 22–32. Society for the Study of Artificial Intelligence and the Simulation of Behaviour (AISB) (2001)
 - 163. Phon-Amnuaisuk, S., Law, E.H.H. and Kuan, H.C.: Evolving music generation with som-fitness genetic programming. In: Giacobini, M., Brabazon, A., Cagnoni, S., Caro, G.D., Drechsler, R., Farooq, M., Fink, A., Lutton, E., Machado, P., Minner, S., O'Neill, M., Romero, J., Rothlauf, F., Squillero, G., Takagi, H., Uyar, S., Yang, S. (eds.) Applications of Evolutionary Computing, EvoWorkshops 2007: EvoCoMnet, EvoFIN, EvoIASP, EvoINTERACTION, EvoMUSART, EvoSTOC and EvoTransLog, Valencia, Spain, Proceedings. Lecture Notes in Computer Science, vol. 4448, pp. 557–566. Springer (2007). Accessed from 11–13 Apr 2007
 - 164. Pirnia, A., McCormack, J.: Compressed multidimensional trees for evolutionary music representation. In: Non-Cochlear Sound: Proceedings of the 38th International Computer Music Conference, ICMC 2012, Ljubljana, Slovenia. Michigan Publishing (2012). Accessed from 9–14 Sept 2012
 - 165. Prisco, R.D., Zaccagnino, G., Zaccagnino, R.: Evobasscomposer: a multi-objective genetic algorithm for 4-voice compositions. In: Pelikan, M., Branke, J. (eds.) Genetic and Evolutionary Computation Conference, GECCO 2010, Proceedings, Portland, Oregon, USA, pp. 817–818. ACM (2010). Accessed from 7–11 July 2010
 - 166. Prisco, R.D., Zaccagnino, G., Zaccagnino, R.: EvoComposer: an evolutionary algorithm for 4-voice music compositions. *Evol. Comput.* **28**(3), 489–530 (2020)
 - 167. Rebelo, S., Bicker, J., Machado, P.: Evolutionary experiments in typesetting of letterpress-inspired posters. In: Cardoso, F.A., Machado, P., Veale, T., Cunha, J.M. (eds.) Proceedings of the Eleventh International Conference on Computational Creativity, ICCC 2020, Coimbra, Portugal, pp. 110–113. Association for Computational Creativity (ACC) (2020). Accessed from 7–11 Sept 2020
 - 168. Reddin, J., McDermott, J., O'Neill, M.: Elevated pitch: automated grammatical evolution of short compositions. In: Giacobini, M., Brabazon, A., Cagnoni, S., Caro, G.A.D., Ekárt, A., Esparcia-Alcázar, A., Farooq, M., Fink, A., Machado, P., McCormack, J., O'Neill, M., Neri, F., Preuss, M., Rothlauf, F., Tarantino, E., Yang, S. (eds.) Applications of Evolutionary Computing, EvoWorkshops 2009: EvoCOMNET, EvoENVIRONMENT, EvoFIN, EvoGAMES, EvoHOT, EvoIASP, EvoINTERACTION, EvoMUSART, EvoNUM, EvoSTOC, EvoTRANSLOG, Tübingen, Germany. Proceedings. Lecture Notes in Computer Science, vol. 5484, pp. 579–584. Springer (2009). Accessed from 15–17 Apr 2009

169. Reynolds, C.W.: Evolving textures from high level descriptions: gray with an accent color. In: Chio, C.D., Brabazon, A., Caro, G.A.D., Drechsler, R., Farooq, M., Grahl, J., Greenfield, G., Prins, C., Romero, J., Squillero, G., Tarantino, E., Tettamanzi, A., Urquhart, N., Etaner-Uyar, A.S. (eds.) *Applications of Evolutionary Computation - EvoApplications 2011: Evo-
COMNET, EvoFIN, EvoHOT, EvoMUSART, EvoSTIM, and EvoTRANSLOG, Torino, Italy,
Proceedings, Part II*. Lecture Notes in Computer Science, vol. 6625, pp. 384–393. Springer (2011). Accessed from 27–29 April 2011
170. Rodrigues, A., Costa, E., Cardoso, A., Machado, P., Cruz, T.: Evolving L-systems with musical notes. In: Johnson, C.G., Ciesielski, V., Correia, J., Machado, P. (eds.) *Evolutionary and Biologically Inspired Music, Sound, Art and Design - 5th International Conference, Evo-
MUSART 2016, Porto, Portugal, Proceedings*. Lecture Notes in Computer Science, vol. 9596, pp. 186–201. Springer (2016). March 30–April 1 2016
171. Romero, J., Machado, P. (eds.): *The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music*. Springer, Natural Computing Series (2008)
172. Rooke, S.: Chapter 13 - eons of genetically evolved algorithmic images. In: Bentley, P.J., Corne, D.W. (eds.) *Creative Evolutionary Systems*, The Morgan Kaufmann Series in Artificial Intelligence, pp. 339–365. Morgan Kaufmann (2002)
173. Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., Chen, X.: Improved techniques for training GANs. In: Lee, D.D., Sugiyama, M., von Luxburg, U., Guyon, I., Garnett, R. (eds.) *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, Barcelona, Spain*, pp. 2226–2234 (2016). Accessed from 5–10 Dec 2016
174. Santos, A., Pinto, H., Pereira Jorge, R., Correia, N.: MuSyFI: music synthesis from images. In: Silva Garza, A., Veale, T., Aguilar, W., Pérez y Pérez, R. (eds.) *Proceedings of the 12th International Conference on Computational Creativity*, pp. 103–112. Association for Computational Creativity (ACC) (2021)
175. Saunders, R.: Artificial creative systems and the evolution of language. In: Ventura, D., Gervás, P., Harrell, D.F., Maher, M.L., Pease, A., Wiggins, G.A. (eds.) *Proceedings of the Second International Conference on Computational Creativity, ICCC 2011, Mexico City, Mexico*, pp. 36–41 (2011). Accessed from 27–29 Apr 2011
176. Saunders, R., Gero, J.S.: The digital clockwork muse: a computational model of aesthetic evolution. In: *Proceedings of the Convention for Artificial Intelligence and Simulation for Behaviour (AISB) 2001 - Symposium on AI and Creativity in Arts and Science* (2001)
177. Schnier, T., Gero, J.S.: Learning genetic representations as alternative to hand-coded shape grammars. In: Gero, J.S., Sudweeks, F. (eds.) *Artificial Intelligence in Design '96*, pp. 39–57. Springer, Netherlands (1996)
178. Schrum, J., Volz, V., Risi, S.: CPPN2GAN: combining compositional pattern producing networks and gans for large-scale pattern generation. In: Coello, C.A.C. (ed.) *GECCO '20: Genetic and Evolutionary Computation Conference, Cancún Mexico*, pp. 139–147. ACM (2020). Accessed from 8–12 July 2020
179. Scirea, M., Barros, G.A.B., Shaker, N., Togelius, J.: SMUG: scientific music generator. In: Toivonen, H., Colton, S., Cook, M., Ventura, D. (eds.) *Proceedings of the Sixth International Conference on Computational Creativity, ICCC 2015, Park City, Utah, USA*, pp. 204–211 (2015). Accessed from June 29–July 2 2015
180. Scirea, M., Togelius, J., Eklund, P., Risi, S.: MetaCompose: a compositional evolutionary music composer. In: Johnson, C.G., Ciesielski, V., Correia, J., Machado, P. (eds.) *Evolutionary and Biologically Inspired Music, Sound, Art and Design - 5th International Conference, Evo-
MUSART 2016, Porto, Portugal, Proceedings*. Lecture Notes in Computer Science, vol. 9596, pp. 202–217. Springer (2016). March 30–April 1 2016
181. Secretan, J., Beato, N., D Ambrosio, D.B., Rodriguez, A., Campbell, A., Stanley, K.O.: Picbreeder: evolving pictures collaboratively online. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '08*, pp. 1759–1768. Association for Computing Machinery (2008)

182. Secretan, J., Beato, N., D'Ambrosio, D.B., Rodriguez, A., Campbell, A., Folsom-Kovarik, J.T., Stanley, K.O.: Picbreeder: a case study in collaborative evolutionary exploration of design space. *Evol. Comput.* **19**(3), 373–403 (2011)
183. Shaker, N., Nicolau, M., Yannakakis, G.N., Togelius, J., O'Neill, M.: Evolving levels for super mario bros using grammatical evolution. In: 2012 IEEE Conference on Computational Intelligence and Games, CIG 2012, Granada, Spain, pp. 304–311. IEEE (2012). Accessed from 11–14 Sept 2012
184. Shao, J., McDermott, J., O'Neill, M., Brabazon, A.: Jive: a generative, interactive, virtual, evolutionary music system. In: Chio, C.D., Brabazon, A., Caro, G.A.D., Ebner, M., Farooq, M., Fink, A., Grahl, J., Greenfield, G., Machado, P., O'Neill, M., Tarantino, E., Urquhart, N. (eds.) Applications of Evolutionary Computation, EvoApplications 2010: EvoCOMNET, EvoENVIRONMENT, EvoFIN, EvoMUSART, and EvoTRANSLOG, Istanbul, Turkey, Proceedings, Part II. Lecture Notes in Computer Science, vol. 6025, pp. 341–350. Springer, 2010. Accessed from 7–9 Apr 2010
185. Sims, K.: Artificial evolution for computer graphics. In: Thomas, J.J. (ed.) Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1991, Providence, RI, USA, pp. 319–328. ACM (1991). Accessed from 27–30 Apr 1991
186. Sims, K.: Interactive evolution of equations for procedural models. *Vis. Comput.* **9**(8), 466–476 (1993)
187. Smith, J.R.: Designing biomorphs with an interactive genetic algorithm. In: Belew, R.K., Booker, L.B. (eds.) Proceedings of the 4th International Conference on Genetic Algorithms, San Diego, CA, USA, pp. 535–538. Morgan Kaufmann (1991)
188. Spector, L., Alpern, A.: Criticism, culture, and the automatic generation of artworks. In: Hayes-Roth, B., Korf, R.E. (eds.) Proceedings of the 12th National Conference on Artificial Intelligence, Seattle, WA, USA, vol. 1, pp. 3–8. AAAI Press/The MIT Press (1994). Accessed from July 31–August 4 1994
189. Spector, L., Alpern, A.: Induction and recapitulation of deep musical structure. In: Proceedings of International Joint Conference on Artificial Intelligence, IJCAI'95 Workshop on Music and AI (1995)
190. Stanley, K.O.: Compositional pattern producing networks: a novel abstraction of development. *Genetic Program. Evolvable Mach.* **8**(2), 131–162 (2007)
191. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. *Evol. Comput.* **10**(2), 99–127 (2002)
192. Suganuma, M., Shirakawa, S., Nagao, T.: A genetic programming approach to designing convolutional neural network architectures. In: Bosman, P.A.N. (ed.) Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2017, Berlin, Germany, pp. 497–504. ACM (2017). Accessed from 15–19 July 2017
193. Sulyok, C., McPherson, A., and Harte, C.: Corpus-taught evolutionary music composition. In: Andrews, P.S., Caves, L.S.D., Doursat, R., Hickinbotham, S.J., Polack, F.A.C., Stepney, S., Taylor, T., Timmis, J. (eds.) Proceedings of the Thirteenth European Conference Artificial Life, ECAL 2015, York, UK, pp. 587–594. MIT Press (2015). Accessed from 20–24 July 2015
194. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, pp. 2818–2826. IEEE Computer Society (2016). Accessed from 27–30 June 2016
195. Tanjil, F., Ross, B.J.: Deep learning concepts for evolutionary art. In: Ekárt, A., Liapis, A., Pena, M.L.C. (eds.) Computational Intelligence in Music, Sound, Art and Design - 8th International Conference, EvoMUSART 2019, Held as Part of EvoStar 2019, Leipzig, Germany, Proceedings. Lecture Notes in Computer Science, vol. 11453, pp. 1–17. Springer (2019). Accessed from 24–26 April 2019
196. Thywissen, K.: GeNotator: an environment for exploring the application of evolutionary techniques in computer-assisted composition. *Organised Sound* **4**(2), 127–133 (1999)

197. Ting, C., Wu, C., Liu, C.: A novel automatic composition system using evolutionary algorithm and phrase imitation. *IEEE Syst. J.* **11**(3), 1284–1295 (2017)
198. Todd, P.M., Werner, G.M.: *Frankensteinian Methods for Evolutionary Music Composition*, pp. 313–339. MIT Press (1999)
199. Todd, S., Latham, W.: *Evolutionary Art and Computers*. Academic (1992)
200. Togelius, J., Burrow, P., Lucas, S.M.: Multi-population competitive co-evolution of car racing controllers. In: Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2007, Singapore, pp. 4043–4050. IEEE (2007). Accessed from 25–28 Sept 2007
201. Togelius, J., Karakovskiy, S., Koutník, J., Schmidhuber, J.: Super Mario evolution. In: Lanzi, P.L. (ed.) *Proceedings of the 2009 IEEE Symposium on Computational Intelligence and Games, CIG 2009*, Milano, Italy, pp. 156–161. IEEE (2009). Accessed from 7–10 Sept 2009
202. Tomasic, P., Papa, G., Znidarsic, M.: Using a genetic algorithm to produce slogans. *Informatica (Slovenia)* **39**(2) (2015)
203. Toutouh, J., Hemberg, E., O'Reilly, U.M.: Spatial evolutionary generative adversarial networks. In: Auger, A., Stützle, T. (eds.) *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019*, Prague, Czech Republic, pp. 472–480. ACM (2019). Accessed from 13–17 July 2019
204. Toutouh, J., Hemberg, E., O'Neill, U.M.: Analyzing the components of distributed coevolutionary GAN training. In: Bäck, T., Preuss, M., Deutz, A.H., Wang, H., Doerr, C., Emmerich, M.T.M., Trautmann, H. (eds.) *Parallel Problem Solving from Nature - PPSN XVI - 16th International Conference, PPSN 2020*, Leiden, The Netherlands, Proceedings, Part I, Lecture Notes in Computer Science, vol. 12269, pp. 552–566. Springer (2020). Accessed from 5–9 Sept 2020
205. Toutouh, J., O'Reilly, U.: Signal Propagation in a Gradient-based and Evolutionary Learning System. In: Chicano, F., Krawiec, K. (eds.) *GECCO '21: Genetic and Evolutionary Computation Conference*, Lille, France, pp. 377–385. ACM (2021). Accessed from 10–14 July 2021
206. Turhan, C.G., Bilge, H.S.: Variational autoencoded compositional pattern generative adversarial network for handwritten super resolution image generation. In: 2018 3rd International Conference on Computer Science and Engineering (UBMK), pp. 564–568 (2018)
207. Unemi, T.: SBART 2.4: breeding 2D CG images and movies and creating a type of collage. In: Jain, L.C. (ed.) *Third International Conference on Knowledge-Based Intelligent Information Engineering Systems, KES 1999*, Adelaide, South Australia, Proceedings, pages 288–291. IEEE (1999). Accessed from 31 August–1 September 1999
208. Ventrella, J.: Evolving the mandelbrot set to imitate figurative art. In: Hingston, P.F., Barone, L.C., Michalewicz, Z. (eds.) *Design by Evolution: Advances in Evolutionary Design*, pp. 145–167. Springer, Berlin, Heidelberg (2008)
209. Vinhas, A., Assunção, F., Correia, J., Ekárt, A., Machado, P.: Fitness and novelty in evolutionary art. In: Johnson, C.G., Ciesielski, V., Correia, J., Machado, P. (eds.) *Evolutionary and Biologically Inspired Music, Sound, Art and Design - 5th International Conference, Evo-MUSART 2016*, Porto, Portugal, Proceedings. Lecture Notes in Computer Science, vol. 9596, pp. 225–240. Springer (2016). Accessed from March 30–April 1 2016
210. Volz, V., Schrum, J., Liu, J., Lucas, S.M., Smith, A., Risi, S.: Evolving mario levels in the latent space of a deep convolutional generative adversarial network. In: Aguirre, H.E., Takadama, K. (eds.) *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2018*, Kyoto, Japan, pp. 221–228. ACM (2018). Accessed from 15–19 July 2018
211. Wang, C., Xu, C., Yao, X., Tao, D.: Evolutionary generative adversarial networks. *IEEE Trans. Evol. Comput.* **23**(6), 921–934 (2019)
212. Webster, R., Rabin, J., Simon, L., Jurie, F.: Detecting overfitting of deep generative networks via latent recovery. In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019*, Long Beach, CA, USA, pp. 11273–11282. Computer Vision Foundation/IEEE (2019). Accessed from 16–20 June 2019
213. Winters, T., Delobelle, P.: Survival of the wittiest: evolving satire with language models. In: de Silva Garza, A.G., Veale, T., Aguilar, W., y Pérez, R.P. (eds.) *Proceedings of the Twelfth*

- International Conference on Computational Creativity, México City, México (Virtual), pp. 82–86. Association for Computational Creativity (ACC) (2021). Accessed from 14–18 Sept 2021
- 214. L. World: Aesthetic selection: the evolutionary art of Steven Rooke. *IEEE Comput. Graph. Appl.* **16**(1), 4 (1996)
 - 215. Zaltron, N., Zurlo, L., Risi, S.: CG-GAN: an interactive evolutionary gan-based approach for facial composite generation. In: The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, pp. 2544–2551. AAAI Press (2020). Accessed from 7–12 Feb 2020
 - 216. Zhang, J., Taarnby, R., Liapis, A., Risi, S.: Draw compile evolve: sparking interactive evolutionary art with human creations. In: Johnson, C.G., Carballal, A., Correia, J. (eds.) Evolutionary and Biologically Inspired Music, Sound, Art and Design - 4th International Conference, EvoMUSART 2015, Copenhagen, Denmark, Proceedings. Lecture Notes in Computer Science, pp. 261–273, vol. 9027. Springer (2015). Accessed from 8–10 Apr 2015
 - 217. Zoric, V., Gamback, B.: the image artist: computer generated art based on musical input. In: Pachet, F., Jordanous, A., León, C. (eds.) Proceedings of the Ninth International Conference on Computational Creativity, ICCC 2018, Salamanca, Spain, pp. 296–303. Association for Computational Creativity (ACC) (2018). Accessed from 25–29 June 2018

Chapter 11

Evolution Through Large Models



**Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh,
and Kenneth O. Stanley**

Abstract This chapter pursues the insight that large language models (LLMs) trained to generate code can vastly improve the effectiveness of mutation operators applied to programs in genetic programming (GP). Because such LLMs benefit from training data that includes sequential changes and modifications, they can approximate likely changes that humans would make. To highlight the breadth of implications of such *evolution through large models* (ELM), in the main experiment ELM combined with MAP-Elites generates hundreds of thousands of functional examples of Python programs that output working ambulating robots in the Sodarace domain, which the original LLM had never seen in pretraining. These examples then help to bootstrap training a new conditional language model that can output the right walker for a particular terrain. The ability to bootstrap new models that can output appropriate artifacts for a given context in a domain where zero training data was previously available carries implications for open-endedness, deep learning, and reinforcement learning. These implications are explored here in depth in the hope of inspiring new directions of research now opened up by ELM.

Kamal Ndousse work done at OpenAI.

J. Lehman (✉) · J. Gordon · S. Jain · C. Yeh · K. O. Stanley
OpenAI Inc., San Francisco, CA, USA
e-mail: joel@openai.com

J. Gordon
e-mail: gordonjo@openai.com

S. Jain
e-mail: jains@openai.com

C. Yeh
e-mail: cathy@openai.com

K. Ndousse
Anthropic Inc., San Francisco, CA, USA

11.1 Introduction

For many in the evolutionary computation (EC) community, the rise of deep learning (DL) has raised questions on its implications for EC. Both approaches scale well with compute and both can yield useful discoveries and meaningful surprises. Yet are they ultimately competing paradigms, or rather are they complementary? In this chapter, we explore the latter possibility, of considerable synergy, by highlighting an untapped implication of large language models (LLMs; [7, 13]) for both genetic programming (GP; [4, 31]) and open-endedness [5, 59, 63].

In particular, in this new Evolution through Large Models (ELM) approach, an LLM trained on code can suggest mutations that are intelligent, thereby facilitating a dramatically more effective mutation operator that sidesteps many of the challenges that previously existed for evolving programs [48]. Interestingly, the benefits of ELM are also reciprocal back to deep learning: the set of samples generated through the LLM can eventually constitute a new training set in a novel domain that can then fine-tune the LLM to perform well in the new domain, a novel data-generation procedure. Furthermore, this approach ultimately opens up new opportunities in the pursuit of open-endedness by increasing the generative capabilities of the LLM solely through its own generated data.

LLMs have recently yielded impressive results in automated code generation [15, 40]. These models bootstrap from human knowledge by learning from very large datasets to achieve general coding competency. The fact that such bootstrapping is possible is clearly relevant to GP. After all, GP is in effect a generative approach to programming. While it might seem at first glance that LLMs therefore might out-compete or subsume GP, in fact GP does still offer an advantage in situations where the particular class of programs targeted by the search is far (or even completely lacking) from the training distribution of the LLM. In such cases, the LLM offers limited recourse (prompt engineering to learn an entirely new domain would be prohibitive), while GP can in principle evolve in any space (though in practice some spaces may be intractable due to the amount of mutation necessary to get consistent signal on fitness).

Interestingly (and perhaps surprisingly), the best of both worlds is easily attainable by combining them: simply by prompting the LLM to generate changes the LLM can serve as a highly sophisticated mutation operator embedded within an overarching evolutionary algorithm. This way, the LLM in concert with evolution can steer each other toward the right region of the solution space even though neither evolution with a conventional mutation operator nor the LLM on its own could generate anything close. In effect, program evolution using LLM-based perturbation begins to bridge the divide between evolutionary algorithms and those that operate on the level of human ideas. That is, LLMs can be trained to approximate how humans intentionally change programs, while staying on the manifold of functionality. Furthermore, such LLMs can be further fine-tuned on successful perturbations for the purposes of self-improvement, culminating in a novel technique for iteratively enhancing the performance of ELM.

To highlight the potential of this approach, in this chapter, an entire dataset in a novel domain is generated from only a single mediocre starting example designed by hand by humans. In particular, the domain is called Sodarace [41, 67], where two-dimensional ambulating robots of arbitrary morphology are constructed for diverse terrains. Sodarace is cheap to simulate, allowing fast iteration, and also makes it easy to appreciate the sophistication of designs intuitively by simply watching the robot walk. In this way, it facilitates quick assessment of whether a design is successful both quantitatively and qualitatively.

To make the contribution of ELM explicit in the experiments in this chapter, the Sodaracers are encoded as raw Python programs that output an enumeration of the ambulating robots' components. That way, it is possible to demonstrate that ELM is a form of GP that can operate on a modern programming language directly, with no special provisions needed beyond the generic (i.e., *not* previously trained in Sodarace) existing code-generating LLM.

A final important insight unlocked by this approach is that the ability to generate diverse solutions in a domain or part of the search space where there was little to no training data is foundational to bootstrapping an open-ended process [5, 59, 61]. After all, open-endedness is fundamentally about searching outside the distribution of previous experience, which is exactly what ELM helps the LLM to do. Because this novel capability has potentially far-reaching implications, we have chosen in this work to focus on the implications of the generated data that can be produced by ELM. Of course, ELM is applicable in many other contexts that will undoubtedly be explored in the future.

More specifically, experiments that follow show that generated data is sufficiently rich that it can serve as training data for fine-tuning LLMs to generate code for viable new Sodaracers consistently, and furthermore that reinforcement learning (RL) can even fine-tune an augmented LLM to output Sodaracers *conditionally*, depending on the terrain. In the future, such conditional invention has the potential to unlock entirely new kinds of open-ended processes, just as humans have open-endedly built civilization over centuries by conditionally inventing its constituents.

In short, the main contributions of this chapter are (1) the ELM method for efficiently evolving programs through LLMs, (2) a technique for improving ELM's ability to search over time by fine-tuning its LLM-based mutation operator, (3) a demonstration of ELM in a domain not included in the training data of the LLM, and (4) validation that data generated through ELM can bootstrap enhanced LLMs that offer a novel path toward open-endedness.

11.2 Background

This section reviews previous work in genetic programming, large language models, and open-endedness.

11.2.1 Genetic Programming

The field of genetic programming (GP) applies evolutionary algorithms to evolve computer programs to solve problems [4, 31, 34]. The promise of GP is that computer code is a computationally universal representation that underlies much modern technology, including artificial intelligence itself. Therefore, it is conceivable for GP to automatically evolve programs that achieve human-level (or beyond) performance across diverse application domains [32]. However, there are obstacles in practice to its successful and widespread application to challenging problems.

One obstacle is that scaling GP to evolve increasingly complicated programs can be challenging [48], and that effectively applying GP to a new domain can require significant domain expertise. A researcher often must explicitly specify what functions, variables, and control structures are available to evolution [10, 31], which limits what ultimately can be evolved. In contrast, a human programmer can open-endedly decide what libraries to import and how to write many interdependent subroutines or classes. Research aims to lift these constraints, often through enabling modular reuse of code, e.g., through automatically defined functions [31], data-mining populations to find common sub-components [29], or attempts to use solutions to previous problems when solving new ones [56]. However, no method yet enables GP to scalably operate on human-designed programming languages with a minimum of domain-specific tweaking.

A second obstacle is that nearly all GP methods explore through random perturbations of code, unlike humans, who through active practice improve their proficiency in making deliberate, complex, and coupled modifications to programs [25, 38]. Unlike perturbing, e.g., neural network weights, wherein continuous parameters subject to small enough perturbations can predictably generate small changes in functionality [35, 53], perturbing code requires discrete changes that often dramatically shift functionality [23], thereby complicating search. While there exist approaches toward more directed generation of offspring (e.g., building probabilistic models of high-performing programs [52], evolving reproduction operators [58], or applying less-impactful mutation operators [23]), the problem remains at core unsolved.

In contrast to GP, humans learn to reason about code in its full complexity through experimentation and learning. This iterative effort leaves a permanent signature in repositories of code, such as GitHub. The next section describes progress in training large language models upon such repositories as a potential way to bypass the above obstacles.

11.2.2 Large Language Models

Large language models (LLMs; [7, 13, 21]), trained on Internet-scale data, have progressed at an impressive pace in recent years. The main idea (in auto-regressive models such as GPT-3 [13] and GPT-4 [46]) is to train increasingly large neural

networks (built on the popular transformer architecture [70], sometimes with billions of parameters) on the seemingly simple task of next-token prediction (i.e., given a sequence of tokens seen so far, predict the proceeding token). Scaling such LLMs (and formulating problems of interest as natural language processing tasks) has resulted in groundbreaking performance across a wide range of tasks [13, 16, 46], including program synthesis [15, 28, 39, 40].

In particular, by training LLMs on large-scale coding data, e.g., from GitHub, it is possible to produce models with impressive function-synthesis capabilities [15, 39, 40], highlighting the possibility to bootstrap the ability to fluently code from large-scale data. Further developments are *diff* models that are trained on diffs from GitHub [9, 51]. A diff is an incremental change to a file that is committed to a version control system such as GitHub, accompanied by a *commit message* describing the intent of the change. In this way, diff models are trained how, given a piece of code and any potential commit message, to suggest an informed *change*. Through the lens of evolutionary algorithms, such diff models can be viewed as intelligent perturbation operators, providing a way to walk over the manifold of code (in a controllable way) through mimicking human programmers. An interesting further possibility is that such models are amenable to further training through gradient descent, implying a potentially powerful mechanism for self-adaptation (e.g., through reinforcing successful diffs during evolution). Both diff models and their capacity for self-adaptation are explored in this work as a way to improve GP. However, it is also important to note that general language models not trained directly on diffs can also act in effect like diff models when given the right kinds of prompts (see Sect. 11.3.1).

11.2.3 Open-Endedness

With origins in the open-ended evolution community [5, 59, 68, 69] within artificial life, the field of open-endedness seeks to create algorithmic systems that produce never-ending innovation [63]. Given the primacy of search with ML, research within open-endedness naturally has focused on refining algorithms for open-ended search, such as those driven by novelty [37, 44] or curiosity [49, 64]. While such focus has indeed lead to algorithmic progress, there is a growing awareness of the criticality of the *environment* in which open-ended algorithms are applied [22, 24, 57, 84].

That is, the environment limits what can arise within the system and for how long its products can remain interesting. As a result, some have argued for more complex environments for open-endedness, such as video games [22, 24], and others have argued that features of the environment should co-evolve with agents [20, 84]. Yet a theory for what specific forms of additional such complexity is needed for enduring open-endedness has been lacking. This chapter contributes a possible theory, arguing that agents outputting inventions into the environment in response to previous inventions may be a principled route to such continuing open-endedness.

One challenge in evolving aspects of the environment (such as inventions) is how they are encoded. Most research applies encodings that are specifically fit to describe some fixed part of a larger environment, e.g., a fixed way of describing edges within a maze [11] or the shape of a 2-D landscape [84]. While sometimes the *encodings* of these parts are universal (e.g., the CPPN encoding of landscapes in [84] can describe any landscape, and the RNN encoding of [20] can describe any maze), it is unclear how to expand the representation to include more of the environment without relying upon ad hoc principles. This chapter argues that computer programs are a general and powerful encoding for continually expanding the richness of an existing environment.

11.3 Approach: Evolution Through Large Models

Three distinct components facilitate ELM. First is the novel mutation operator driven by an LLM. Second is an evolutionary outer loop that calls this mutation operator. Finally, the third component is a method for updating the LLM to improve based on its preceding performance. Each of these is detailed in this section.

11.3.1 *Mutation Through Diff*

The main idea behind ELM centers on rethinking the mutation operator for code by exploiting the capabilities of LLMs. In conventional GP, the language of the code and the types of changes allowed through mutation are both chosen intentionally to yield a reasonable chance that perturbations can lead to useful functional changes [31]. In contrast, LLMs unlock an entirely different basis for mutation: it would be more ideal if the mutation operator *understood* the code and how it can be changed in interesting ways, more like a human than a stochastic event.

LLMs can indeed be trained to output code in an auto-regressive manner by exposing them to extensive programming examples [15, 40]. A *diff model* [51] can similarly be auto-regressively trained on a collection of code diffs (e.g., from GitHub). Each diff targets a single file, where the file and diff are short enough to fit into the context of the LLM. The model is trained to predict the diff (formatted, for example, in unified diff format [1]) from the concatenation of the file and the commit message, where the loss includes only the tokens that make up the diff, thereby encouraging the model to predict the diff but not to memorize the file and commit message. In other words, the model learns to predict plausible changes to code from examples of changes made to code by human programmers. It is important to note that the idea of diff models (or their initial training) [51] is not a contribution of this chapter, but diff models are rather a tool applied here in a new context (to produce mutations).

To achieve meaningful mutations, ELM can choose among a set of *commit messages*, which convey to the LLM the details of the operation it should perform in

lieu of mutation. These messages offer significant power and nuance for calibrating mutation operators that is likely highly novel to anyone familiar with implementing mutation in GP or evolutionary algorithms in general. In the experiment in this chapter, the three commit messages and their respective probabilities of being chosen are

- Changed make_walker function (40% chance).
- Changed parameters in make_walker function (30% chance).
- Small change to make_walker function (30% chance).

Of course, any commit message is conceivable. The LLM’s ability to interpret general natural language means that the scope for research exploration (and domain-specificity) here is vast.

As a simple experiment to highlight diff models’ ability to intelligently modify code, an implementation of a function with an adjustable amount of bugs is perturbed with either a simple GP mutation operator or with a 300 M-parameter diff model. The hypothesis is that an intelligent perturbation operator will be better able to make multiple correlated changes to code (in this case to correct the bugs). The 4-Parity task (which is inspired by a standard GP benchmark [31]) serves as a representative test bed. Note that a correct implementation of 4-Parity returns the sum of the four input bits, modulo two. Up to five bugs are introduced to 4-Parity, first by incrementally misnaming each of the variables in the sum calculation, and for the fifth bug, the modulo is changed from two to three. Then, perturbation operators are tested for their ability to (in one perturbation step) change the buggy version of the code to one that successfully passes unit tests. Results in figure Fig. 11.1 highlight how with increasing bugs GP mutation becomes exponentially more unlikely to produce a successful solution (note that *no* mutation from GP solves all five, given 100,000 trials). In contrast, the diff operator is able to fix all five bugs, and its performance is impacted more by the number of *different* types of bugs (i.e., the fifth bug affects the modulo calculation rather than renaming variables) than by the raw number of bugs itself. Further details (including a supporting experiment with another task with similar results) are given in Appendix A of [36], p. 42.

Because the tools involved in an ELM implementation are unconventional, we finally wanted to highlight here several alternatives for implementing such systems in practice today. One option is to use models available on the OpenAI API that can edit through following instructions [2, 47]. A second option is to create an intelligent mutation operator through few-shot prompting instead of through explicit training (as in the diff model). That is, one could design prompts for a model trained on code (like Codex [15] or GPT-6-J [82]). To show the potential to replicate (or improve upon) the results in this chapter, we conducted a simple experiment comparing (on the 4-Parity problem) prompt engineering and edit mode to the diff model. Figure 11.2 shows how models from the API outperform the diff model used in the chapter. Further experimental details can be found in Appendix A of [36], p. 42.

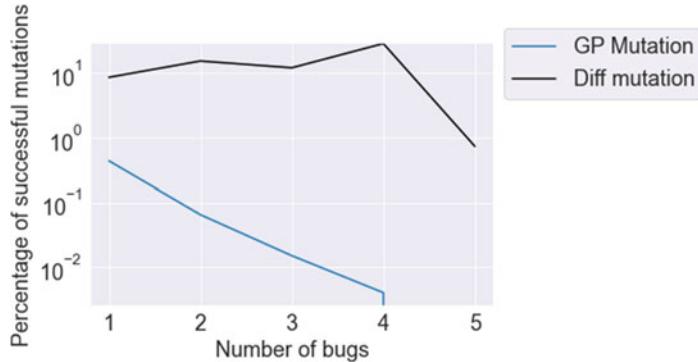


Fig. 11.1 Comparing diff mutation to GP mutation in fixing 4-Parity bugs. The figure shows how the ability of a single mutation to produce correct solutions changes as bugs are incrementally added to a working 4-Parity implementation. Note that success percentage is shown in *log scale*, i.e., success for GP mutation decreases exponentially in the number of mutations (and produces no solutions when there are five bugs). In contrast, diff mutation degrades only with the fifth bug. The conclusion is that LLM-based mutation can indeed make multiple sensible coupled changes to code

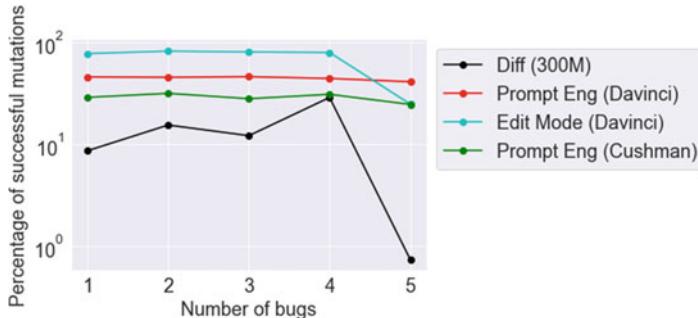


Fig. 11.2 Comparing alternate LLM-based mutations in fixing 4-Parity bugs. The performance of different mutation operators in fixing bugs is shown as bugs are incrementally added to a working 4-Parity implementation. Both edit mode and prompt-engineering approaches outperform the diff model applied in this chapter's experiments. The conclusion is that both prompt engineering on LLMs trained on code and using edit mode models from the OpenAI API are viable options to build upon the work in this chapter

11.3.2 The Evolutionary Algorithm and Implications for Open-Endedness

Because the mutation operator is effectively a modular component for many evolutionary algorithms [19, 43], ELM can be implemented within a diversity of contexts. Of course, the approach is most applicable to a case where the genetic encoding is through a known programming language, because that is how the benefits of the LLM will be realized. Genetic encodings in natural language or any other language

at which LLMs excel are also conceivable, but of course the utility of such encodings would depend on how they are applied and their mapping to a potentially useful phenotype. The experiments in this chapter focus on Python 3 genotypes, which are also by their nature variable in length. The ability to use modern programming languages as genotypes without the need for any special accommodation is a key benefit of ELM.

While there are many options for the evolutionary algorithm in the outer loop, we have chosen in this chapter to implement ELM within a quality diversity (QD) algorithm [45, 50]. An important motivation for this choice is that the emergence of the ability to search intelligently for arbitrarily complex programs is tantalizingly close to overcoming some of the key obstacles to open-endedness [61], and ELM is an opportunity to highlight this opportunity.

Recall that we do not yet know how to make an algorithm that exhibits genuinely open-ended divergence. While there has been progress toward open-endedness in recent years, the state of the art remains *weak open-endedness*, wherein novel and interesting discovery continues only for a brief time, eventually ending in a plateau as the possibilities are exhausted [3, 11, 12, 63, 83, 84]. In contrast, in *strong open-endedness*, the process would never plateau—if we left and returned a year later, or even a *million* years later, its products would continue to become more interesting over time. No algorithm comes close to such an achievement, though it is evidently possible in nature.

The question then is what stands between today’s algorithms and tractable strong open-endedness. This gap remains despite that recent work in open-endedness appears to make progress. For example, the Enhanced POET algorithm continues to generate diverse and increasingly complex terrains for bipedal robots to solve [84]. In their hide-and-seek experiment, [3] show agents discovering increasingly complex strategies like assembling blocks into a hideout. Yet despite such algorithms clearly demonstrating the capability to continue to invent new solutions, all such demonstrations share a singular downfall: they slow down and eventually end. Formalizing ELM within a QD framework in effect offers a novel opportunity to address this challenge.

This opportunity connects to the difficulty of formulating an artificial environment that imposes no limit on what even the most capable open-ended algorithm can achieve, as noted in the Background. The challenge of devising artificial environments with unbounded potential raises the intriguing question of what property our universe and planet possess that is lacking in current artificial environments. This question is critically important for open-endedness because in the absence of that property, open-ended algorithms cannot demonstrate their full potential. If the problem indeed stems from the fact that artificial environments to date offer only finite possible experiences until their potential is exhausted, then to overcome this bottleneck the environment itself needs to possess the potential to change forever.

Since the emergence of intelligence in nature, much environmental change has been driven by the intelligent agents themselves. Eventually, humans acquired the ability to leave *detached* artifacts in the environment that radically alter its potential for themselves and other agents, like a house, a vehicle, or even a *program*. Unlike

new organisms that are evolved over generations, such *detached conditional things* (DCTs) are generated intentionally as a condition of the observations of the agent. Once DCTs enter the world, open-endedness accelerates because the environment is rapidly updating even within the course of a single lifetime.

Each DCT creates an opportunity for further DCTs. For example, the invention of the door creates the opportunity for keys to be invented, which then sets the stage for lock picks, and so on. And because they are detached, DCTs can leave a permanent legacy in the environment well beyond the lifetime of their inventor. In this way, invention in the era of DCTs is open-ended, and accordingly has continued for thousands of years, from fire and wheels to space stations and computers.

This theory of DCTs supplies an abstract answer to the problem of a limited environment: Agents must be able to imprint the environment with DCTs in response to those already present within it. However, realizing DCTs in practice requires addressing a separate question: how can agents be enabled to efficiently invent DCTs of limitless complexity in a new domain?

Interestingly, computer programs are universal representations, meaning that the procedure of assembling new artifacts can naturally be described algorithmically. For example, programmers have leveraged code to help create enormously complex artifacts (like the layouts of computer chips or instructions for 3-D printers to produce complex physical objects). Of course, programs themselves can function as DCTs. In this way, a procedure that can search through modern program space and ultimately generate such programs conditionally is a candidate for creating open-ended environments of unlimited capacity. The experiment in this chapter will demonstrate in more detail how ELM makes such a construct conceivable; the significance of QD is that its ability to generate a diverse space of artifacts can serve as the bootstrap to obtaining agents capable of generating DCTs. In short, the QD algorithm is generating *training data* that can transform the LLM into a kind of DCT generator.

While any QD algorithm can work with ELM, the specific algorithm in the experiment in this chapter is MAP-Elites [18, 45] (Fig. 11.3). The core of MAP-Elites is a uniformly spaced grid of niches (called the *map*) that spans user-specified dimensions of solution diversity, called the *behavior characterization*. Upon initialization, a single pre-existing (hand-designed in this chapter) solution is evaluated and placed into the map. In each iteration thereafter, an inhabited niche is randomly chosen and the solution within that niche is perturbed by the diff model and evaluated. The new candidate solution is assigned its niche from its behavior characterization, and if that niche is unfilled or the new solution outperforms the niche's current inhabitant, it becomes the champion of that niche; otherwise, the candidate is discarded. In this way, over iterations of search, the map gradually fills with increasingly diverse and high-quality solutions.

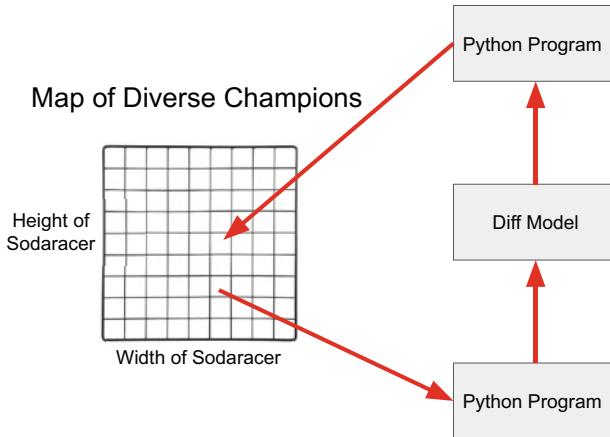


Fig. 11.3 MAP-Elites with ELM. In each iteration, an existing Python solution is sampled from the map of solutions for each independent replica of a diff model. Each replica generates a batch of diffs that are applied to the sampled solution to generate modified candidate solutions. These candidates are evaluated and are then inserted into the map if they either establish a new niche or outperform the niche’s current champion. Over iterations, a single initial seed program gives rise to a diversity of high-performing Python programs

11.3.3 Fine-Tuning the Diff Operator

Interestingly, because the mutation (diff) operator is itself an LLM, it has the potential to be improved with respect to the domain. While self-adaptation [26, 33, 42] has a long tradition in evolutionary computation, including algorithms such as CMA-ES [26] and natural evolution strategies [85], the kinds of improvements possible in ELM are unique by offering the possibility of the LLM learning *how to think about change*. That is, ideas for changes that are most promising in one domain might be different than in another, and the richness of the LLM offers the potential to capture such nuance through experience. In particular, the pretrained diff model can be trained further (which is called *fine-tuning*) with accepted diffs (by MAP-Elites) from initial iterations or runs of ELM. That way, the diff operator updates to understand better the kinds of modifications that lead to either higher quality, more novelty, or both. This fine-tuning technique can cause ELM itself to improve over iterations. Of course, over a long run, the ideal kinds of changes might change; continually fine-tuning based on recent experience can potentially track such drifting opportunities. In this chapter, the potential of fine-tuning is demonstrated through a single fine-tuning iteration, but the investigation of such continual refinement is an open research opportunity. Note that the prompt-engineering approach to LLM mutation described at the end of Sect. 11.3.1 can also benefit from fine-tuning in this way.

11.4 Experiment and Results

The primary motivation for the experiment that follows is to give a taste of the breadth of implications of ELM, to evolutionary computation, to deep learning, and to open-endedness. For this purpose, this experiment focuses on the problem of the *invention* of complex artifacts (which could eventually serve as DCTs in a future more ambitious experiment). While the potential scope of applications for ELM is broad, the opportunity to learn to invent complex artifacts in an arbitrary domain extends directly from the augmented ability to search through programs; seeing this inventive capability realized thereby highlights novel opportunities opening up.

The experiment will aim to bootstrap from a few hand-written (and barely functional) examples of an invention into an LLM-based inventor that can fluidly output appropriate inventions conditioned on its environment. This concept is demonstrated in the domain of *Sodarace* [41, 67], a physics-based invention domain that serves as a cheap-to-simulate microcosm of invention. The goal in Sodarace is to construct from collections of masses and oscillating springs two-dimensional robots that can locomote competently. A wide range of interesting Sodaracer robots are possible, as highlighted by previous ML research [67] and the origins of the domain: Sodarace began as a web application called Sodaconstructor, wherein the human design of Sodaracers was sufficiently compelling for an online community to form around the endeavor [41].

An individual Sodaracer (Fig. 11.4) is composed of a variable-sized collection of point masses (each fully described by its initial 2-D position) and oscillating springs that connect masses together. The motion of the robot is driven by the oscillation of its springs, and each spring has parameters specifying the amplitude and phase of its oscillation (by convention all springs have the same period). To evaluate a particular Sodaracer, it is simulated in a specific terrain for a fixed amount of time

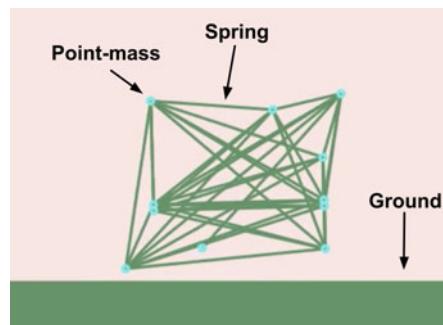


Fig. 11.4 An Example Sodaracer. The objective in the Sodarace domain is to design a Sodaracer that locomotes effectively across the ground terrain. Labeled in the image are examples of a mass and a spring that connects two masses together. A Sodaracer design consists of a variable number of masses and springs, where springs also have oscillatory parameters that determine Sodaracer's motion

and its ability to traverse that terrain is measured (i.e., how far Sodaracer’s center of mass moves along the x-axis); additionally, to measure the diversity of solutions for MAP-Elites, features capturing gross aspects of the robot’s morphology (i.e., its initial height, width, and total mass) are recorded. While a search algorithm could operate directly in the space of masses and springs, here instead LLMs output Python code that describes the morphology of the Sodaracer. For examples of such source code, see Appendices B and G of [36], p. 45 and p. 53, respectively. In this way, the programs evolved by ELM are in effect *indirect encodings* [6, 8, 60, 62] for Sodaracers. That is, in principle, any indirect encoding expressible through code could be evolved from scratch or modified by ELM.

More ambitiously than only generating a repertoire of Sodaracer designs, the experiment will attempt to implement an entire *invention pipeline* that ultimately yields a novel kind of conditional LLM that can input a terrain and output an appropriate Sodaracer for that terrain. ELM thereby serves as the initial *data-generation phase* of this pipeline, showing in this way how ELM can serve in general as a way of generating domain data for downstream deep learning where it did not previously exist. Furthermore, in the future, the ability to train such conditional inventors could serve as a foundation for an open-ended world of DCT-generating agents.

In practice, the aim of the invention pipeline is to create an agent that can output complex artifacts conditionally, based on its observation of the environment. If *invention* is conceived as an action, then learning to invent conditionally can be viewed as a reinforcement learning (RL) problem [66]. That is, for any given observation, the agent can be rewarded depending on the success of the resultant invention. For example, in Sodarace, the agent might observe a specific terrain such as a hill and then output a design for a Sodaracer artifact. The reward then depends upon the performance of the Sodaracer in the observed terrain.

This approach sounds straightforward—it is simply RL with complex outputs—but there is a problem. If the agent has no prior experience in the domain (e.g., in Sodarace), then outputting even a valid (let alone working) artifact is effectively impossible. As a result, there is no gradient for RL and it cannot bootstrap into the new domain.

Therefore, to get RL started, some form of pretraining is necessary. In effect, the RL fine-tuning described above is actually the last step in a *pipeline*, where the preceding step is to teach the agent something preliminary about its domain. For that purpose, an LLM can be trained on a large set of *examples* from the target domain. For example, these examples could be Sodarace walker designs. After exposure to enough such designs, in principle, the LLM knows something about the domain and can output sample artifacts from the training distribution. With such knowledge later passed on to RL, it should now be possible to bootstrap into conditional fine-tuning.

However, there is *still* a problem: where did all the examples come from for training the LLM? If the hope is for the conditional inventor eventually to invent in a novel domain like Sodarace where a generic LLM is unlikely to have any exposure, then the source for all the examples needed to train the LLM is itself elusive. As a consequence, the pipeline needs yet one more preceding step—which is where ELM

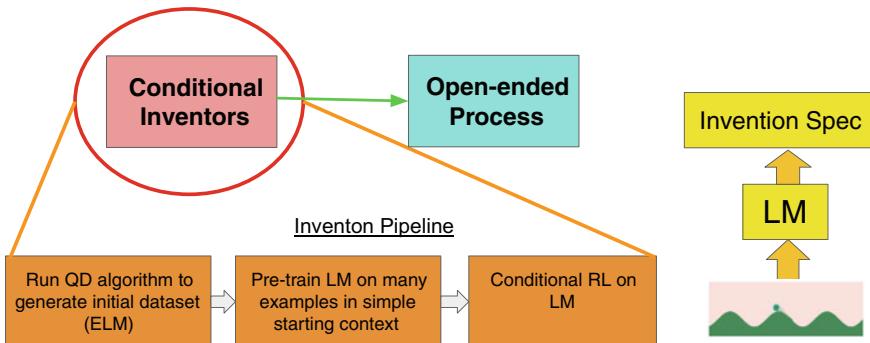


Fig. 11.5 The Invention Pipeline. (left) A three-staged training pipeline bootstraps from a single example of an invention to an LLM that can output an invention tailored to its current condition. The hope for the future is for such a conditional inventor agent to help seed an open-ended process, wherein interactions between agents and their inventions spur continual innovation. (right) In the Sodarace domain, the conditional inventor observes the terrain, which conditions the LLM to output the specification of the desired invention

comes in—to generate a set of example artifacts from scratch, which could then train the LLM that will eventually bootstrap RL.

Generating a diverse and large set of initial training examples is a search problem. However, because no LLM yet has any exposure to the right kind of data, it is a search problem within the invention space rather than within the weight space of neural networks. Searching for diverse functional examples (to get a wide pretraining distribution) within the space of artifacts is the natural role of QD (i.e., MAP-Elites in this chapter). Combined with the diff function, the result is ELM, which yields a novel approach to generating training examples, thereby bootstrapping the entire process.

To recap, what emerges is a three-stage *invention pipeline* for training conditional inventors (Fig. 11.5):

1. **ELM.** Search for a diverse set of example artifacts (e.g., Sodaracers on flat ground).
2. **Pre-train the LLM with examples from ELM.** The LLM accordingly learns to output example inventions from the training distribution.
3. **Learn to invent conditionally.** Splice new conditional inputs onto the LLM and fine-tune it through RL to produce appropriate inventions for the conditions it observes.

11.4.1 Encoding Sodaracers with Python

Previous experiments targeting Sodarace have leveraged specialized evolutionary encodings [67]. Instead, in this work, plaintext Python code acts as a generic representation for inventions. By showing how Python can be used to represent artifacts in an arbitrary domain, it opens up the possibility of using it as a generic encoding in diverse future domains. More specifically, in the experiments, an individual is evaluated by executing its code through the Python interpreter. The product of the interpreter (for a viable individual) is a data structure containing the description of a Sodaracer (i.e., a Python dictionary containing lists of both point masses and springs), which can then be passed to the Sodarace simulator to evaluate the encoded Sodaracer’s behavior. Note that Sodaracers are encoded in Python throughout the invention pipeline, i.e., ELM evolves Python programs and the language models in both latter stages of the pipeline are trained to output Python programs (Figs. 11.6 and 11.7).

Preliminary experiments showed that the diff model’s initial performance (i.e., before fine-tuning) in creating useful perturbations depended upon the design of the “interface” through which Sodaracers were procedurally constructed. That is, while a Sodaracer can be constructed in Python by directly adding elements to a Python dictionary with keys such as “joints” and “muscles,” a more Pythonic interface (which was more effective and is what is used in the experiments) is to create a simple class with two methods: “addJoint” (to add a spring) and “addMuscle” (to add a point mass.) The idea is that such an interface (here encapsulated in a class called “walker_creator”) is closer to the training distribution of Python code (although still no Sodarace examples in this format exist). For example, below is the encoding of a simple hand-designed square Sodaracer (that is also used in the experiments as a seed), as well as its translation after being executed into a dictionary of joints and muscles. The interface also includes logic for ensuring that the Sodaracer will not break the underlying Box2D physics engine, e.g., that each joint is connected only to so many muscles, that the strength of muscles is limited, and that there is a minimum distance between joints. Note that the benefit of evolving a program that produces a data structure rather than directly evolving the data structure itself relates to the benefits of indirect encoding (i.e., a program can leverage regularities through loops, conditionals, and functions, to efficiently encode large complex structures) [62]. Figure 11.8 shows an image of this walker when instantiated.

11.5 Pipeline Stage 1: Data Generation Through ELM

Recall that the aim in this first stage is to generate a large variety of high-quality examples from a single example starter program through ELM. In this stage of the pipeline, the Sodarace environment is a simple flat terrain.

```

1 from walk_creator import walker_creator
2
3 def make_square(wc, x0, y0, x1, y1):
4     """ Make a square with top left x0,y0 and top right x1,y1 """
5     j0 = wc.add_joint(x0, y0)
6     j1 = wc.add_joint(x0, y1)
7     j2 = wc.add_joint(x1, y1)
8     j3 = wc.add_joint(x1, y0)
9
10    return j0, j1, j2, j3
11
12
13 def make_walker():
14     wc = walker_creator()
15
16     # the main body is a square
17     sides = make_square(wc, 0, 0, 10, 10)
18     center = wc.add_joint(5, 5)
19
20     # connect the square with distance muscles
21     for k in range(len(sides)-1):
22         wc.add_muscle(sides[k], sides[k+1])
23         wc.add_muscle(sides[3], sides[0])
24
25     # one prong of the square is a distance muscle
26     wc.add_muscle(sides[3], center)
27
28     # the other prongs from the center of the square are active
29     wc.add_muscle(sides[0], center, False, 5.0, 0.0)
30     wc.add_muscle(sides[1], center, False, 10.0, 0.0)
31     wc.add_muscle(sides[2], center, False, 2.0, 0.0)
32
33     return wc.get_walker()

```

Fig. 11.6 Listing 1: Example Sodaracer-generating program

Recall that ELM in this experiment will evolve through MAP-Elites (Fig. 11.3) [45]. The core of MAP-Elites is a uniformly spaced grid of niches (called the *map*) that spans user-specified dimensions of solution diversity, called the *behavior characterization*. In experiments here, the behavior characterization consists of the height, width, and mass of Sodaracers, and the map is a $12 \times 12 \times 12$ grid into which any behavioral characterization can be mapped. Upon initialization, a single hand-designed solution is evaluated and placed into the map. In each iteration thereafter, an inhabited niche is randomly chosen and the solution within that niche is perturbed by the diff model and evaluated. The new candidate solution is assigned its niche from its behavior characterization, and if that niche is unfilled or the new solution outperforms the niche's current inhabitant, it becomes the champion of that niche;

```

1 {
2     "joints": [(0, 0), (0, 10), (10, 10), (10, 0), (5, 5)],
3     "muscles": [
4         [0, 1, {"type": "distance"}],
5         [1, 2, {"type": "distance"}],
6         [2, 3, {"type": "distance"}],
7         [3, 0, {"type": "distance"}],
8         [3, 4, {"type": "distance"}],
9         [0, 4, {"type": "muscle", "amplitude": 2.12, "phase": 0.0}],
10        [1, 4, {"type": "muscle", "amplitude": 2.12, "phase": 0.0}],
11        [2, 4, {"type": "muscle", "amplitude": 2.12, "phase": 0.0}],
12    ],
13 }

```

Fig. 11.7 Listing 2: Intermediate Sodaracer representation from running the above Python seed program

Fig. 11.8 Instantiation of a hand-designed square

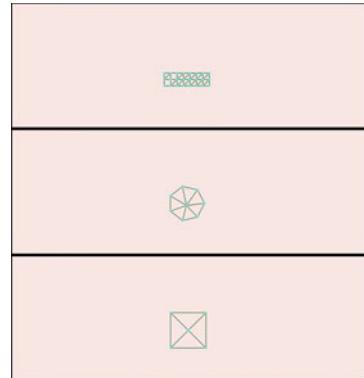
Sodaracer. A video of this walker is available [71]



otherwise, the candidate is discarded. In this way, over iterations of search, the map gradually fills with increasingly diverse and high-quality solutions.

To address the need for seed solutions, four simple seeds were written that explore different architectural motifs: the Square seed, the Radial seed, and two CPPN-like seeds (CPPN stands for *compositional pattern-producing network* [60]); note that source code for these seeds is provided in Appendix B of [36], p. 45. The Square seed instantiates a polygon-like bias, by including a function that creates a square composed of four masses from two coordinates, and code that calls that function and connects the masses together with a for-loop. The Radial seed instead implements a radial bias by replacing the square-generating function with a function that places a given number of masses in a circular shape. Finally, the CPPN-like seeds roughly implement the CPPN-based encoding used by previous work in Sodarace [67], i.e., it places masses and connects springs between them as a mathematical function of their coordinates. The CPPN-based seed's code can be neatly divided into (1) implementing the core functionality of a CPPN and (2) describing a particular instantiation of a CPPN, and thus enables exploring the consequences of letting core functionality of the encoding itself evolve. To this end, there are two CPPN seeds, one in which the CPPN encoding is fixed, called the CPPN-fixed seed, and one where it is mutable, called the CPPN-Mutable seed. Note that these seed programs were not highly tuned as the videos in Fig. 11.9 highlight.

Fig. 11.9 The three seed solutions. From top to bottom: CPPN seed, radial seed, and square seed. Same video as for Fig. 11.8 [71]



11.5.1 Experimental Details and Results

Three independent runs of ELM were conducted with each seed, running for 1,024,000 evaluations each (composed of 2,000 iterations of 512 diffs per iteration). A 300 M-parameter pretrained diff model [51] served as the perturbation operator in these experiments.

One metric of success for ELM is the number of niches filled, which represents the diversity of data generated by ELM, under the hypothesis that such diverse data will benefit later pipeline stages. Figure 11.10 shows that runs of ELM tend to discover a large proportion of niches, highlighting how the system can bootstrap from a single user-provided example to fill the space of desired possibilities. However, the speed of spreading through niches varies across seeds; in particular, introducing loops and/or function composition is required for the Square seed to spread into high-mass niches (e.g., to connect many squares together), which emerges slowly in some runs.

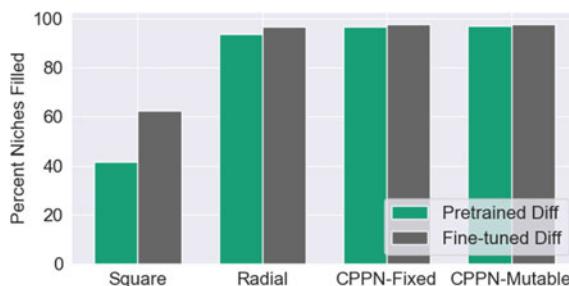


Fig. 11.10 Amount of niches filled across seeds. The figure shows the percentage of all niches (1,728 in total) that are filled by the end of ELM runs across different seeds. Results are averaged across three independent runs for each seed. In general, nearly all seeds fill the map, although the Square seed proceeds more slowly than other seeds

Beyond diversity, the quality of solutions is also important. A gross measure of quality is the maximum fitness discovered by runs, shown in Fig. 11.11. A more nuanced metric that takes both quality and diversity into account is the QD score [50], calculated as the sum of the performance of all champions in the final map. This metric, shown averaged over runs in Fig. 11.12, rewards both quality (having higher scores in each niche) and diversity (having discovered more niches), and thus serves as a succinct measure of ELM’s goal of accumulating diverse, high-quality solutions (and in later stages in the pipeline, of how well an LLM has modeled the distribution of solutions that ELM has uncovered). Attainment of QD differs across seeds, while the CPPN seed uncovers diversity most quickly, the Radial seed generates higher quality solutions on average. The relationship between the seed and the products of search is complex and deserves further future study (see also Appendix D of [36], p. 49 for further analysis of seed robustness).

Fine-tuning the diff model on accepted diffs from an initial series of runs greatly increased performance (Fig. 11.13), while Sodarace-generating programs are out of distribution for the pretrained diff model (applying a Python encoding to this domain is a novel enterprise), fine-tuning effectively aligns the diff model with the domain, an interesting result. Figure 11.13c shows how the fine-tuned diff model produces a significantly higher percentage of diffs that are valid (i.e., able to be applied) and runnable (i.e., the patched program is executable). Because of their higher performance, the outputs of runs applying the fine-tuned diff model are the ones passed to later stages in the pipeline.

Fig. 11.11 Maximum fitness across seeds. The maximum performance attained on average by different seeds is shown. The results suggest that ELM’s capacity to find high-fitness solutions is somewhat robust to seed design

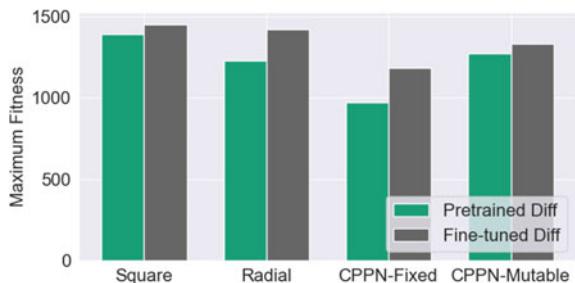


Fig. 11.12 Quality diversity score across seeds. Shown is the average final QD score attained by runs initialized from different seeds. The conclusion is that fine-tuning the diff model has a significant impact on attained QD score, as does the choice of seed

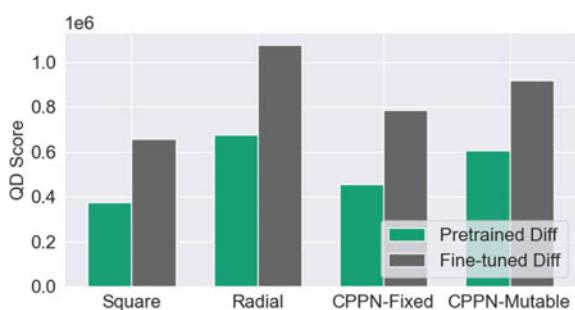
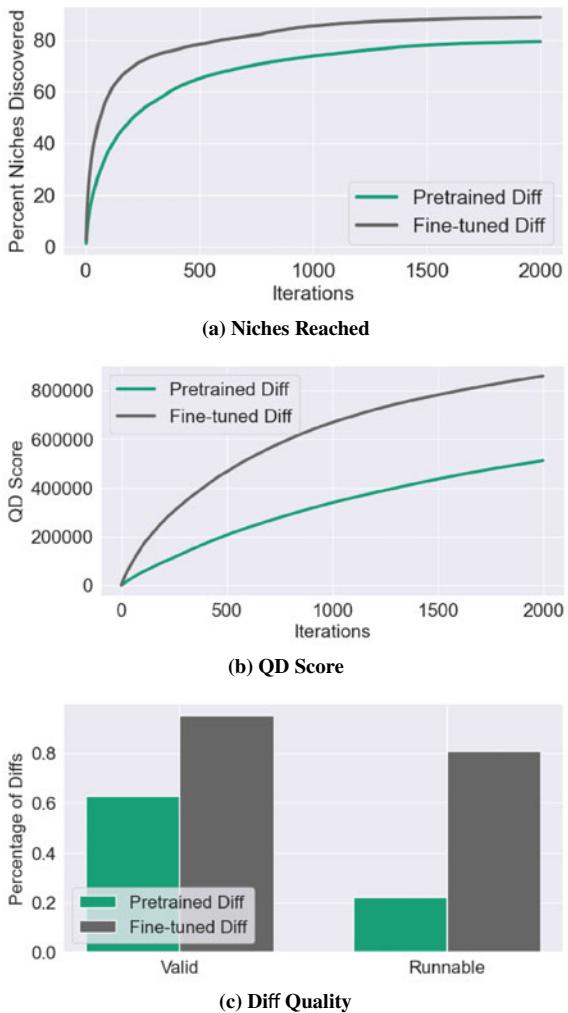


Fig. 11.13 The impact of fine-tuning the diff model on the performance of ELM. For both the pretrained diff model and the fine-tuned one, shown are **a** the number of niches reached, **b** QD score of the produced map, and **c** percentage of valid/runnable diffs proposed. The experiments demonstrate that fine-tuning the diff model improves performance of the evolutionary process across all three metrics



Note that further rounds of fine-tuning are possible (e.g., fine-tuning the diff model again from the improved products of the first round); however, preliminary experiments showed diminishing returns. Future work could explore how to continually improve such models, such as by identifying and encouraging more impactful perturbations instead of including and weighting equally all accepted diffs.

The seeds and fine-tuned diff model also qualitatively impact the kinds of solutions discovered by ELM. While the Radial seed performs well quantitatively (in terms of quality and diversity), it turns out that its products tend to exploit chaotic dynamics that seem overfit to the flat terrain (this hypothesis is tentatively validated in the Stage 3 experiments). The Square and CPPN seeds in contrast are more likely to output inventions that leverage more predictable dynamics. For these reasons, the Radial seed runs were not ultimately used in future stages.

A video selection of the highest quality Sodaracers from these initial runs that showcases the considerable diversity uncovered can be viewed at [72]. An example of a lineage of Sodaracers progressing from the Square seed to a high-quality final Sodaracer can be seen at [73]. In short, ELM shows that by combining an intelligent LLM-based mutation operator with a QD algorithm it is possible to generate hundreds of thousands of working training examples in a completely novel domain where no data was previously available.

11.6 Pipeline Stage 2: Language Model Training

The product of Stage 1 is a collection of programs, whereas Stage 3 RL requires an initial model that can output valid Sodaracer-generating programs. Thus, the second stage of the invention pipeline fine-tunes an LLM on the products of ELM, which serves as the initialization for an RL-based conditional inventor. To do so first requires compiling the results of Stage 1 into a fine-tuning dataset.

While there are many ways to distill a dataset of programs from runs of ELM, a simple thresholded approach is adopted here (although see Appendix E of [36], p. 50 for another simple approach that did not work in practice). The main idea is to append all reasonably capable solutions for each niche.

In more detail, from each run all solutions ever admitted to the map are included, subject to meeting a minimal bar for performance. Some parts of the behavior space offer more stringent challenges (i.e., it is more difficult to locomote when required to be tall but not wide and to have low mass), and yet in some terrains encountered in Stage 3, those kinds of solutions may yet be most effective despite their low level of absolute performance. Thus, for each niche, the maximum performance across all runs is calculated, and the minimal bar for inclusion is set as a percentage of that per-niche score. With a higher percentage threshold, less data is included, but the quality of that data will be higher.

As noted in the previous section, solutions from the Radial seed were qualitatively chaotic. Furthermore, preliminary experiments suggest that such chaotic behavior significantly harms downstream Stage 3 performance. For these reasons, Radial runs of ELM were excluded from the LLM datasets. Datasets for each of the remaining treatments were compiled from nine runs from ELM with the fine-tuned diff model (three runs for each of the Square, CPPN-Fixed, and CPPN-Mutable seeds). In total, the 50% cutoff threshold dataset consisted of 280 K examples, and the 80% cutoff threshold dataset contained a subset of 95 K of those examples.

A variety of pretrained code-generating models were then fine-tuned with these examples (using the standard LLM log-probability loss), leaving out 5% of the data to serve as a test set. Models ranging from 0.1 to 680 M parameters were trained (architectural details for these models can be seen in Appendix C of [36], p. 48). Also, as a control to support the hypothesis that Sodarace models benefit from code-generation pretraining, a 300 M model was also trained instead from a random initialization (signified with “RI” in charts that follow).

Minimum test losses (i.e., loss on generated Sodaracers held-out from the fine-tuning dataset) of the 80% Percentage Threshold models are shown in Fig. 11.14. The 50% Percentage Threshold models exhibit qualitatively similar results across model size (but as both thresholds represent different datasets, loss values are not directly comparable between them). The conclusions are that model sizes above 85 M may not better fit the data, and that random initialization does hurt performance relative to fine-tuning from a model pretrained on code.

However, loss is not the whole story. The interesting question for Stage 2 is whether the LLMs trained from the data generated in Stage 1 can generate the same diversity and quality of data. Therefore, the QD score metric and number of niches discovered (both of which were also reported for Stage 1) are calculated for samples taken from trained LLMs. Because these metrics can be maximized by a model that memorizes the data, and because empirically QD score was more correlated with loss on the training set rather than the test set, the LLM checkpoint for each model is selected on the basis of lowest training loss. In particular, 1,024 samples are taken from each model, which are then evaluated and inserted into a new MAP-Elites map. For comparison, the same metrics are calculated using the Stage 1 dataset, by taking the same number of samples from it and evaluating them in the same way. These results are shown in Fig. 11.15, highlighting that the model samples achieve a similar level of performance as dataset samples, suggesting that they have modeled the data well. Also, there is a slight but consistent QD benefit from models trained on the 80% cutoff dataset, reflecting the higher average QD of that dataset.

A natural further question is how well the model will do when taken out of distribution, i.e., how well has it really internalized the dynamics of Sodarace? That is, the training and test set for fine-tuning are taken from the same runs, and thus the model will likely have encountered all of the motifs in the test set, and so it may not be a representative test of how well the model will generalize in the future. A preliminary test in this spirit is to take the first half of the Python programs describing several inventions from unseen runs, and explore the capacity of different models to generate functional completions. Though the Radial seed usually produced chaotic Sodaracers, in one preliminary run of ELM with the Radial seed, a functional wheel was discovered. As noted previously data from this run (or any other radial runs) was not used to train the models in Stage 2, nor was it used to fine-tune the diff model in Stage 1; thus the ability to complete the wheel can serve as a proxy for generalization. Similarly, two other high-performing individuals were taken from other preliminary

Fig. 11.14 Test loss across model sizes. The minimum test loss achieved by training runs on the 80% Percentage Threshold dataset across model sizes is shown. Model sizes above 85 M may not better fit the data, and random initialization hurts performance

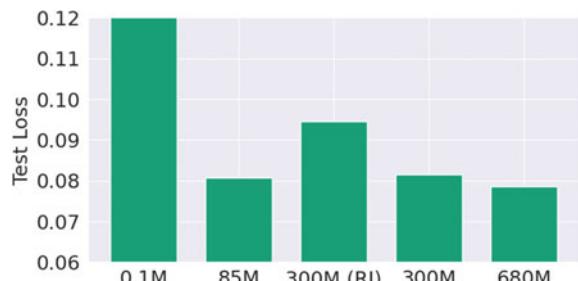
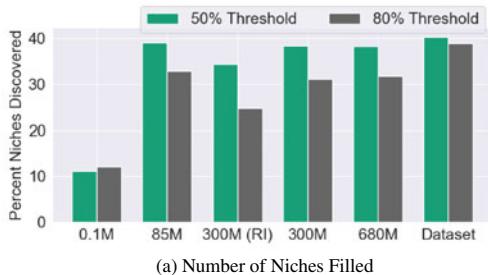
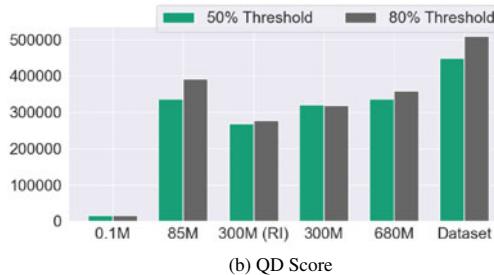


Fig. 11.15 Measuring the quality and diversity of model samples. Two metrics evaluating samples from trained LLMs are shown (across model size and training dataset): **a** the percentage of niches discovered and **b** the QD score achieved. The 80% threshold dataset is on average less diverse but of higher quality than the 50% threshold dataset, and induces the same properties in models trained upon it. There is not a trend in increasing quality or diversity as model size increases beyond 85M, and random initialization hurts performance

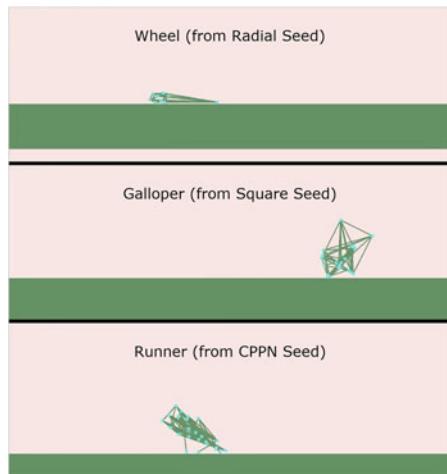
Fig. 11.16 Generalization tests. In this test, the model is asked to complete samples, taken from the first half of the dataset from the unseen runs. These unseen originals are shown in the videos [74]. From top to bottom: Wheel, from radial seed; Galloper, from square seed; Runner, from CPPN seed



(a) Number of Niches Filled



(b) QD Score



runs of the CPPN seed and the Square seed, to create a set of three out-of-distribution completion tests. See Fig. 11.16 for visualizations of these walkers, including videos; source code for these generalization examples can be found in Appendix F of [36], p. 50). Note that further tests of generalization are documented in Appendix H, [36], p. 55.

For each of the three completion tasks, 1,024 completion samples are taken from each model and then evaluated in simulation. In contrast to the in-distribution metrics, in this generalization-focused test, performance was more correlated with the model’s test loss rather than training loss, and thus what checkpoint to evaluate for each model was selected on the basis of lowest test loss. Results are shown in

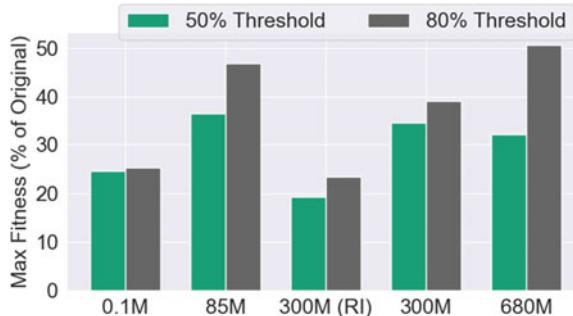


Fig. 11.17 Out-of-distribution completion performance. Shown is the percentage of the original solutions’ performance that is attained by completions from trained LLMs. The percentage shown is the maximum attained over 1,024 independent completion samples from each model. The results are averaged over three out-of-distribution solutions (taken from runs not included in LLM training). The conclusion is that the 80% threshold models perform better than the 50% threshold, and that there is no obvious trend in performance once model size reaches 85 M parameters

Fig. 11.17, highlighting that larger models, and those trained on the 80% threshold, generally perform better at this task. Note that the randomly initialized (RI) 300 M model significantly underperforms, providing more evidence that pretraining on code provides a valuable prior.

Videos of the best performing sample for the Wheel completion from each model are at [75] (for the 80% threshold dataset; the random-initialized 300 M model is not shown because it generated no valid samples for this completion). For the Galloper and Runner completions, the structure and/or behavior of completions often does not match the original sample (especially for the Galloper). In the video [76], a higher performing completion is shown for both of the Galloper and the Runner.

Overall, these results show that an LLM can effectively integrate synthetic data generated through ELM in a novel domain.

11.7 Pipeline Stage 3: Conditional RL

In the final stage, reinforcement learning (RL) is invoked to fine-tune the pretrained LLM output by Stage 2 of the pipeline. The goal is to produce a model that outputs Python programs representing Sodaracers in response to *particular terrains*. Importantly, the output of Stage 2 is an *unconditional* model, in the sense that it samples Sodaracers from a distribution defined by the output of Stage 1, without considering the terrain in which the samples will be deployed. The first step in Stage 3 is thus to convert the model to a *conditional* one, i.e., a model that accepts terrains as inputs, and produces samples of Sodaracers in response.

To achieve this functional form, we first introduce the notion of a *terrain embedding network* (TEN). The role of the TEN is to map a representation of the terrain to a representation that can be used by the model to sample conditionally. In particular, the output of TENs is a vector (or sequence of vectors) in d , the dimension in which the model embeds tokens. That way, the output of the TEN can be treated as the activation from a given prefix, and the model can proceed in effect now sampling conditioned on the output of the TEN.

Concretely, an unconditional auto-regressive LLM defines a sampling distribution over a sequence of tokens $\mathbf{x} = (x_1, \dots, x_n)$ as $p_{\theta}(\mathbf{x}) = \prod_{i=1}^n p_{\theta}(x_i | x_{<i})$. In this stage, we introduce the additional module f_{TEN} , which represents terrains t in \mathbb{R}^d . As $f_{\text{TEN}}(t) \in \mathbb{R}^d$, we can consider the resulting conditional model without further modification:

$$p_{\theta}(\mathbf{x}|t) = \prod_{i=1}^n p_{\theta}(x_i | x_{<i}, f_{\text{TEN}}(t)). \quad (11.1)$$

This approach is similar to the controllable transformer proposed by [30], but with the conditional codes being the output of a TEN, rather than particular tokens from the existing vocabulary.

Given a distribution over terrains $p(t)$, an RL setting is constructed to train the parameters of the TEN and further fine-tune the LLM parameters to the conditional setting. In particular, an episode now consists of sampling $t \sim p(t)$, and sampling a program from the conditional distribution defined in (11.1). The program is converted to a Sodaracer, evaluated in simulation with the terrain t , and the reward is defined as the absolute distance traversed by the Sodaracer in a given period of time.

11.7.1 Terrain Distributions

In this experiment, the distribution over terrains that the model is exposed to is chosen to explore the viability of producing conditional inventors with the Invention Pipeline. The future vision is to lay the groundwork for the ability to deploy agents capable of conditional invention in rich, potentially multi-agent environments that support the development of open-ended processes. In such settings, it stands to reason that learning to output complex artifacts conditioned on observations of the environment would be a prerequisite to ongoing open-ended innovation.

However, in preliminary experiments in the Sodarace domain, learning tended to “gravitate” toward collapsed solutions, wherein a single program is produced that achieves reasonable performance on a subset of the terrains in the distribution support. To reduce the viability of such an outcome and simulate a scenario where conditionality is essential, a small and discrete set of terrains for which a single program *cannot* achieve good performance provides a test where conditional solutions should be significantly more advantageous.

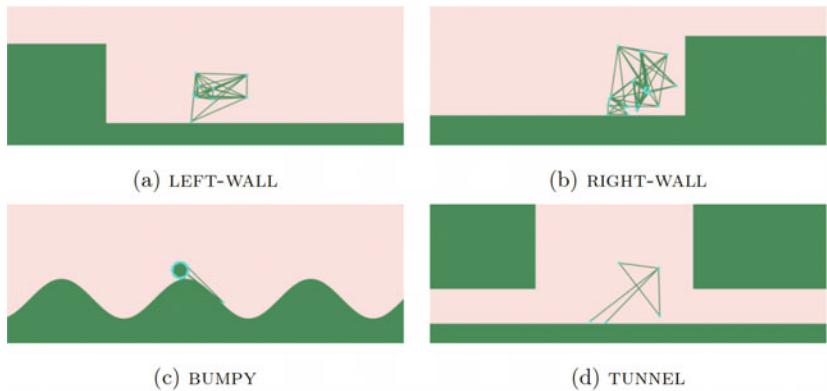


Fig. 11.18 Terrains used in experiments. A small set of terrains from which distributions that force *conditionality* can be constructed. The terrains are **a** LEFT- WALL, **b** RIGHT- WALL, **c** BUMPY, and **d** TUNNEL. The Sodaracers produced by the models are incapable of scaling the walls in LEFT- WALL and RIGHT- WALL, and therefore must produce different Sodaracers for these two terrains. Similarly, achieving good performance in the TUNNEL terrain can only be achieved with short Sodaracers, which struggle to locomote as quickly as taller ones, encouraging the model to distinguish between these terrains. Finally, Sodaracers that are proficient in locomotion on flat terrains tend to perform poorly on BUMPY, encouraging the model to produce yet another Sodaracer for this terrain. In contrast to TUNNEL, which requires Sodaracers with a particular morphology, achieving good performance on BUMPY requires modifying the way Sodaracers *locomote*. Example Sodaracers are added to the figures to illustrate the scale of the terrains

In the experiments, uniform distributions are considered over sets of terrains as illustrated in Fig. 11.18. Two subsets are considered, both of which contain LEFT- WALL and RIGHT- WALL. One set additionally contains TUNNEL, and the other includes BUMPY. These sets were specifically chosen such that the models are incapable of producing a single Sodaracer that achieves good performance on all terrains; to maximize the learning objective, the model must leverage the TEN to incorporate conditionality.

11.7.2 Parametrizing TENs

Two parametrizations for the TEN are explored.

11.7.2.1 Discrete Codes

The terrain distribution has a discrete and finite support. As such, a simple parametrization wherein the terrains are treated as additional tokens in the existing vocabulary, and the embedding for each terrain is learned separately may be

used. The advantage of such a parametrization is that it introduces a relatively small number of new parameters to be optimized with RL, and it is conceptually simple to understand and debug. However, the main disadvantages of such a parameterization are that

- the number of parameters scales with the size of the terrain set and
- it does not allow the model to naturally generalize to unseen terrains at test time, which may be an important constraint for downstream open-ended processes.

11.7.2.2 ResNets

An alternative parametrization is visual representations of the terrains, which can then be processed by visual recognition models. In particular, a ResNet50 [27] embeds images into \mathbb{R}^d as a TEN when experimenting with visual representations of terrains. The main advantages of this parametrization are that it is quite general, could conceivably be used in multiple settings (e.g., teaching a code-generating LLM to write programs in response to visual input), and in theory can generalize to unseen terrains. The main drawback of this approach is that it introduces a large number of new parameters that must be optimized using a sparse RL signal. Conversely, for large terrain distributions, this approach makes it possible to amortize the number of additional parameters necessary for designing conditional inventors.

11.7.3 Experimental Details and Results

Each RL episode consists of sampling a batch of terrains from the distribution, producing samples from the conditional LLM, and evaluating them in simulation to produce the reward.

Proximal policy optimization [55] is the RL algorithm, in conjunction with generalized advantage estimation [54], with default hyperparameters. In preliminary experiments, we found it important to add a KL term (between the policy network and the pretrained LLM from Stage 2) *to the reward function*, as proposed by [17, 65]. The value network is parametrized as a scalar-function version of the policy network, i.e., a separate LLM with a separate prepended TEN initialized from the Stage 2 models. Figure 11.19 illustrates the architectures and pipelines for the policy and value-function networks. Each iteration consists of batches of 1,024 samples (distributed over 32 GPUs), and training runs consist of 100 iterations.

RL is run on pretrained, 300M-parameter LLMs trained with datasets having cutoff thresholds in $\{50\%, 80\%\}$. Recall that we use the cutoff threshold to control the trade-off between data quality and quantity, such that higher thresholds result in smaller pretraining datasets with a higher density of quality instances. For each dataset and terrain distribution combination, three runs are performed using different seeds, and the performance is averaged over samples from the resulting model

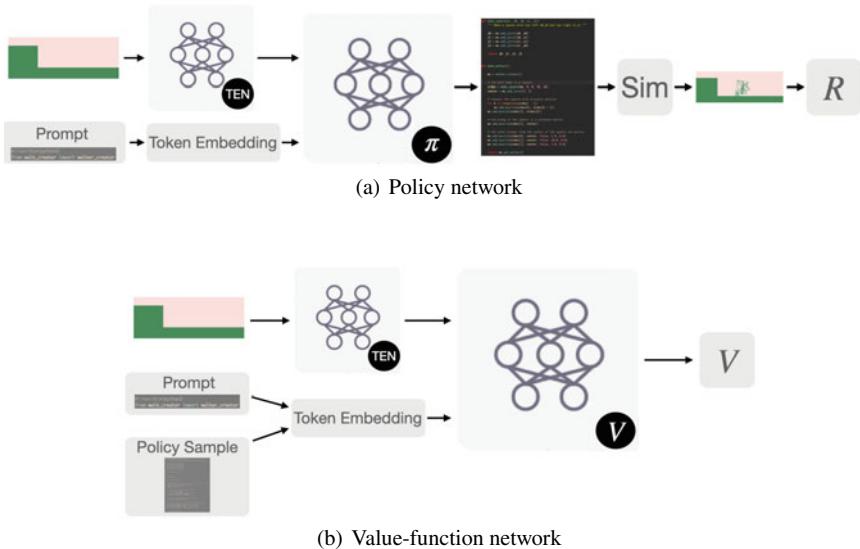


Fig. 11.19 Illustration of the RL architecture. The conditional policy (a) and value function (b) are depicted, both augmented with a separate TEN for terrain embeddings. The policy is conditioned on a particular terrain (via the TEN) and prompt, and produces a sample, which is interpreted as Python code. The code is then used to compile a Sodaracer, which is evaluated in a simulation to produce a reward R . The value function is conditioned on the same terrain (via its own TEN) and prompt, and outputs an estimation of the value (V) of every token output by the policy sample. During learning, the value function is trained to predict advantages estimated using GAE [54]

for each terrain, from over all runs, though we exclude a small number of runs that diverged during training. To compute a measure of the performance of datasets and pretrained LLMs, we invoke test-time compute: 1,024 Sodaracers are sampled uniformly and evaluated from each dataset/model (recall that there is one model for both cutoff thresholds), and the best performing Sodaracer is considered for each terrain. Figures 11.20 and 11.21 detail the results of these experiments with the TUNNEL and BUMPY distributions, respectively.

In short, Figs. 11.20 and 11.21 help us understand whether RL is able to discover conditional solutions, which we interpret as conditional inventors of Sodaracers that are capable of locomoting on particular terrains. Moreover, Figs. 11.20 and 11.21 enable us to compare the performance of Sodaracers produced at different stages of the pipeline, and how performance is affected by the choice of cutoff threshold. A particularly interesting question is whether RL is able to consistently improve upon the performance of test-time compute with the pretrained models produced in Stage 2.

The RL procedure was at times brittle: training sometimes diverged, and some results were inconsistent. Divergence tended to be more frequent using the ResNet-TENs, which is unsurprising considering the ResNets introduce many more param-

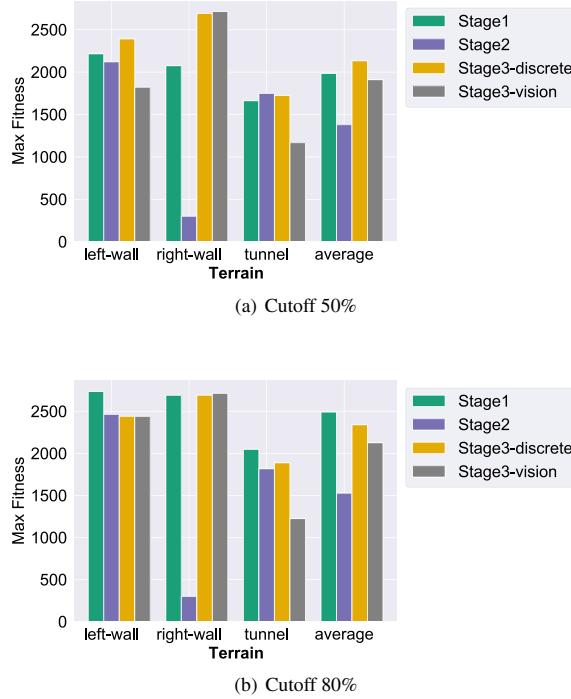


Fig. 11.20 Comparing performance of models and datasets across the stages of the pipeline on the terrain distribution including the TUNNEL. Results are detailed when training the LM using a dataset with a cutoff of **a** 50%, and **b** 80%. Stage 3 models are able to discover conditional solutions in both cases, consistently perform comparably to test-time compute on the dataset, and better than the Stage 2 pretrained LMs. For the 80% cutoff threshold, while performance is better than with 50% at all stages, the pipeline struggles to improve performance over that of the dataset. Conversely, for the 50% cutoff threshold, the Stage 3 (discrete) model improves upon all stages, demonstrating the ability of the pipeline to improve the performance of the models

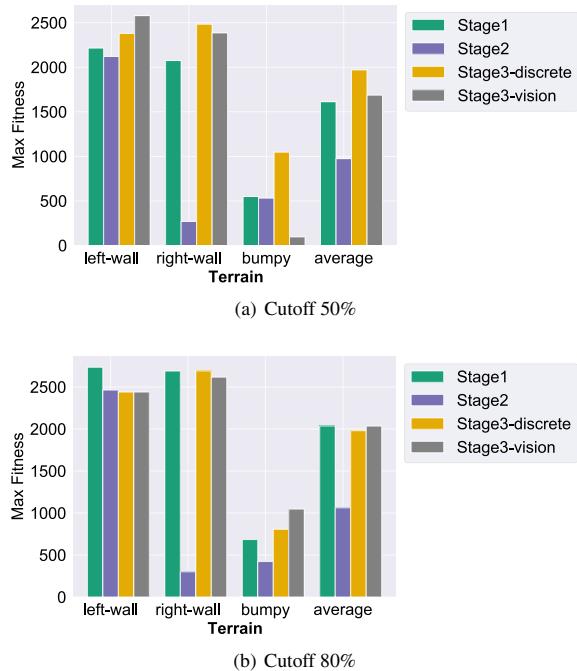
eters to the model, which are in turn trained with an extremely impoverished distribution of images (one for each terrain in the distribution).

Despite the fragility, RL fine-tuning is successful in producing conditional inventors in this domain: the models tend to produce a single Sodaracer for each terrain, which differ across terrains in the distribution. Importantly, the produced Sodaracers achieve good performance for the conditioned terrain, while failing to locomote on the other terrains. Videos showcase Sodaracers invented for the Tunnel distribution [77] and for the bumpy distribution [78]. In short, the main result is the outputs of Stage 3, and thus the complete pipelines are conditional inventors of the desired form.

Moreover, in most cases, the RL models are comparable to or better than the best performing Sodaracers sampled from the dataset or the pretrained LLM. This consistency implies that Stage 3 enables the models to learn to use the TENs in conjunction with the LLMs, and further can fine-tune the models' outputs to improve performance, though not always by significant margins.

Fig. 11.21 Comparing performance of models and datasets across the stages of the pipeline on the terrain distribution

including BUMPY. Results are detailed when training the LM using a dataset with a cutoff of **a** 50% and **b** 80%. Similar trends can be seen to those in Fig. 11.20: Stage 3 models are able to discover conditional solutions and consistently perform comparably to test-time compute on the dataset, improving upon Stage 2 models. For the 80% cutoff threshold, the pipeline achieves comparable performance to that of the dataset, while improving performance for the 50% cutoff threshold



Models trained with a cutoff of 80% tend to achieve slightly better performance, and proved more stable during training, though the differences are not significant. This result implies that the trade-off between data quality and quantity may play a role in downstream tasks (such as RL fine-tuning), a point that warrants further investigation in future work. One interesting avenue for research in this direction is to consider pretraining procedures that include information regarding the quality of the instances (where such information is available), e.g., as proposed by [14].

Finally, we note that “collapsed” solutions in which the same Sodaracer is produced every time a particular terrain is observed (as opposed to significantly different samples each time the same terrain is seen) are sensible in this setting, as there should exist a dominant Sodaracer for each terrain. However, interestingly, in true open-ended systems this property may not hold: if the environment is constantly shifting, which excludes the existence of single, dominant inventions. In such a setting, the stochasticity of the model is expected to be beneficial, enabling the model to adapt and produce a diversity of useful solutions.

11.7.4 Qualitative Observations

Several interesting structures and solution classes were qualitatively observed throughout the experiments, which provide additional insight into the pipeline’s ability to conditionally invent solutions to different terrains. One such example is the

emergence of very *short* Sodaracers, which arose in response to the TUNNEL terrain. The video visualizations [79] highlight examples of such Sodaracers produced in response to TUNNEL.

Another interesting class of Sodaracers appeared in earlier experiments with ELM; a wheel-like structure emerged during the evolutionary process, and persevered throughout the pipeline. During Stage 3, the wheel proved particularly adept at locomoting in the BUMPY terrain, and consistently emerged as the solution to BUMPY produced by the Stage 3 models for that terrain across RL runs. Unfortunately, the wheel did not re-emerge in the ELM runs used in the final experiments in this chapter. The video [80] demonstrates several solutions of this form discovered by RL when trained with the BUMPY terrain distribution as well as the TUNNEL distribution. For contrast, this video [81] shows failure modes on bumpy for several Sodaracers effective in locomoting on flat terrains.

Such qualitative observations provide further evidence that the pipeline is capable of producing interesting inventors and creative solutions to problems, even in a simplified domain that is not open-ended. We hypothesize that when unleashed in more complex domains, this capability of conditional invention will contribute to the open-endedness of the induced process by continually introducing new objects to the environment, and thus changing its properties for other agents.

11.8 Discussion and Conclusion

An important difference between natural evolution and most of EC is the very beginning—nature began with a single “example” or seed, the first cell on Earth, which was already bestowed with critical initial functionality and information. In contrast, runs in EC usually begin with randomized configurations with little or no useful information. Because programming languages like Python for humans are natural modalities for formalizing complicated ideas and relationships, such a program could serve as a seed more in the spirit of nature. However, the problem then is that arbitrary mutations to an already-formulated program are very unlikely to be useful.

A few years ago, the idea that the mutation operator could “know” how to perturb such programs in reasonable and promising ways would be fanciful, but, as shown in this chapter, the emergence of LLMs has now made such capabilities a reality. The MAP-Elites algorithm combined with ELM easily bootstraps datasets of hundreds of thousands of examples in a completely foreign domain (to the initial LLM) from initial human-written seeds. The validity of this generated data is confirmed by the invention pipeline that follows—conditional LLMs were ultimately trained starting from this data that cannot be trained from scratch.

More broadly, the main idea introduced here is that LLMs trained on code open up a significant new kind of intelligent GP enabled by ELM that is no longer at the mercy of the raw search landscape induced by code. While the experiment in this chapter points to a set of implications for open-endedness, deep learning, and RL, the potential applications are numerous and many previous challenges in the GP field could be revisited with this new tool.

The experiment in this chapter shows that intelligent LLM-based mutation operators can successfully drive exploration by being combined with other search algorithms (e.g., MAP-Elites in this work). Furthermore, optimizing such mutation operators based on the quality of their output during the search itself appears to make them work even better for exploration. Not only are the discoveries of such search potentially useful in their own right (like wheels in the Sodarace domain), but they offer an entirely new option for generating example data or optimizing existing solutions in domains where data is sparse or non-existent. For example, such search through LLM-based perturbation could feasibly be applied to optimize the MAP-Elites search algorithm itself, or for LLM architecture and hyperparameter search.

From the perspective of open-endedness, the challenge in principle is that the search is by definition continually and even intentionally shifting out of distribution. As soon as a new invention or DCT is achieved, open-endedness demands that its now-familiar comfort zone be at least partially abandoned for new frontiers. The experiment here wherein LLMs trained from simple flat-ground walkers were able to leverage that knowledge to appropriately generate specialized walkers for different terrains shows just this kind of informed leap to a new frontier. If such a process of leaps upon leaps can be made to continue indefinitely, then an unbounded explosion of emergent complexity could be within reach.

One important question for future work is the extent to which the resultant model can interpolate or extrapolate to examples (i.e., environments) outside its training distribution. While RL can harness existing knowledge in the LLM to bootstrap into new tasks, extrapolating principles from such knowledge is much harder and likely to require further weight updates through additional learning. It is possible that a sophisticated future open-ended system would entangle both continual evolution and RL for DCTs together.

Overall, the hope is that the simple core insight that the efficacy of mutation in GP can now dramatically improve through ELM will inspire a broad array of novel applications and research directions. The observation that EC can benefit directly and dramatically from advances in deep learning (and deep learning from EC further down the invention pipeline) can also help to motivate the further pursuit of synergies between the fields.

References

1. Detailed description of unified format. Detailed-Unified.html. https://www.gnu.org/software/difutils/manual/html_node/
2. Open AI blogpost: New GPT-3 capabilities: Edit and insert (2022). <https://openai.com/blog/gpt-3-edit-insert/>
3. Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B., Mordatch, I.: Emergent tool use from multi-agent autocurricula (2019). [arXiv:1909.07528](https://arxiv.org/abs/1909.07528)
4. Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.D.: Genetic Programming - An Introduction. Morgan Kaufmann Publishers Inc. (1998)
5. Bedau, M.A., McCaskill, J.S., Packard, N.H., Rasmussen, S., Adami, C., Green, D.G., Ikegami, T., Kaneko, K., Ray, T.S.: Open problems in artificial life. *Artif. Life* 6(4), 363–376 (2000)

6. Bentley, P.J., Kumar, S.: Three ways to grow designs: a comparison of embryogenies for an evolutionary design problem. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-1999), pp. 35–43 (1999)
7. Bommasani, R., Hudson, D.A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M.S., Bohg, J., Bosselut, A., Brunskill, E., et al.: On the opportunities and risks of foundation models (2021). [arXiv:2108.07258](https://arxiv.org/abs/2108.07258)
8. Bongard, J.C., Pfeifer, R.: Repeated structure and dissociation of genotypic and phenotypic complexity in artificial ontogeny. In: Spector, L., Goodman, E.D., Wu, A., Langdon, W.B., Voigt, H.-M., Gen, M., Sen, S., Dorigo, M., Pezeshk, S., Garzon, M.H., Burke, E. (eds.) Genetic and Evolutionary Computation Conference, pp. 829–836 (2001)
9. Bradley, H., Fan, H., Saini, H., Adithyan, R., Purohit, S., Lehman, J.: Diff models - a new way to edit code. CarperAI Blog (2023)
10. Brameier, M., Banzhaf, W.: A comparison of linear genetic programming and neural networks in medical data mining. IEEE Trans. Evol. Comput. **5**(1), 17–26 (2001)
11. Brant, J.C., Stanley, K.O.: Minimal criterion coevolution: a new approach to open-ended search. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 67–74. ACM (2017)
12. Brant, J.C., Stanley, K.O.: Diversity preservation in minimal criterion coevolution through resource limitation. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference, GECCO '20, pp. 58–66, New York, NY, USA. Association for Computing Machinery (2020)
13. Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D.M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., Amodei, D.: Language models are few-shot learners (2020). [arXiv:2005.14165](https://arxiv.org/abs/2005.14165)
14. Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., Mordatch, I.: Decision transformer: reinforcement learning via sequence modeling (2021). [arXiv:2106.01345](https://arxiv.org/abs/2106.01345)
15. Chen, M., Tworek, J., Jun, H., Yuan, Q., de Oliveira Pinto, H.P., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., et al.: Evaluating large language models trained on code (2021). [arXiv:2107.03374](https://arxiv.org/abs/2107.03374)
16. Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H.W., Sutton, C., Gehrmann, S., et al.: Palm: scaling language modeling with pathways (2022). [arXiv:2204.02311](https://arxiv.org/abs/2204.02311)
17. Christiano, P., Leike, J., Brown, T.B., Martic, M., Legg, S., Amodei, D.: Deep reinforcement learning from human preferences (2017). [arXiv:1706.03741](https://arxiv.org/abs/1706.03741)
18. Cully, A., Clune, J., Tarapore, D., Mouret, J.-B.: Robots that can adapt like animals. Nature **521**(7553), 503–507 (2015)
19. De Jong, K.A.: Evolutionary Computation: A Unified Approach. MIT Press, Cambridge, MA (2006)
20. Dennis, M., Jaques, N., Vinitsky, E., Bayen, A., Russell, S., Critch, A., Levine, S.: Emergent complexity and zero-shot transfer via unsupervised environment design. Adv. Neural Inf. Process. Syst. **33**, 13049–13061 (2020)
21. Devlin, J., Chang, M.-W., Lee, K., Toutanova, K.: Bert: pre-training of deep bidirectional transformers for language understanding (2018). [arXiv:1810.04805](https://arxiv.org/abs/1810.04805)
22. Earle, S., Togelius, J., Soros, L.B.: Video games as a testbed for open-ended phenomena. In: 2021 IEEE Conference on Games (CoG), pp. 1–9. IEEE (2021)
23. Galván-López, E., McDermott, J., O'Neill, M., Brabazon, A.: Towards an understanding of locality in genetic programming. In: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, pp. 901–908 (2010)
24. Grbic, D., Palm, R.B., Najarro, E., Glanois, C., Risi, S.: Evocraft: a new challenge for open-endedness. In: International Conference on the Applications of Evolutionary Computation (Part of EvoStar), pp. 325–340. Springer (2021)

25. Gugerty, L., Olson, G.: Debugging by skilled and novice programmers. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 171–174 (1986)
26. Hansen, N.: The CMA evolution strategy: a comparing review. In: Towards a New Evolutionary Computation, pp. 75–102. Springer (2006)
27. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition (2015). [arXiv:1512.03385](https://arxiv.org/abs/1512.03385)
28. Hendrycks, D., Basart, S., Kadavath, S., Mazeika, M., Arora, A., Guo, E., Burns, C., Puranik, S., He, H., Song, D., et al.: Measuring coding challenge competence with apps (2021). [arXiv:2105.09938](https://arxiv.org/abs/2105.09938)
29. Jonyer, I., Himes, A.: Improving modularity in genetic programming using graph-based data mining. In: FLAIRS Conference, pp. 556–561 (2006)
30. Keskar, N.S., McCann, B., Varshney, L.R., Xiong, C., Socher, R.: CTRL: a conditional transformer language model for controllable generation (2019). [arXiv:1909.05858](https://arxiv.org/abs/1909.05858)
31. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge, MA (1992)
32. Koza, J.R., Keane, M.A., Streeter, M.J., Mydlowec, W., Yu, J., Lanza, G.: Genetic Programming IV: Routine human-competitive machine intelligence, vol. 5. Springer Science & Business Media, 2006
33. Kramer, O.: Evolutionary self-adaptation: a survey of operators and strategy parameters. Evolutionary Intelligence **3**(2), 51–65 (2010)
34. Langdon, W.B., Poli, R.: Foundations of Genetic Programming. Springer Science & Business Media (2013)
35. Lehman, J., Chen, J., Clune, J., Stanley, K.O.: Safe mutations for deep and recurrent neural networks through output gradients. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 117–124. ACM (2018)
36. Lehman, J., Gordon, J., Jain, S., Ndousse, K., Yeh, C., Stanley, K.O.: Evolution through large models (2022). [arxiv:2206.08896](https://arxiv.org/abs/2206.08896)
37. Lehman, J., Stanley, K.O.: Abandoning objectives: evolution through the search for novelty alone. Evol. Comput. **19**(2), 189–223 (2011)
38. Li, P.L., Ko, A.J., Zhu, J.: What makes a great software engineer? In: 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, vol. 1, pp. 700–710. IEEE (2015)
39. Li, R., Allal, L.B., Zi, Y., Muenninghoff, N., Kocetkov, D., Mou, C., Marone, M., Akiki, C., Li, J., Chim, J., et al.: Starcoder: may the source be with you! (2023). [arXiv:2305.06161](https://arxiv.org/abs/2305.06161)
40. Li, Y., Choi, D., Chung, J., Kushman, N., Schrittweiser, J., Leblond, R., Eccles, T., Keeling, J., Gimeno, F., Lago, A.D., et al.: Competition-level code generation with alphacode (2022). [arXiv:2203.07814](https://arxiv.org/abs/2203.07814)
41. McOwan, P., Burton, E.: SodaRace website (2000–2013). <http://sodarace.net>
42. Meyer-Nieberg, S., Beyer, H.-G.: Self-adaptation in evolutionary algorithms. In: Parameter Setting in Evolutionary Algorithms, pp. 47–75. Springer (2007)
43. Mitchell, M.: An Introduction to Genetic Algorithms. MIT Press (1999)
44. Mouret, J.-B., Doncieux, S.: Encouraging behavioral diversity in evolutionary robotics: an empirical study. Evol. Comput. **20**(1), 91–133 (2012)
45. Mouret, J.-B., Clune, J.: Illuminating search spaces by mapping elites (2015). [arXiv:1504.04909](https://arxiv.org/abs/1504.04909)
46. OpenAI. Gpt-4 technical report (2023)
47. Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C.L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al.: Training language models to follow instructions with human feedback (2022). [arXiv:2203.02155](https://arxiv.org/abs/2203.02155)
48. O’Neill, M., Vanneschi, L., Gustafson, S., Banzhaf, W.: Open issues in genetic programming. Gen. Progr. Evolvable Mach. **11**(3), 339–363 (2010)
49. Pathak, D., Agrawal, P., Efros, A.A., Darrell, T.: Curiosity-driven exploration by self-supervised prediction. In: International Conference on Machine Learning, pp. 2778–2787. PMLR (2017)
50. Pugh, J.K., Soros, L.B., Stanley, K.O.: Quality diversity: a new frontier for evolutionary computation. Front. Robot. AI **40** (2016)

51. Ray, A., McCandlish, S.: Independent contribution: training diff models (2020)
52. Salustowicz, R., Schmidhuber, J.: Probabilistic incremental program evolution. *Evol. Comput.* **5**(2), 123–141 (1997)
53. Schulman, J., Levine, S., Abbeel, P., Jordan, M., Moritz, P.: Trust region policy optimization. In: International Conference on Machine Learning, pp. 1889–1897 (2015)
54. Schulman, J., Moritz, P., Levine, S., Jordan, M., Abbeel, P.: High-dimensional continuous control using generalized advantage estimation (2015). [arXiv:1506.02438](https://arxiv.org/abs/1506.02438)
55. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms (2017). [arXiv:1707.06347](https://arxiv.org/abs/1707.06347)
56. Seront, G.: External concepts reuse in genetic programming. In: Working Notes for the AAAI Symposium on Genetic Programming, pp. 94–98. MIT/AAAI Cambridge (1995)
57. Soros, L.B., Stanley, K.O.: Identifying minimal conditions for open-ended evolution through the artificial life world of chromaria. In: Proceedings of the Fourteenth International Conference on the Synthesis and Simulation of Living Systems, Cambridge, MA, pp. 793–800. MIT Press (2014)
58. Spector, L., Robinson, A.: Genetic programming and autoconstructive evolution with the push programming language. *Gen. Progr. Evolvable Mach.* **3**(1), 7–40 (2002)
59. Standish, R.K.: Open-ended artificial evolution. *Int. J. Comput. Intell. Appl.* **3**(02), 167–175 (2003)
60. Stanley, K.O.: Compositional pattern producing networks: A novel abstraction of development. *Gen. Progr. Evolvable Mach. Spec. Issue Devel. Syst.* **8**(2), 131–162 (2007)
61. Stanley, K.O., Lehman, J., Soros, L.: Open-endedness: the last grand challenge you've never heard of. O'Reilly Radar Online Article (2017)
62. Stanley, K.O., Miikkulainen, R.: A taxonomy for artificial embryogeny. *Artif. Life* **9**(2), 93–130 (2003)
63. Stanley, K.O., Lehman, J., Soros, L.: Open-endedness: the last grand challenge you've never heard of. O'Reilly Online. Accessed 19 Dec 2017
64. Stanton, C., Clune, J.: Curiosity search: producing generalists by encouraging individuals to continually explore and acquire skills throughout their lifetime. *PloS ONE* **11**(9), e0162235 (2016)
65. Stiennon, N., Ouyang, L., Wu, J., Ziegler, Lowe, D.M.R., Voss, C., Radford, A., Amodei, D., Christiano, P.: Learning to summarize from human feedback (2020). [arXiv:2009.01325](https://arxiv.org/abs/2009.01325)
66. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction (1998)
67. Szerlip, P., Stanley, K.O.: Indirectly encoding running and jumping sodarace creatures for artificial life. *Artif. Life* **21**(4), 432–444 (2015)
68. Taylor, T.: Exploring the concept of open-ended evolution. In: Artificial Life 13 (Proceedings of the Thirteenth International Conference on the Simulation and Synthesis of Living Systems), Cambridge, MA, pp. 540–541. MIT Press (2012)
69. Taylor, T., Bedau, M., Channon, A., Ackley, D., Banzhaf, W., Beslon, G., Dolson, E., Froese, T., Hickinbotham, S., Ikegami, T., et al.: Open-ended evolution: Perspectives from the OEE workshop in York. *Artif. Life* **22**(3), 408–423 (2016)
70. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. *Adv. Neural Inf. Proc. Syst.* **30** (2017)
71. Video. Available at: <https://y2u.be/jeP8Nslu48>
72. Video. Available at: <https://y2u.be/QNyNtvwA9FI>
73. Video. Available at: <https://y2u.be/M9pAjUx6dyM>
74. Video. Available at: <https://y2u.be/8C2K5fk28HI>
75. Video. Available at: <https://y2u.be/-LW2cCwSdRU>
76. Video. Available at: <https://y2u.be/XR3L4cZ83xU>
77. Video. Available at: <https://y2u.be/e53NwdT4RdM>
78. Video. Available at: <https://y2u.be/WEM1dBtLLTw>
79. Video. Available at: https://y2u.be/P9A1ruI3_tU
80. Video. Available at: <https://y2u.be/l5PVSLDknWM>
81. Video. Available at: <https://y2u.be/Mo-rXnFq6vQ>

82. Wang, B., Komatsuzaki, A.: GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model (2021). <https://github.com/kingoflolz/mesh-transformer-jax>
83. Wang, R., Lehman, J., Clune, J., Stanley, K.O.: Poet: open-ended coevolution of environments and their optimized solutions. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '19, New York, NY, USA, pp. 142–151. Association for Computing Machinery (2019)
84. Wang, R., Lehman, J., Rawal, A., Zhi, J., Li, Y., Clune, J., Stanley, K.O.: Enhanced poet: open-ended reinforcement learning through unbounded invention of learning challenges and their solutions. In: International Conference on Machine Learning, pp. 9940–9951. PMLR (2020)
85. Wierstra, D., Schaul, T., Peters, J., Schmidhuber, J.: Natural evolution strategies. In: IEEE Congress on Evolutionary Computation. CEC 2008. (IEEE World Congress on Computational Intelligence), pp. 3381–3387. IEEE (2008)

Chapter 12

Hardware-Aware Evolutionary Approaches to Deep Neural Networks



Lukas Sekanina, Vojtech Mrazek, and Michal Pinos

Abstract This chapter gives an overview of evolutionary algorithm (EA) based methods applied to the design of efficient implementations of deep neural networks (DNN). We introduce various acceleration hardware platforms for DNNs developed especially for energy-efficient computing in edge devices. In addition to evolutionary optimization of their particular components or settings, we will describe neural architecture search (NAS) methods adopted to directly design highly optimized DNN architectures for a given hardware platform. Techniques that co-optimize hardware platforms and neural network architecture to maximize the accuracy-energy trade-offs will be emphasized. Case studies will primarily be devoted to NAS for image classification. Finally, the open challenges of this popular research area will be discussed.

12.1 Introduction

Previous chapters have shown that Evolutionary Algorithms (EA) can be utilized for neural architectures search (NAS) and for solving various hard optimization problems in the scope of machine learning applications. This chapter is devoted to the use of evolutionary algorithms in the task of discovering high-quality *implementations* of deep learning algorithms. We will focus on deep neural networks (DNN) and their distinct subclass—*convolutional neural networks* (CNN)—that are currently employed as machine learning engines in many challenging applications operated on different types of platforms, ranging from ultra-low-power edge devices via mobile

L. Sekanina (✉) · V. Mrazek · M. Pinos

Faculty of Information Technology, Brno University of Technology, Brno, Czech Republic
e-mail: sekanina@fit.vutbr.cz

V. Mrazek

e-mail: mrazek@fit.vutbr.cz

M. Pinos

e-mail: ipinos@fit.vutbr.cz

phones to high-performance accelerators in data centers [8, 28, 68]. Hence, in addition to producing high-quality outputs, many of these implementations have to be energy efficient. This is achieved by developing specialized neural architectures and hardware inference accelerators for CNNs (and DNNs in general).

Section 12.2 introduces the principles of efficient processing of CNNs in specialized hardware. It describes two fundamental architectures of CNN accelerators that are currently used—*temporal architecture* which is typical for common processors and Graphic Processing Units (GPUs), and *spatial architecture*, often adopted in application-specific integrated circuits (ASICs) or field programmable gate arrays (FPGAs). Particularly we emphasize the role of *mapping*, i.e., the strategy determining how a computational graph of (a potentially very complex) CNN is executed on limited hardware resources available on a chip. This chapter also deals with hardware simulators and fast predictors of hardware parameters of CNNs.

In Sect. 12.3, we start with a fully trained CNN model and discuss various optimizations that can be conducted by EAs to obtain its hardware implementation with desired properties. We focus on the evolutionary design of components (such as approximate multipliers and activation functions) of hardware accelerators, optimized precision scaling, evolutionary optimization of the CNN-to-accelerator mapping, and weight compression.

Section 12.4 is devoted to *hardware-aware NAS methods* and *NAS methods with hardware co-design*. Considering a given task (e.g., image classification) and target hardware, the *hardware-aware NAS* algorithms try to deliver the most suitable CNN architecture whose hardware implementation satisfies given constraints, e.g., on maximum latency. Note that the hardware platform is not directly optimized; it can be seen as a series of constraints for the NAS method. *NAS with hardware co-optimization* evolves CNN architecture and configuration of a configurable hardware accelerator in parallel, i.e., in addition to the space of CNN architectures, it optimizes hardware configuration (e.g., type of used resources, mapping strategies, buffer sizes, and compiler options). We survey the key evolutionary NAS methods addressing the above-mentioned approaches.

Finally, conclusions and open research challenges are presented in Sect. 12.5. Specifically, we address the problem of benchmarking of hardware-aware NAS methods, security & reliability issues, novel unconventional hardware platforms for DNNs, and design time reduction.

12.2 Hardware Platforms for Efficient Processing of DNNs

The efficient processing of deep neural networks has been addressed in the literature in great detail, including a prominent book [69] and comprehensive surveys [8, 46, 51, 61]. This section aims to explain the basic concepts of efficient processing of CNNs and briefly survey hardware accelerators introduced for inference. First, we recall the principles and terminology of CNNs in Sect. 12.2.1. We distinguish two main architectures—temporal architecture (Sect. 12.2.2) and spatial architecture

(Sect. 12.2.3). In Sect. 12.2.4, we will also deal with simulators and performance predictors developed to simplify the hardware accelerator design process.

12.2.1 Convolutional Layers

A typical CNN consists of convolutional layers, pooling layers, fully connected layers, and some other less computationally intensive units. Convolutional layers are responsible for more than 90% of overall computation, dominating runtime and energy consumption of inference [69]. Figure 12.1 illustrates how the output, the so-called *output feature maps* (**O**), of a convolutional layer, are obtained. The *input feature maps* (**I**), holding either the input image or an intermediate result of a previous layer, are processed by applying a set of *filter weights* (**Weights**). As multiple input feature maps can be processed in parallel, multiple output feature maps are obtained, where **N** is their number; **N** also denotes the batch size. Table 12.1 summarizes all symbols used.

A straightforward software implementation of the convolutional layer operation is depicted in the algorithm of Fig. 12.2. The core operation is the so-called Multiply-And-Accumulate (MAC) operation. It multiplies a weight with an input activation and adds the product to a partial sum. Figure 12.3 shows its basic circuit implementation, including the number of bits for each signal when a fixed-point (FX) number representation is utilized. The $2N$ -bit product is added to a $2N + M$ bit partial sum,

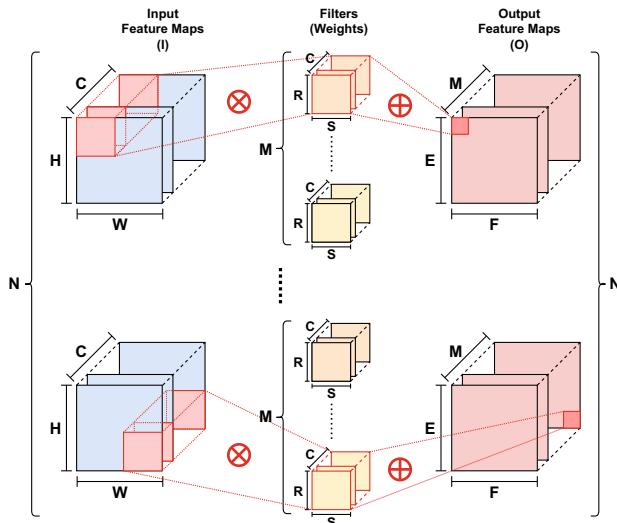


Fig. 12.1 Applying a set of filters on the input feature maps to calculate the output feature maps in convolutional layers. Symbols are defined in Table 12.1

Table 12.1 Symbols used to describe convolution layers

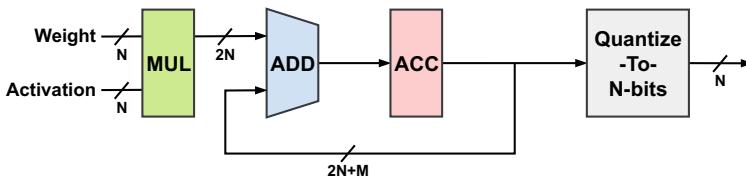
Symbol	Description
H	Input feature map height
W	Input feature map width
C	Number of input channels
R	Filter height
S	Filter width
M	Number of output channels
E	Output feature map height
F	Output feature map width
N	Number of input/output feature maps
U	Stride

Algorithm 1: Generalized convolution in CNNs

```

1 for ( $n = 0; n < \text{N}; n++$ ) {
2   for ( $m = 0; m < \text{M}; m++$ ) {
3     for ( $x = 0; x < \text{F}; x++$ ) {
4       for ( $y = 0; y < \text{E}; y++$ ) {
5          $\text{O}[n][m][x][y] = 0;$ 
6         for ( $i = 0; i < \text{R}; i++$ ) {
7           for ( $j = 0; j < \text{S}; j++$ ) {
8             for ( $k = 0; k < \text{C}; k++$ ) {
9                $\text{O}[n][m][x][y] +=$ 
10               $\text{I}[n][k][\text{U}x+i][\text{U}y+j] \times \text{Weight}[n][k][i][j];$ 
11            }
12          }
13        }
14      }
15    }
16  }
17   $\text{O}[n][m][x][y] += \text{B}[m];$ 
18   $\text{O}[n][m][x][y] = \text{Activation}(\text{O}[n][m][x][y]);$ 

```

Fig. 12.2 Algorithm for convolution**Fig. 12.3** A typical implementation of the Multiply&Accumulate (MAC) circuit utilizing the N -bit fixed-point number representation for the weights

where M depends on the number of weights. The result is quantized into N bits. The floating-point (FP) number representation is typically used for training.

Computing the resulting output feature maps of convolutional layers and fully connected layers is usually transformed into matrix multiplications. The key factors

determining how much time and energy will one inference require are the size of these matrices, resources available on a given hardware platform, and a data flow control algorithm. Since a moderate CNN can have millions of parameters, these matrices do not fit the local memory of the accelerator. Hence, they must be processed at the block level, where a block is a submatrix that can be stored in local memory. These blocks are sent from the main (external) memory to local memory, then read and processed using arithmetic units (in MAC circuits). The partial results are stored in local memory or copied into the main memory when needed.

12.2.2 DNN Accelerators: Temporal Architecture

Hardware accelerators with *temporal architecture* employ a set of Arithmetic Logic Units (ALUs) with a fixed connection pattern and a hierarchical memory subsystem. General-purpose CPUs and GPUs are typical examples of temporal architectures. The ALU always receives data from local memory and returns the result to the same memory. As there is minimal support for data reuse, i.e., direct data sending between multiple ALUs, memory access becomes the main performance and energy bottleneck. Libraries such as MKL [25] and cuDNN [12] provide highly optimized matrix multiplication and other algorithms for CPUs and GPUs.

GPUs consist of hundreds to thousands of lightweight processing cores with a high-throughput memory subsystem organized into a high-performance single-instruction multiple data (SIMD) programmable stream architecture. Hence, GPUs are useful for the parallelization of matrix multiplication and other operations conducted on FP data types during DNN training and inference. GPUs range from small devices (e.g., NVIDIA Jetson Nano with 472 GFLOPS and 5–10 W) to high-performance nodes of supercomputers (e.g., NVIDIA V100 with 100 TFLOPS and 300 W).

Compared to GPUs, standard CPUs equipped with a few cores offer limited options for DNN acceleration. However, the data-level parallelism provided by SIMD instructions (SSE, AVX) is often exploited. A detailed survey of CPU-based DNN acceleration techniques is provided by Mittal et al. [47]. Low-cost microcontrollers (MCU) are typically employed to implement DNNs on low-power edge devices. Their instruction set can be enhanced with specialized instructions to accelerate MACs and, thus, convolutions. For example, RI5CY is an open MCU class RISC-V Core for energy-efficient processing of Quantized NNs (QNN) utilizing specialized SIMD instructions, fast dot product unit, and multiple FX data formats such as INT-2, INT-4, INT-8, INT-16, INT-32 [21]. To provide DNNs on low-power processors, GAP-8 programmable chip featuring an 8-core cluster composed of RISC-V processors, cache, and other components was developed [20].

As programming of CPUs and GPUs does not require hardware design skills, the DNN accelerators based on CPUs and GPUs are, in principle, more accessible to a broader spectrum of designers than specialized ASICs.

12.2.3 DNN Accelerators: Spatial Architecture

The spatial architectures, typically implemented in application-specific integrated circuits or in the field programmable gate arrays, employ an array of many locally communicating processing elements (PE), see Fig. 12.4. Each of them implements a MAC circuit, a small local memory (registers), and a controller. PEs are usually organized as pipelined systolic arrays optimized for fast execution of DNN operations. Hence, a PE can directly send its output to other PE(s), which leads to faster and energy-efficient computing of DNN operations, eliminating thus memory accesses. The used on-chip network determines connection options among the PEs. The execution time and energy of a DNN accelerator are thus primarily determined by the PE array size, memory subsystem, on-chip network, and the so-called data flow organization.

Data flow is a general term covering the computation order and parallelization strategy applied in the accelerator. It defines the order of arithmetic operations and memory accesses to maximize data reuse. The term *mapping* refers to the dataflow strategy (i.e., computation order and parallelism strategy) coupled with the tiling strategy (selecting the size of input data with respect to available hardware resources such as buffers and PEs). A particular mapping of a given DNN on hardware resources and executing this mapping is implemented by the accelerator controller.

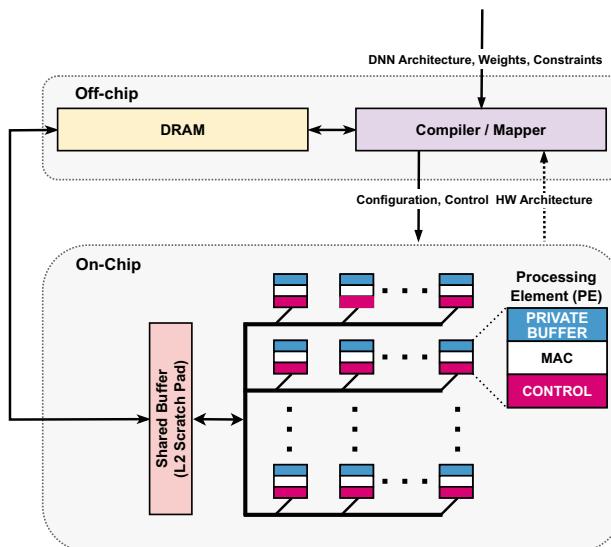


Fig. 12.4 Generic programmable hardware spatial accelerator for DNN inference. Processing elements form a two-dimensional systolic array enabling highly efficient parallel computing and data reuse. The weights are stored in the external DRAM memory; some can be cached on the chip. Based on the DNN architecture description, hardware configuration, and other constraints, the compiler (mapper) generates the mapping of the DNN on hardware resources and its execution plan

ASIC accelerators for DNNs (as surveyed [8, 69]) share the implementation ideas discussed in the previous paragraph. However, their various implementations differ in many aspects, including the fabrication technology, maximum operation frequency, bit precision, the size (of the PE array and on-chip memory), interconnection network, dataflow algorithm, support for weight compression, etc.

These accelerators often utilize the principles of *approximate computing* to provide the best trade-offs between inference accuracy and other objectives (performance, energy). Common approximation techniques are precision scaling, employing approximate arithmetic operations, network pruning, and approximate memory [45]. The most significant gains are obtained when the cross-layer approximation approach is adopted, involving software, architecture, and hardware, breaking thus conventional methods focused on optimizing each layer of abstraction independently [72]. Recent ASIC accelerators also employ the principles of *in-memory computing* to alleviate the data communication bottleneck between PEs and memory elements. In-memory computing aims to extend typical memory architecture with the capability of performing some arithmetic operations to accelerate data processing [66].

Google's Tensor Processing Unit (TPU) family is an example of DNN accelerators implemented as ASICs [28]. TPUv1, introduced in 2016, provided a systolic array of 256×256 8-bit FX multipliers allowing significantly accelerated matrix multiplications for CNN inference (with the peak performance 92 TOPS at 75 W). TPUv2 and TPUv3 offer increased performance and support FP operations which make them usable for DNN training. EdgeTPU, with a peak performance of 4 TOPs and 2 TOPs/W, was developed for edge computing and smartphones. It is programmed using TensorFlow Lite models.

FPGAs have traditionally been seen as programmable arrays of logic blocks whose function is defined by means of look-up tables (LUT) and whose interconnection is based on programmable switches. However, modern FPGAs are heterogenous systems containing not only programmable logic but also embedded memories (BRAM), processors, programmable interfaces, and other specialized circuits (the so-called hard blocks) such as configurable digital signal processing (DSP) blocks. The DSP blocks are especially useful for accelerating the convolution operations of DNNs. Designers can currently choose from various models ranging from small Xilinx Zynq chips suitable for IoT nodes to complex systems on a chip such as Xilinx Versal integrating programmable logic for flexible parallel compute-intensive tasks, processors for sequential processing tasks, and vector processors for domain-specific parallel data processing [46, 61].

Table 12.2 summarizes the key parameters of selected platforms when programmed to accelerate the AlexNet inference. While ASIC provides the most energy-efficient DNN processing (see the Efficiency column), Titan X GPU shows the highest performance (see the Performance column). The architecture of a given chip, fabrication technology, and operational frequency are primarily determining these properties.

Table 12.2 Performance and energy efficiency of AlexNet on various platforms (composed using [10, 79])

Platform	Chip	Freq. (MHz)	Precision	Perform. (inference/s)	Power (W)	Efficiency (inference/s/W)
ASIC	Eyeriss	200	FX16	34.7	0.3	124.8
FPGA	Kintex KU115	235	FX8	2252	22.9	98.3
FPGA	Kintex KU115	235	FX16	1126	22.9	49.2
FPGA	Zynq XC7Z045	200	FX8	340	7.2	47.2
FPGA	Zynq XC7Z045	200	FX16	170	7.2	23.6
GPU	Jetson TX2	1300	FP16	250	10.7	23.3
GPU	Titan X	1417	FP32	5120	227.0	22.6
CPU	Core-i7	3500	FP32	162	73.0	2.2

12.2.4 Hardware Simulators and Performance Predictors

Suppose we have a CNN model (i.e., a computation graph) and need to know the hardware parameters (e.g., latency and energy) of its potential implementation on a given accelerator. However, there are usually many options on how to configure a given accelerator, map the CNN on the available resources, and schedule the CNN processing.

To choose the most suitable implementation, a search has to be conducted in the space of possible mappings and hardware configurations. The objective is to minimize the inference time (latency), energy, or other parameters. From Table 12.3, presenting the cost of typical operations conducted on a chip (when a 45 nm technology is considered), one can conclude that (1) minimizing the access to external memory has to be optimized with the highest priority, and (2) optimizing the bit width saves some energy not only when arithmetic operations are conducted but also when data are moved to/from memory; moreover, shorter weights will reduce the memory size.

Hardware parameters of a given CNN implementation are usually obtained using the following methods:

- Measuring a real hardware implementation provides exact values; however, it is time-consuming to build and measure real hardware.
- Simulation using precise hardware simulators can provide accurate results, but it is still time-consuming when conducted at the gate level.
- Analytical estimation consists of analyzing the CNN's computational graph and applying precomputed knowledge about the cost of particular operations on given hardware [7].

Table 12.3 Energy needed to perform selected operations on a chip fabricated in 45 nm technology [69]

Operation (-)	Type (-)	Width (bits)	Energy (pJ)
Add	Integer	8	0.003
		16	0.005
		32	0.1
	Float	16	0.4
		32	0.9
Multiply	Integer	8	0.2
		32	3.1
	Float	16	1.1
		32	3.7
Read	SRAM	32	5
	DRAM	32	640

- Building a surrogate model capable of predicting a given hardware parameter. It requires selecting suitable features for the predictor and collecting annotated data. Various machine learning models have been utilized for this purpose, e.g., linear regression [71], neural network [80], Gaussian process [37], and Bayesian Ridge Regression [78].

Predicting the resulting latency using easy-to-obtain properties of DNNs, such as the number of weights or MACs, is highly unreliable [69]. However, a recent paper shows that one carefully constructed proxy model (predictor) is enough for hardware-aware NAS [37]. This is also documented by detailed benchmarking [31], disclosing that the inference latency and energy of CNN architecture on hardware are strongly correlated. It concludes that an energy constraint can be implicitly mapped to a corresponding latency constraint in NAS methods.

To illustrate the complexity of the mapping optimization, we consider ResNet-50 CNN [23], which should be implemented on Eyeriss and Simba accelerators. Tools such as Accelergy and Timeloop help in determining the most suitable mapping. Accelergy [75] is an early-stage energy and execution time estimation tool. It estimates hardware parameters of a CNN implementation on a given accelerator whose organization is described at the architecture level in the YAML language, i.e., using characteristics such as the number of PEs, memory size, on-chip network, and data flow organization. Plug-ins for different fabrication technologies can be integrated. Timeloop [53] is a tool searching for the most suitable mapping of CNN to hardware accelerator (several search methods are available in the tool). It performs CNN layer-wise data tiling reflecting the memory hierarchy of the accelerator.

As an example, Fig. 12.5 shows a distribution of the energy efficiency of half a million randomly generated mappings for the first convolutional layer of the ResNet-50 network. Timeloop conducted this estimation for two accelerator models: Eyeriss, featuring 168 PEs, and Simba, equipped with 16 PEs. Both models were simulated

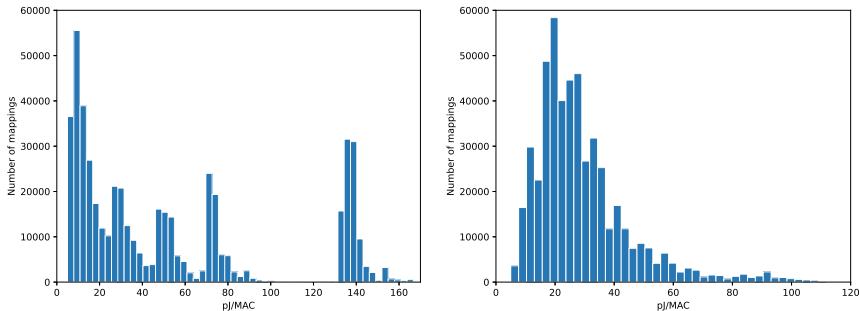


Fig. 12.5 The distribution of mappings showcasing energy efficiency (pJ/MAC) for the first layer of ResNet-50. The mappings were generated by Timeloop for the Eyeriss accelerator (left) and the Simba accelerator (right)

using 45 nm fabrication technology and utilized external weight memory (LPDDR4). Note that the considered layer performs convolution over the $224 \times 224 \times 3$ input feature map, with 7×7 filters, stride 2, and 64 output channels. The energy efficiency of mappings exhibits substantial variations. These variations stem from the diverse options for tiling and scheduling, which are represented by different mappings. The more efficient mappings excel in utilizing buffer capacity, network subsystems, and loop ordering to maximize data reuse. Nevertheless, the optimal mapping is subject to change based on the workload. What may be an optimal mapping for one architecture could prove to be suboptimal or even invalid for another architecture.

12.3 Evolutionary Optimization in DNN Hardware Accelerators

Surveys [44, 61] document the gradually increasing interest in optimizing hardware implementations of fully trained CNNs (i.e., the inference accelerators), especially in the context of edge computing. In this chapter, we present approaches utilizing EAs for this purpose. Our primary focus will be on elucidating the rationale behind employing EAs in each design or optimization problem and discussing the methodologies employed.

12.3.1 Evolutionary Design of Components of Hardware Accelerators

Elementary components of CNN accelerators such as various arithmetic operations can be evolved and optimized to improve hardware parameters of these accelerators,

and in some cases, the accuracy, too. Genetic programming (GP) is almost exclusively used as the evolutionary method. The fitness function reflecting the functionality of candidate designs is based on either (1) applying a candidate solution in a CNN, training the CNN, and interpreting the CNN's error as the fitness value, or (2) comparing the functionality of a candidate solution with a reference implementation of a given component on some data and determining the fitness as, e.g., a mean squared error. While approach (1), in principle, leads to more reliable solutions (the entire CNN is evaluated), approach (2) is computationally less expensive. In this context, typical targets for the evolutionary approach are multipliers, MACs, and activation functions.

12.3.1.1 Approximate Multipliers

The inexact (approximate) multipliers provide inexact products; however, this inexactness can be tolerated because CNNs are often highly error-resilient [60]. In addition to reducing the bit width of multipliers used in MAC units, the approximation can be achieved by simplifying the logic equations specifying the product. The task is to design approximate multipliers showing good trade-offs between the accuracy and hardware parameters (such as energy and latency). In addition to many manual approximation methods, a fully automated circuit approximation methodology based on Cartesian genetic programming (CGP) has been developed [49, 70]. A common strategy is to optimize the multiplier with respect to an exact multiplier. In the fitness function, the error is expressed using error metrics such as the worst-case error or the mean absolute error. If a multiplier showing a good trade-off between the error and hardware parameters is discovered by CGP, it is used instead of the exact multipliers in one or several layers of a CNN. The CNN's accuracy is then determined, typically after a short fine-tuning. Based on the final accuracy, the evolved multiplier is accepted or rejected. This two-step design process is adopted because many candidate multipliers have to be generated, and evaluating each of them directly in the final CNN is very time-consuming.

Every candidate approximate multiplier \tilde{M} , which is generated by a gate-level CGP, has two inputs (n and m bits) and produces a $n + m$ bit output [70]. The objective is to minimize the cost of the circuit (which highly correlates with power consumption) assuming that \tilde{M} shows the worst-case error (WCE) at most ϵ (Eq. 12.1):

$$F(\tilde{M}, \epsilon) = \begin{cases} \text{cost}(\tilde{M}) & \text{if } \text{WCE}(\tilde{M}) \leq \epsilon \wedge \\ & \quad \text{WCE}_{zr}(\tilde{M}) = 0 \\ \infty & \text{otherwise} \end{cases} \quad (12.1)$$

The cost is estimated as the sum of the weighted areas of the gates used in the circuit. As the approximate multipliers are supposed to be used in neural networks, the requirement for accurate multiplying by zero ($\text{WCE}_{zr}(\tilde{M}) = 0$) is integrated together with the WCE constraint in the fitness function. The validity of both

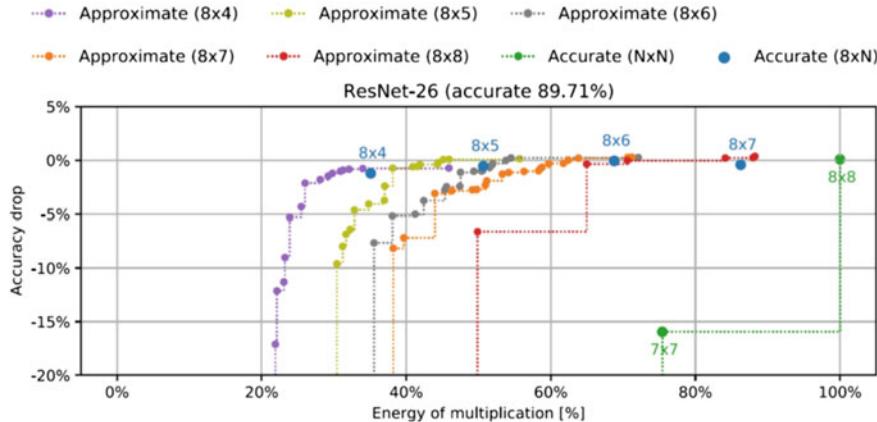


Fig. 12.6 The energy-accuracy trade-offs when all exact 8-bit multiplications of all convolutional layers of ResNet-26 CNN are replaced with an approximate multiplier taken from the EvoApproxLib library of $8 \times N$ -bit approximate multipliers (35 different multiplier implementations tested)

conditions is checked using a single pass of exhaustive simulation of each candidate multiplier. At the end of evolution, the best-scored circuit is synthesized to get all its hardware parameters. The resulting approximate multiplier is also used in a given CNN to obtain classification accuracy, which is typically worsened in comparison with the CNN utilizing exact multipliers. However, the accuracy has usually recovered after retraining for a few epochs.

The case study reported in [49] deals with a situation in which all 8-bit multiplications of all convolutional layers of ResNet CNNs are replaced with one particular approximate implementation of the multiplier. Various evolved approximate $8 \times N$ -bit multipliers that are available in the EvoApproxLib [48] are tested. One operand (the activation) is always at 8 bits and the second operand (the weight) is on N bits, where $N = \{4, 5, 6, 7, 8\}$. Optimizing the bit width leads not only to smaller circuits but also to the reduced size of weight memory. Figure 12.6 shows trade-offs between accuracy and energy of multiplication when ResNet-26 uses various approximate multipliers in its convolutional layers. For a small drop in accuracy, a 50% energy reduction of multipliers is obtained if a suitable approximate multiplier is used. Results are given for the 45 nm process and power supply voltage $V_{dd} = 1$ V. The same approach was taken to design approximate MAC circuits in authors' work [9].

12.3.1.2 Activation Functions

There are some frequently used activation functions such as ReLU or Sigmoid. However, better activation functions can be obtained for particular data sets and CNN architectures. EAs are thus employed to either deliver new functions (without considering their hardware implementation) [4, 52, 65] or optimize existing functions

(with respect to resources) [58]. A common approach to evolve a new activation function is to employ a tree-based genetic programming (GP) which typically utilizes the function set consisting of hardware unfriendly functions (atan , tanh , etc.). The fitness is based on evaluating the entire CNN in which a candidate activation function is embedded [4, 52].

For example, Lapid and Sipper [30] employs coevolution to evolve activation functions for image-classification tasks using CGP. The function set comprises ten commonly used activation functions (ReLU, tanh, ELU, etc.) and 5 arithmetic operations ($+$, $-$, \times , minimum, and maximum). A cooperative coevolution algorithm evolves input-layer, hidden-layer, and output-layer activation functions (each having a separate population). An individual's fitness is determined by the activation function's ability to cooperate with members of the other populations (the fitness procedure is elaborated in [30]). On four classification datasets (MNIST, Fashion-MNIST, KMNIST, USPS) and two neural networks (a 7-layer MLP and an 8-layer CNN), the method was capable of improving the classification accuracy compared to the reference solution.

A hardware-aware evolutionary design of activation functions is conducted by Prashanth and Madhav [58]. Ordinary activation functions (sigmoid, tanh, Gaussian, ReLU, GeLU, Softplus) are considered as golden solutions, and their gate-level implementations are evolved for a given bit width using CGP. In the fitness function, a fully functional solution specified by a truth table is sought in the first step. Once it is obtained, its size is minimized in the second step.

12.3.1.3 Component Selection and Precision Scaling

Suppose that an 8-bit CNN that has to be accelerated consists of more layers (neurons) than processing units available in the accelerator. Furthermore, approximate multipliers can be utilized in configurable processing units. Two tasks have to be solved together: (1) the assignment of the approximate multipliers to MACs of the processing units and (2) the assignment of the convolutional layers to the processing units. ALWANN is an optimization tool capable of selecting a suitable approximate multiplier for each processing unit in such a way that one approximate multiplier serves several layers, and the overall classification error and energy consumption are minimized [50]. The optimizations, including the multiplier selection problem, are solved by means of the NSGA-II algorithm in which the overall CNN accuracy and the energy consumed by the approximate layers are considered. Each candidate solution is uniquely defined by a pair of mappings (map1 , map2), where map1 is a list of k integers in which each integer represents an approximate multiplier (taken from *EvoApproxLib*) assigned to processing units $1 \dots k$, and map2 is another list of l integers in which each integer determines the index of a processing unit that will be used to compute the output of the layer $1 \dots l$. Additional restrictions may be applied depending on the chosen HW accelerator's structure. In order to altogether avoid the computationally expensive retraining of CNN, which is usually employed to improve the classification accuracy, a simple weight updating scheme is proposed

that compensates for the inaccuracy introduced by employing approximate multipliers. ALWANN is evaluated for two architectures of CNN accelerators with approximate multipliers from the open-source EvoApproxLib library while executing three versions of ResNet on CIFAR-10. ALWANN saves 30% of energy needed for multiplication in convolutional layers of ResNet-50 while the accuracy is degraded by only 0.6% (0.9% for the ResNet-14).

Barone et al. [1] propose E-IDEA, an automatic application-driven approximation tool targeting different implementations (hardware and software). E-IDEA uses Clang-Chimera tool to analyze the Abstract Syntax Tree (AST) of the application's source code. Through the so-called mutators, approximations can be introduced at the source code level. The set of mutators includes loop-perforation mutators, precision-scaling mutators for floating-point arithmetic, a precision-scaling mutator for integer arithmetic, and a mutator supporting approximate arithmetic operator models of circuits being part of the EvoApproxLib library. An evolutionary approximation method based on NSGA-II tries to find the best approximation version of a given C/C++ code according to user-defined optimization objectives. A candidate solution is represented as a vector of integers; each of them corresponds to one parameter that can be modified. A set of matching rules specifies the positions in the source code at which a mutation can be applied. E-IDEA was used to approximate weighted sums computed within neurons to reduce hardware requirements and power consumption. Clang-Chimera was configured to truncate input operands and results of multiplications in the three convolutional and the two fully connected layers of the considered network (LeNet). Thus, the tool generates an approximate version of the considered CNN in which it is possible to configure the number of approximate bits to tune the introduced approximation degree for each multiplication involved in the weighted sum. Moreover, Clang-Chimera could select a suitable approximate multiplier from a library of approximate multipliers in these layers. NSGA-II optimized the CNN error on MNIST, aiming at reducing the circuit area. This allowed finding solutions to achieve more than 30% savings, with a negligible accuracy loss (0.48%) compared to the reference solution.

12.3.1.4 The CNN-to-Hardware Mapping Optimization

In the GAMMA (Genetic Algorithm-based Mapper for ML Accelerators) framework, configurable hardware accelerators are considered, i.e., computation order, parallelizing dimensions, and tile sizes can be configured at compile-time [29]. For given constraints (the maximum number of parallelism levels and maximum tile sizes), GAMMA is searching for the most suitable mapping of a CNN layer (see the algorithm in Fig. 12.2) on the hardware resources modeled in MAESTRO [42]. For a given CNN layer, hardware configuration (the number of PEs, local buffer size, global buffer size, latency, and bandwidth), and a mapping strategy, MAESTRO estimates the statistics such as latency, energy, runtime, power, and area.

The mapping is composed of several levels. Each level represents parallelism across a spatial dimension of the accelerator. Figure 12.7 shows how a two-level

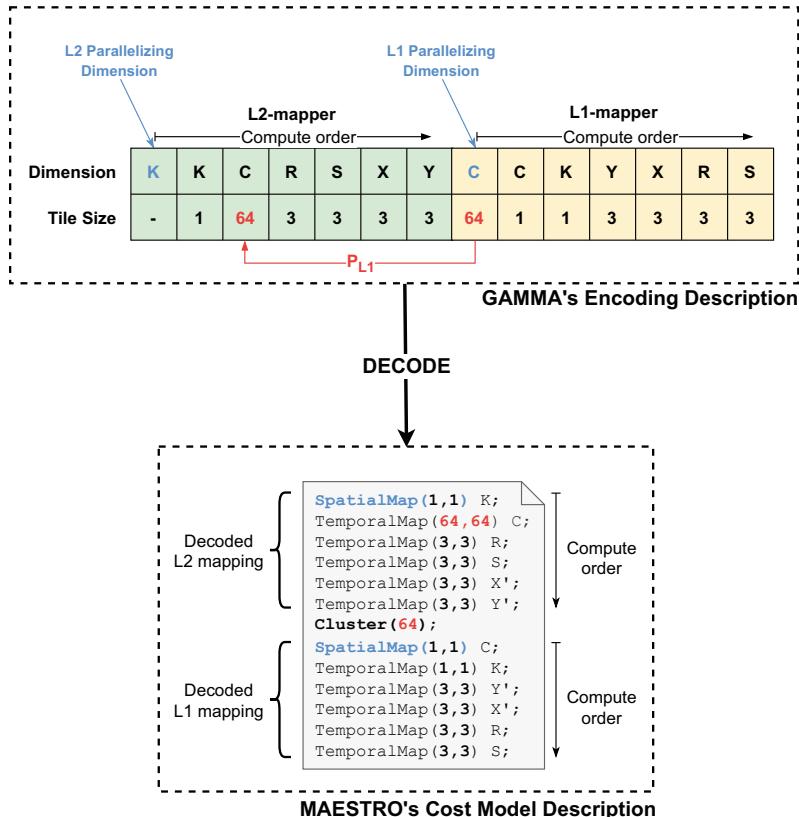


Fig. 12.7 GAMMA's encoding of a two-level mapper (top) and its decoded description for cost model (MAESTRO) of an NVDLA-like two-level mapper (bottom). Adopted from [29]

mapping is encoded in the chromosome. The convolutional layer is specified using C input channels, K output channels, input activations of size $X \times Y$, output activations of size $X' \times Y'$, and filers of size $R \times S$. Each dimension is encoded using seven pairs of values. A pair of genes contains a CNN layer tensor notation (e.g., K, C) and its tile size. The ordering of pairs specifies the computation order. The first pair defines the parallelizing dimension. The L1-mapper describes the inner loop. The L2-mapper describes the outer loop, while containing PL1 number of instances of L1-mapper. The chromosome is used to create a candidate mapping which is then evaluated in MAESTRO.

GAMMA supports several optimization algorithms, including a genetic algorithm with application-specific operators for mutation and crossover [29]. The fitness is defined as a reward value (e.g., latency, energy, or power) if a constraint on hardware resources is met. Otherwise (i.e., when evolved mapping requires more resources than the accelerator provides), a large penalty is assigned. GAMMA is evaluated on

five CNN models with different complexity (VGG16, MobileNet-V2, ResNet-50, ResNet-18, MnasNet) and two platforms (TPU and Eyeriss) with different number of hardware resources. Across CNN models and various hardware platforms considered, GAMMA finds solutions costing $5 \times$ to $(1.2\text{E}+5) \times$ less latency and $2 \times$ to $(1.6\text{E}+4) \times$ less energy.

In another method, AnaCoNGA, the quantization problem and hardware optimization problem are solved concurrently for a given (trained) CNN [19]. The hardware architecture search (HAS) is embedded into quantization strategy search (QSS), in a nested genetic algorithm formulation. For each potential quantization strategy proposed by QSS, the HAS loop efficiently optimizes the accelerator's parameters. In QSS, a multi-objective GA is used to tackle the multi-criteria optimization problem of maximizing accuracy and minimizing hardware-related costs. No hardware design takes place in this search. The quantization search space for a CNN has a size of Q^{2L} , where Q is the set of possible quantization levels for weights and activations, and L is the number of layers in the neural network. In the second GA (HAS), each individual's genome captures hardware parameters such as the PE size, the number of binary dot-products each PE can perform in parallel, and buffer sizes. The fitness criteria of this GA are the hardware design's execution performance (compute cycles and DRAM accesses) of a predetermined quantized CNN, as well as the number of FPGA resources (BRAMs and LUTs) it requires for its allocation. Note that an alternative approach to the nested formulation could be to combine HAS genomes with QSS genomes into one GA. However, this would result in a prohibitively complex and large search space which is difficult for GA.

In order to quickly evaluate candidate hardware designs, an analytical hardware model for the execution of CNN on a state-of-the-art accelerator (such as the Xilinx Z7020 SoC on the PYNQ-Z1 board) is created. AnaCoNGA is evaluated on ResNet-20 (using CIFAR-10 data set), ResNet56 (CIFAR-100), and ResNet-18 (ImageNet). With AnaCoNGA, the accuracy of ResNet-20 (on CIFAR-10) is improved by 2.88% compared to a uniform 2-bit CNN, and achieved a 35% and 37% improvement in latency and DRAM accesses, while reducing LUT and BRAM resources by 9% and 59% respectively, when compared to an edge variant of the accelerator [19].

12.3.1.5 Weight Sharing and Compression

Weight sharing enables to replace a group of similar weights with a single value. Instead of storing all the CNN weights, only a limited number of shared values and a codebook, where original weights are replaced by their corresponding indexes, are stored in the CNN memory. This approach is also known as *weight compression*. For example, if the original weights are encoded on 32 bits and there are 256 different shared values, only the shared values and 8-bit indexes must be stored, reducing thus the memory footprint almost four times compared to the original weight set. Shared weight values are typically obtained with a clustering algorithm like K-means. The weight-sharing idea can be applied at the level of the entire CNN or for each layer separately. The objective is to automatically select the optimal number of shared

values per layer for the input CNN assuming that a range of possible numbers of shared values is provided. Introducing the shared values typically leads to a loss in accuracy. Hence, a suitable trade-off between the accuracy drop and compression rate is sought. Dupuis et al. [17] adopted a two-step approach to compute the shared weights. In the first step, the number of shared values is determined locally and separately for each layer by an exhaustive search. The second step, based on the values obtained in the first step, tries to determine the most suitable combinations of shared weights across the entire CNN. As the number of combinations grows exponentially with the number of layers, the problem is solved by NSGA-II. The bottleneck is the accuracy evaluation because it requires evaluating CNN for each set of candidate (shared) weights. A proxy regression model was created using data obtained in the first step to accelerate the evaluation. The results carried out on recent CNN models, trained with the ImageNet dataset, show over $5 \times$ memory compression at an acceptable accuracy loss without any retraining step.

12.4 NAS Considering the Target Hardware

NAS methods utilizing evolutionary algorithms to deliver a CNN architecture with a minimum error on test data were introduced in previous chapters of this book. *Hardware-aware NAS* methods extend this approach by considering other objectives such as latency, energy efficiency, and memory footprint with respect to a hardware platform implementing the neural network [3, 63]. Hardware-aware NAS methods can be seen as multi-objective optimization methods. Hence, in certain steps of the NAS algorithm, all relevant objectives must be evaluated, either by direct measurement on real hardware or estimated using software models (Sect. 12.2.4). A common approach to solve the multi-objective NAS problem adopted by the NAS community is either (i) to transform it into a single-objective one (using suitable constraints, prioritization, or aggregation techniques) and solve it with a common single-objective method or (ii) to employ a truly multi-objective approach (such as NSGA-II) [15].

A common practice is to model a candidate CNN using a directed acyclic graph encoded as a variable-length string. If only some hyperparameters are optimized then the chromosome is a fixed-length list of integers. All possible strings describing valid CNNs constitute the search space. To reduce the NAS time, the so-called *supernet* is often constructed first [55]. A supernet is an over-parameterized neural network built over a certain backbone CNN model, in which many options are supported for selected hyperparameters. The supernet is trained to solve a given problem. Its training is usually very costly as the supernet is more complex than any individual CNN. However, this cost can be amortized as many suitable subnetworks, including their weights, can be extracted from the resulting supernet for a given specification (e.g., latency, accuracy, or energy constraint on given hardware) and used without repeating the expensive training process. The search for a suitable subnetwork, which can be conducted using an EA, is less expensive because it does not involve any

training. A general limitation of this approach is that the supernet restricts the search space to its subnetworks.

Two major directions can be identified in the area of NAS methods explicitly targeting hardware implementations:

- *Hardware-aware NAS*, whose goal is to find the most suitable CNN model concerning a target hardware platform and the objectives to be optimized. Note that there is no additional search space to the neural architecture search space.
- *NAS with hardware co-optimization*, whose goal is to co-optimize CNN model and hardware configuration (such as amount and type of resources, dataflow strategies, buffer sizes, and compiler options). These methods work in three search spaces (weights, neural architectures, and hardware configurations) and must innovatively orchestrate several search algorithms to produce the best trade-offs between the accuracy and various hardware-relevant metrics.

These two directions will be discussed in the rest of this section.

12.4.1 HW-Aware Evolutionary NAS

An evident approach to optimizing the CNN architecture for given hardware is employing only hardware-friendly hyperparameters and operations, i.e., suitable convolution types, arithmetic operator implementations, quantization schemes, or memory access mechanisms with respect to the optimization objectives. For example, based on benchmarking 32 different operators, Hurricane [78] uses different subsets of operator choices for three hardware platforms. This way, the search space is narrowed toward CNN architectures suitable for a given hardware platform.

12.4.1.1 Classification of Evolutionary HW-Aware NAS Methods

Table 12.4 showcases the key properties of selected evolutionary hardware-aware NAS methods. The search is conducted either at the *macro level* (i.e., the entire CNN is encoded in the chromosome) or the *subnetwork* level (also known as a *block* or a *cell*), in which only a subnetwork is optimized by an EA. The resulting subnetwork can be used multiple times in the final CNN. These NAS methods are often called a *micro-level* NAS. The search space can also be reduced to a few hyperparameters of a given pre-designed CNN architecture (see *hyperp* in the *Search Space* column in Table 12.4). Column *SuperNet* informs whether a supernet is used. The design objectives are listed in the *Objectives* column; the accuracy is not mentioned as it is always involved. The *Estimation Method* column tells us if at all and how particular hardware parameters are estimated. We observe that latency (Lat) and Energy are often estimated rather than measured. If the accuracy (Acc) is estimated, then an NN-based predictor (surrogate) is almost always utilized for this purpose [38, 73]. The *Target device* informs about the target hardware platform(s). Finally, column

Table 12.4 Hardware-aware evolutionary NAS methods. Titles of some data sets are abbreviated, e.g., C-10 for Cifar-10, C-100 for Cifar-100, ImgNet for ImageNet; the + symbol denotes that some additional data sets were omitted because of space limitations

Method	Ref.	Year	Search space	Super net	Objectives	Estimation method	Target device	Data set
Large-Scale	[59]	2017	macro		None	None	GPU	C-10, C-100
JASQNet	[11]	2018	cell		None	None	GPU	ImgNet, C-10
ECAD	[13]	2019	hyperp		Lat, Energy	simulator	FPGA	MNIST
ChamNet	[14]	2019	hyperp		Lat, Energy	Acc, Energy: GP predictor	GPU, DSP, Mobile	ImgNet
LEMONADE	[18]	2019	macro, cell		Params	None	GPU	ImgNet, C-10
NSGANetV1	[39]	2019	block		FLOPS	None	GPU	C-10, C-100
APQ	[73]	2020	block	Y	Lat, Energy	Acc: NN; Lat: LUT	ASIC	ImgNet
DeepMaker	[36]	2020	hyperp		Size	None	CPU, GPU, FPGA	MNIST, C-10, C-100
HNAS	[76]	2020	macro		Lat	Lat: LUT	GPU, Mobile	ImgNet
Hurricane	[78]	2020	macro	Y	Lat	Lat: Bayes, Regression	DSP, CPU, ASIC	ImgNet
MCU Net	[33]	2020	macro	Y	Lat, Mem, Flash	None	MCU	ImgNet, WWF, KWS
NasCaps	[43]	2020	hyperp		Lat, Mem, Energy	Lat: cycles; Energy: model	ASIC	C-10, MNIST, FMMNIST+
NSGANetV2	[38]	2020	block	Y	Lat, MAC, Params	Acc: ML-surrogate	GPU	ImgNet, C-10, C-100+
OFA	[6]	2020	block	Y	Lat	Acc, Lat: NN	GPU, FPGA, Mobile	ImgNet
PONAS	[24]	2020	macro	Y	Params, FLOPS	LUT	GPU	ImgNet
Schorn et al.	[62]	2020	hyperp		Lat, Energy, FT	formula	GPU	C-10, GTSRB
SPOS	[22]	2020	blocks	Y	Lat, FLOPS	None	GPU	ImgNet
μ NAS	[32]	2021	macro		Lat, Mem, MAC	Lat: MAC	MCU	C-10, MNIST, Charts74K+
HSCoNAS	[41]	2021	block	Y	Lat	Lat: formula	GPU, CPU	ImgNet
Prabakaran et al.	[57]	2021	macro		Mem	None	GPU, CPU	anomaly in ECG signals
Wang et al.	[74]	2021	block		FLOPS	None	GPU	ID2012, ISCX VPN
NAS4RRAM	[77]	2021	cells		Energy	simulator	RFAM chip	C-10, C-100
NEMOKD	[67]	2021	hyperp		Lat	None	Movidius	C-10, C-100
GoldenNAS	[40]	2022	layer	Y	Lat	None	Edge GPU, GPU, CPU	ImgNet
Lu et al.	[37]	2022	macro	Y	Lat	MLP	mobile, GPU, CPU	ImgNet

Data Set lists the problems/data set(s) used for evaluation. It has to be noticed that, in addition to image classification, some other tasks are approached.

Some multi-objective NAS methods (e.g., [11, 18, 38, 39]) only optimize the number of FLOPs, which is not highly correlated with the real hardware parameters such as latency and energy. We included these methods in Table 12.4 to cover the whole scope of methods in this area. The following paragraphs briefly present some recent hardware-aware NAS methods utilizing evolutionary algorithms.

12.4.1.2 Selected Evolutionary NAS Methods

The first evolutionary NAS methods such as [59] did not consider any hardware parameters during the evolution. Later, the NAS has become a truly multi-objective method.

The Lamarckian Evolutionary algorithm for Multi-Objective Neural Architecture DEsign (LEMONADE) [18] is a multi-objective NAS. It first selects a subset of architectures, assigning a higher probability to architectures that would fill gaps on the Pareto front for the objectives that can easily be evaluated (e.g., the number of parameters); then, it trains and evaluates only this subset to save computational resources during the architecture search. It proposes a Lamarckian inheritance mechanism that generates child networks that are warm-started with the predictive performance of their trained parents. This is accomplished by using (approximate) network morphism operators for generating children. Within 5 days on 16 GPUs, LEMONADE discovers architectures that are competitive in terms of predictive performance and resource consumption with hand-designed networks, such as MobileNet-V2.

Schorn et al. [62] also employ a set of evaluation functions for the prediction of energy consumption, latency, and required bandwidth of DNNs on hardware, solely based on the topology of neural architecture to avoid the need for expensive simulations or training of candidate CNNs. Furthermore, they also consider error resilience as one of the objectives. Error resilience is seen as the robustness of the neural network classifier against perturbations in its neuron activation values. Such perturbations can be the result of random hardware faults, such as radiation-induced bit-flips. Hence, random bit-flip error simulations are used to evaluate the actual resilience of the obtained set of neural networks. Evolved CNNs achieve about a $6 \times$ to $7 \times$ lower data corruption rate at 0.5% bit error rate in the feature maps of the network in comparison with MobileNet-V2.

GoldenNAS [40] introduces a novel dynamic channel scaling scheme to enable the channel-level search, a progressive space shrinking method to progressively shrink the search space toward target hardware, and an adaptive batch normalization technique to enable the depthwise adaptiveness of CNNs under dynamic environments. GoldenNAS adopts the weight-sharing technique based on the supernet paradigm, where the supernet is derived from ShuffleNetV2. Multiobjective EA samples the supernet to obtain CNNs showing suitable trade-offs between accuracy and latency for various hardware platforms—GPU (Nvidia Quadro GV100), CPU (Intel Xeon

Gold 6136), and edge device (Nvidia Jetson Xavier). Latency is modeled analytically. Note that supernet training for 100 epochs takes about 70 GPU hours, and each stage of the progressive space shrinking takes about 22 GPU hours. The evolutionary search process requires about 6 GPU hours to finish.

Lu et al. [37] utilizes latency monotonicity (i.e., the observation that the architecture latency rankings on different devices are often correlated on other devices) to reuse models from one proxy device on several target devices (several mobile and non-mobile devices tested in this paper). It avoids building a latency predictor for each target device. Hence, only one latency predictor based on an MLP with four layers is used. The search space is built up on MobileNet-V2 with multiplier 1.3, with the channel number in each block fixed. The optimization involves the depth of each stage, the kernel size of convolutional layers, and the expansion ratio of each block. The depth can be chosen from {2, 3, 4}, kernel size can be {3, 5, 7}, and candidate expansion ratios are {3, 4, 6}. There are five stages whose configurations can be searched. The one-shot NAS utilizes Once-For-All network [6] as a supernet. EA is searching for optimal architectures using one proxy device with 1000 individuals in the population and 50 generations for each latency constraint. The evolutionary search takes less than 30 s for each run.

APQ [73] also exploits the supernet. It utilizes a joint model architecture-pruning-quantization search. A mixed quantization is applied after extracting pruned subnetworks from the supernet. An energy/latency look-up table is used to provide the hardware feedback during the search.

12.4.2 NAS with Hardware Co-design

As emphasized by Lin et al. [35], NAS with hardware co-design opens a new search space—hardware configurations—to deeply co-optimize the CNN architecture and its hardware implementation with the aim of delivering the most suitable trade-offs between the accuracy and hardware parameters. In addition to the CNN architecture and weights, the hardware configuration is optimized, which can involve optimizing the bit widths, quantization levels, PE array size, buffer size, MAC circuit configuration (e.g., by utilizing approximate multipliers), data flow organization, tiling strategy, loop order, memory subsystem parameters, preferences for the high-level synthesis tools, etc.

12.4.2.1 A Single Search Algorithm

A straightforward approach is to add the hardware parameters to the chromosome which describes the CNN architecture, and extend thus the search space of the original NAS algorithm.

For example, Pinos et al. [56] evolved CNN architecture together with the selection of suitable approximate multipliers for particular CNN layers to reduce power

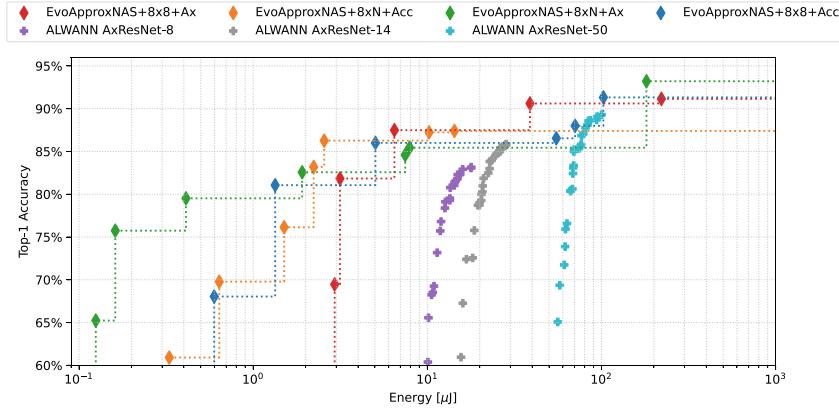


Fig. 12.8 The energy-accuracy trade-offs obtained by EvoApproxNAS on CIFAR-10 for four sets of multipliers that can be used by convolutional layers. Results are compared with various ResNet networks optimized with the ALWANN method [50]

consumption. The method, EvoApproxNAS, is based on CGP in which each node represents a network layer and a layer can use one of 35 multipliers. Figure 12.8 shows resulting Pareto fronts from four independent experiments in which EvoApproxNAS utilized four different sets of multipliers in convolutional layers: 8×8 -bit accurate (blue), $8 \times N$ -bit accurate (orange), 8×8 -bit approximate (red), and $8 \times N$ -bit approximate (green). By combining all these Pareto fronts, one can observe those approximate multipliers allow EvoApproxNAS to reach the best trade-offs between accuracy and energy of multiplication for almost all investigated regions of parameters.

12.4.2.2 Two Search Algorithms

The approach presented in the previous section leads to a time-consuming search process due to the prohibitively huge joint space composed of the coupled yet different CNN architecture and hardware configuration spaces with extremely sparse optima.

To reduce the search cost, the problem is often decoupled. Two search algorithms are now employed. Algorithm A_1 is a common NAS and works in the space of CNN architectures. Algorithm A_2 then performs the search in the space of hardware configurations. It can again be based on an EA; however, other search techniques have been utilized in the literature [63]. The search algorithms can interact in different ways, for example:

1. A_1 samples a CNN model α . No training of α is performed.
2. A_2 is executed to find the most suitable hardware configuration c_{hw} (satisfying all hardware constraints imposed by the specification) for α .

3. If no suitable hardware configuration is obtained, α is discarded, and step (1) is taken again.
4. If c_{hw} satisfies all constraints, then α is trained and then tested on the test data to get its accuracy $Acc(\alpha)$.
5. Steps (1) to (4) are repeated until a suitable solution $(\alpha, c_{hw}, Acc(\alpha))$ is not reached.

This approach is especially useful if finding a suitable hardware configuration for α takes significantly less time than the training of α . On the other hand, if a super net is employed, it is not necessary to train candidate CNN architectures (subnets), and a search in the hardware configuration space can be conducted for architectures showing acceptable accuracy. From the NAS-hardware co-design methods, surveyed by Sekanina [63], Table 12.5 lists those utilizing an evolutionary approach for A_1 (the *NAS Method* column) or A_2 (the *HW opt. method*). Selected hardware parameters that are optimized are listed in the *Design parameters* column. The remaining columns have the same meaning as in Table 12.4. Note that the use of EAs is relatively unexplored in this new area as documented by only three items in Table 12.5.

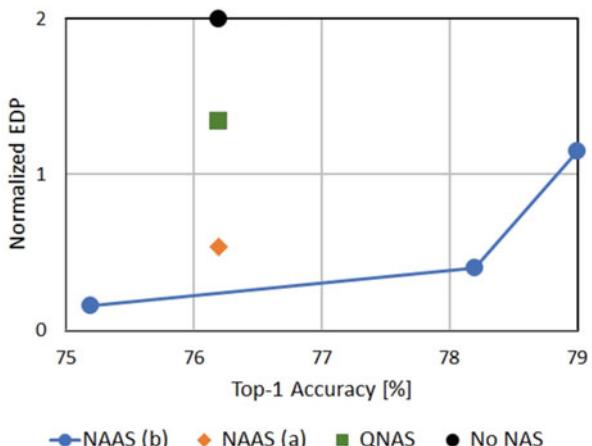
QNAs [34] focuses on optimizing the parameters of a mixed-precision systolic-array-like architecture (the array size, buffer input/weight/output size) while searching the quantized neural architecture. It includes an EA-based hardware architecture search and a one-shot supernet-based quantized neural architecture search. First, a suite of neural architectures is sampled as a benchmark to find the hardware architecture that achieves the best performance on the benchmark. The hardware architecture is fixed, and the quantized neural architecture search (QNAs) is then performed to determine the neural architecture and quantization policy. The quantized neural network is composed of multiple ResNet blocks.

NAAS [35] holistically searches the neural network architecture and accelerator architecture, and unlike other methods (e.g., [27]), compiler mapping. The accelerator search space is defined by the number of processing elements, local memory size, global buffer size, memory bandwidth, and connectivity parameters. NAAS employs EA to optimize these parameters as well as the compiler mapping (the execution

Table 12.5 Evolutionary NAS methods with hardware co-design

Method	Ref.	NAS method	Objective	HW opt. method	Design parameters	Target device	Data set
QNAs	[34]	EA, supernet	EDP	EA	#PE, mem. params	ASIC	ImgNet, C-10
NAAS	[35]	Gradient, supernet	EDP	EA	#PE, mem. params., compiler mapping	TPU, ASIC	ImgNet, C-10
Pinos et al.	[56]	EA	Energy	in NAS	Approximate multiplier type	ASIC	C-10, SVHN

Fig. 12.9 Normalized EDP and top-1 accuracy (on ImageNet) obtained by NAS methods for CNNs running on an ASIC [35]: NAAS co-optimizing HW, compiler mapping, and NN architecture (blue); NAAS co-optimizing HW and compiler mapping (orange); hardware search from QNAS (green); No NAS conducted (black). Adopted from [35]



order and the tiling size). It introduces a special encoding, called importance-based encoding, for the accelerator space and the compiler mapping strategies to avoid enumerating all possible situations and representing them by indexes. First, NAAS generates a pool of accelerator candidates. For each accelerator candidate, a network architecture is sampled from a pre-trained network [6] that satisfies the pre-defined accuracy requirement. Since each subnet is well-trained, the accuracy evaluation is fast. Finally, the compiler mapping strategy is sought for the network candidate on the corresponding accelerator candidate.

Figure 12.9 shows the impact of various approaches in optimizing the accuracy and Energy-Delay-Product (EDP) of ImageNet classifiers based on ResNet-50 and implemented on an Eyeriss-like chip. The original implementation (black point) of ResNet-50 (no NAS employed) is improved by a hardware search algorithm from QNAS [34] (green point). Additional improvement is provided by NAAS performing the hardware and compiler mapping co-search (orange point). The best trade-offs are reported for NAAS utilizing the hardware, compiler mapping, and CNN architecture co-search (blue points). These results (adopted from [35]) demonstrate that exploiting more design spaces can lead to better CNN implementations.

12.5 Conclusions and Open Challenges

We surveyed evolutionary approaches developed to optimize hardware implementations of CNNs and the NAS methods utilizing EAs. The optimization of various components and implementation principles of hardware accelerators with EAs seems to be a useful strategy because the relationships among all internal variables in complex systems such as hardware accelerators are highly nonlinear and corresponding search spaces are hard to explore.

Introducing the hardware search space in NAS algorithms has led to more efficient implementations of CNNs on particular hardware platforms. However, several search algorithms working in the space of weights, neural architectures, and hardware configurations have to be coordinated, making the entire method complicated. Successful adoption of EAs in these applications requires utilizing not only modern multi-objective evolutionary design and optimization methods but also state-of-the-art (surrogate) modeling and simulation techniques to get reliable information about the underlying hardware quickly.

In the following sub-sections, we outline the challenges that are critical for the successful development in this area.

12.5.1 Benchmarking and Reproducibility

As hardware-aware NAS methods are multi-objective, their fair assessment consists of evaluating multiple parameters of resulting implementations of CNNs and the design cost (time). It thus leads to an expensive construction and comparison of Pareto fronts in multidimensional spaces, which is often hard to perform because of incomplete information about some NAS methods. To support a fair benchmarking methodology and accelerate the development of new NAS methods, the open-source data sets containing many pre-trained and evaluated CNNs from well-defined search spaces were introduced in the literature, e.g., NAS-Bench-201 [16]. In addition to the accuracy for each design point in the search space, hardware parameters (such as latency and power) are also precomputed for some hardware accelerators [31]. Hence, new NAS algorithms can quickly be developed and evaluated for pre-defined search spaces. We see a lot of space for further opening the whole field to a broader community of researchers and practitioners by sharing NAS implementation source codes, data generated by NAS methods, data measured on real accelerators, and data obtained from simulations of various configurations of hardware accelerators. This effort should also improve the reproducibility of results in this area.

12.5.2 Security and Reliability

In addition to optimizing the quality of service, performance, and power consumption, other objectives must be considered when hardware-accelerated DNNs are deployed in real-world systems. DNN systems are highly vulnerable to security and reliability threats at both the cloud and the edge. Security attacks include inserting random or crafted noise into the data, inserting malicious components into the system hardware, polluting inputs with imperceptible noise during inference, and monitoring system-side channels to deduce the underlying model [64]. Reliability

issues include process variation during hardware fabrication, memory errors, and specific environmental conditions around the system that compromise reliability during training and inference. Shafique et al. [64] surveyed the threats and their respective countermeasures. One example of reliability-aware EA-based NAS—paper [62]—was discussed in Sect. 12.4. We expect a lot of research that could potentially utilize evolutionary algorithms in the areas of security and reliability of CNN accelerators.

12.5.3 Unconventional Hardware Platforms

Emerging technologies such as memristive crossbars or in-memory computing are investigated for CNN accelerators to reduce power consumption and other critical parameters [2, 66]. A very specialized simulator is usually developed to analyze the properties of these unconventional circuits and systems. The simulator can be connected with a NAS algorithm to find best-performing CNN-accelerator pairs. For example, PABO [54] uses NAS connected with a memristive crossbar-based CNN accelerator, where the CNN is mapped across the on-chip crossbar storage spatially. NAS4RRAM is a NAS method for optimizing CNNs and Resistive Random Access Memory (RRAM)-based accelerators [77]. NACIM [26] jointly explores device, circuit, and architecture design space and also takes device variation into account to find the most robust neural architectures, coupled with the most efficient hardware design for an in-memory computing ASIC. In the future, more exotic hardware platforms for CNNs could be introduced (e.g., similar to the nanoparticle networks configured using evolutionary algorithms for solving simple problems [5]) to provide richer and deeper interaction of machine learning and configurable physical *materio*.

12.5.4 Design Cost

The evolutionary NAS method is a computationally expensive approach requiring many core hours producing considerable CO_2 emissions. We expect many new approaches to reduce the computation cost in all directions, including efficient search algorithms, network training algorithms, hardware simulation, and benchmarking strategies.

Acknowledgements This work was supported by the Czech science foundation project *Automated design of hardware accelerators for resource-aware machine learning* under number 21-13001S.

References

1. Barone, S., Traiola, M., Barbareschi, M., Bosio, A.: Multi-objective application-driven approximate design method. *IEEE Access* **9**, 86975–86993 (2021)
2. Bavikadi, S., Dhaville, A., Ganguly, A., Haridass, A., Hendy, H., Merkel, C., Reddi, V.J., Sutradhar, P.R., Joseph, A., Pudukotai Dinakarao, S.M.: A survey on machine learning accelerators and evolutionary hardware platforms. *IEEE Design & Test* **39**(3), 91–116 (2022)
3. Benmeziane, H., El Maghraoui, K., Ouarnoughi, H., Niar, S., Wistuba, M., Wang, N.: Hardware-aware neural architecture search: survey and taxonomy. In: Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21, pp. 4322–4329. International Joint Conferences on Artificial Intelligence Organization (2021). Survey Track
4. Bingham, G., Macke, W., Miikkulainen, R.: Evolutionary optimization of deep learning activation functions. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference, GECCO '20, pp. 289–296. ACM (2020)
5. Bose, S.K., Lawrence, C.P., Liu, Z., Makarenko, K.S., van Damme, R.M.J., Broersma, H.J., van der Wiel, W.G.: Evolution of a designless nanoparticle network into reconfigurable boolean logic. *Nat. Nanotechnol.* **10**, 1048–1052 (2015)
6. Cai, H., Gan, C., Wang, T., Zhang, Z., Han, S.: Once-for-all: train one network and specialize it for efficient deployment. In: 8th International Conference on Learning Representations, ICLR. OpenReview.net (2020)
7. Cai, H., Zhu, L., Han, S.: Proxylessnas: direct neural architecture search on target task and hardware. In: 7th International Conference on Learning Representations, ICLR. OpenReview.net (2019)
8. Capra, M., Bussolino, B., Marchisio, A., Shafique, M., Masera, G., Martina, M.: An updated survey of efficient hardware architectures for accelerating deep convolutional neural networks. *Future Internet* **12**(7), 113 (2020)
9. Ceska, M., Matyas, J., Mrazek, V., Sekanina, L., Vasicesk, Z., Vojnar, T.: Sagtree: towards efficient mutation in evolutionary circuit approximation. *Swarm Evol. Comput.* **69**, 100986 (2022)
10. Chen, Y.-H., Krishna, T., Emer, J.S., Sze, V.: Eyeriss: an energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE J. Solid-State Circuits* **52**(1), 127–138 (2017)
11. Chen, Y., Meng, G., Zhang, Q., Zhang, X., Song, L., Xiang, S., Pan, C.: Joint neural architecture search and quantization (2018)
12. Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B., Shelhamer, E.: cuDNN: efficient primitives for deep learning (2014)
13. Colangelo, P., Segal, O., Speicher, A., Margala, M.: Artificial neural network and accelerator co-design using evolutionary algorithms. In: 2019 IEEE High Performance Extreme Computing Conference (HPEC), pp. 1–8 (2019)
14. Dai, X., Zhang, P., Wu, B., Yin, H., Sun, F., Wang, Y., Dukhan, M., Hu, Y., Wu, Y., Jia, Y., Vajda, P., Uyttendaele, M., Jha, N.K.: ChamNet: towards efficient network design through platform-aware model adaptation. In: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 11390–11399 (2019)
15. Deb, K.: *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley (2009)
16. Dong, X., Yang, Y.: NAS-Bench-201: extending the scope of reproducible neural architecture search. In: International Conference on Learning Representations (2020)
17. Dupuis, E., Novo, D., O'Connor, I., Bosio, A.: A heuristic exploration of retraining-free weight-sharing for CNN compression. In: 27th Asia and South Pacific Design Automation Conference, ASP-DAC, pp. 134–139. IEEE (2022)
18. Elsken, T., Metzen, J.H., Hutter, F.: Efficient multi-objective neural architecture search via Lamarckian evolution. In: 7th International Conference on Learning Representations, ICLR 2019. OpenReview.net (2019)

19. Fasfous, N., Vemparala, M.R., Frickenstein, A., Valpreda, E., Salihu, D., Höfer, J., Singh, A., Nagaraja, N.-S., Voegel, H.-J., Doan, N.A.V., Martina, M., Becker, J., Stechele, W.: AnaCoNGA: analytical HW-CNN co-design using nested genetic algorithms. In: 2022 Design, Automation & Test in Europe Conference & Exhibition (DATE), pp. 238–243 (2022)
20. Garofalo, A., Rusci, M., Conti, F., Rossi, D., Benini, L.: PULP-NN: a computing library for quantized neural network inference at the edge on RISC-V based parallel ultra low power clusters. In: 2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS), pp. 33–36 (2019)
21. Garofalo, A., Tagliavini, G., Conti, F., Rossi, D., Benini, L.: XpulpNN: accelerating quantized neural networks on RISC-V processors through ISA extensions. In: 2020 Design, Automation Test in Europe Conference Exhibition (DATE), pp. 186–191 (2020)
22. Guo, Z., Zhang, X., Mu, H., Heng, W., Liu, Z., Wei, Y., Sun, J.: Single path one-shot neural architecture search with uniform sampling (2019). [arXiv:abs/1904.00420](https://arxiv.org/abs/1904.00420)
23. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition (2015)
24. Huang, S.-Y., Chu, W.-T.: PONAS: progressive one-shot neural architecture search for very efficient deployment (2020). [arXiv:abs/2003.05112](https://arxiv.org/abs/2003.05112)
25. Intel. Intel-optimized math library for numerical computing (2021)
26. Jiang, W., Lou, Q., Yan, Z., Yang, L., Hu, J., Hu, X.S., Shi, Y.: Device-circuit-architecture co-exploration for computing-in-memory neural accelerators. *IEEE Trans. Comput.* **70**(4), 595–605 (2021)
27. Jiang, W., Yang, L., Sha, E.H.-M., Zhuge, Q., Gu, S., Dasgupta, S., Shi, Y., Hu, J.: Hardware/software co-exploration of neural architectures. *IEEE Trans. Comput.-Aided Design Integr. Circ. Syst.* **39**(12), 4805–4815 (2020)
28. Jouppi, N.P., Young, C., Patil, N., Patterson, D.: A domain-specific architecture for deep neural networks. *Commun. ACM* **61**(9), 50–59 (2018)
29. Kao, S.-C., Krishna, T.: Gamma: automating the hw mapping of dnn models on accelerators via genetic algorithm. In: Proceedings of the 39th International Conference on Computer-Aided Design, ICCAD ’20. ACM (2020)
30. Lapid, R., Sipper, M.: Evolution of activation functions for deep learning-based image classification. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO ’22, pp. 2113–2121. ACM (2022)
31. Li, C., Yu, Z., Fu, Y., Zhang, Y., Zhao, Y., You, H., Yu, Q., Wang, Y., Hao, C., Lin, Y.: HW-NAS-Bench: hardware-aware neural architecture search benchmark. In: 9th International Conference on Learning Representations, ICLR 2021. OpenReview.net (2021)
32. Liberis, E., Dudziak, L., Lane, N.D.: μ NAS: constrained neural architecture search for microcontrollers. In: EuroMLSys ’21, pp. 70–79. ACM (2021)
33. Lin, J., Chen, W.-M., Lin, Y., Cohn, J., Gan, C., Han, S.: MCUNet: tiny deep learning on iot devices. In: 34th Conference on Neural Information Processing Systems (NeurIPS 2020), pp. 1–12 (2020)
34. Lin, Y., Hafdi, D., Wang, H., Liu, Z., Han, S.: Neural-hardware architecture search. In: 33rd Conference on Neural Information Processing Systems (NeurIPS 2019) (2019)
35. Lin, Y., Yang, M., Han, S.: NAAS: neural accelerator architecture search. In: 2021 58th ACM/ESDA/IEEE Design Automation Conference (DAC) (2021)
36. Loni, M., Sinaei, S., Zoljodi, A., Daneshatalab, M., Sjödin, M.: DeepMaker: a multi-objective optimization framework for deep neural networks in embedded systems. *Microprocess. Microsyst.* **73**, 102989 (2020)
37. Lu, B., Yang, J., Jiang, W., Shi, Y., Ren, S.: One proxy device is enough for hardware-aware neural architecture search. *Proc. ACM Meas. Anal. Comput. Syst.* **5**(3) (2021)
38. Lu, Z., Deb, K., Goodman, E., Banzhaf, W., Boddeti, V.N.: NSGANetV2: evolutionary multi-objective surrogate-assisted neural architecture search. In: Computer Vision – ECCV 2020, pp. 35–51. Springer, Cham (2020)
39. Lu, Z., Whalen, I., Boddeti, V., Dhebar, Y., Deb, K., Goodman, E., Banzhaf, W.: NSGA-Net: neural architecture search using multi-objective genetic algorithm. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO ’19, pp. 419–427. ACM (2019)

40. Luo, X., Liu, D., Huai, S., Kong, H., Chen, H., Liu, W.: Designing efficient DNNs via hardware-aware neural architecture search and beyond. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **41**(6), 1799–1812 (2022)
41. Luo, X., Liu, D., Huai, S., Liu, W.: HSCoNAS: hardware-software co-design of efficient DNNs via neural architecture search. In: DATE 2021 (2021)
42. MAESTRO. An open-source infrastructure for modeling dataflows within deep learning accelerators (2021)
43. Marchisio, A., Massa, A., Mrazek, V., Bussolino, B., Martina, M., Shafique, M.: NASCaps: a framework for neural architecture search to optimize the accuracy and hardware efficiency of convolutional capsule networks. In: 2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD), pp. 1–9 (2020)
44. Mazumder, A.N., Meng, J., Rashid, H.-A., Kallakuri, U., Zhang, X., Seo, J.-S., Mohsenin, T.: A survey on the optimization of neural network accelerators for micro-ai on-device inference. *IEEE J. Emerg. Select. Topics Circ. Syst.* **11**(4), 532–547 (2021)
45. Mittal, S.: A survey of techniques for approximate computing. *ACM Comput. Surv.* **48**(4), 1–33 (2016)
46. Mittal, S.: A survey of FPGA-based accelerators for convolutional neural networks. *Neural Comput. Appl.* **32**(32), 1109–1139 (2020)
47. Mittal, S., Rajput, P., Subramoney, S.: A survey of deep learning on cpus: opportunities and co-optimizations. *IEEE Trans. Neural Netw. Learn. Syst.* **33**(10), 5095–5115 (2022)
48. Mrazek, V., Hrbacek, R., et al.: Evoapprox8b: library of approximate adders and multipliers for circuit design and benchmarking of approximation methods. In: Proceedings of DATE’17, pp. 258–261 (2017)
49. Mrazek, V., Sekanina, L., Vasicek, Z.: Libraries of approximate circuits: Automated design and application in CNN accelerators. *IEEE J. Emerg. Select. Topics Circuits Syst.* **10**(4), 406–418 (2020)
50. Mrazek, V., Vasicek, Z., Sekanina, L., Hanif, A.M., Shafique, M.: ALWANN: automatic layer-wise approximation of deep neural network accelerators without retraining. In: Proceedings of the IEEE/ACM International Conference on Computer-Aided Design, pp. 1–8. IEEE (2019)
51. Murshed, M.G.S., Murphy, C., Hou, D., Khan, N., Ananthanarayanan, G., Hussain, F.: Machine learning at the network edge: a survey. *ACM Comput. Surv.* **54**(8) (2021)
52. Nader, A., Azar, D.: Evolution of activation functions: an empirical investigation. *ACM Trans. Evol. Learn. Optim.* **1**(2) (2021)
53. Parashar, A., Raina, P., Shao, Y.S., Chen, Y.-H., Ying, V.A., Mukkara, A., Venkatesan, R., Khailany, B., Keckler, S.W., Emer, J.: Timeloop: a systematic approach to dnn accelerator evaluation. In: 2019 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), pp. 304–315 (2019)
54. Parsa, M., Ankit, A., Ziabari, A., Roy, K.: PABO: pseudo agent-based multi-objective bayesian hyperparameter optimization for efficient neural accelerator design. In: 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 1–8 (2019)
55. Pham, H., Guan, M.Y., Zoph, B., Le, Q.V., Dean, J.: Efficient neural architecture search via parameter sharing. In: Proceedings of the 35th International Conference on Machine Learning, ICML 2018, vol. 80, pp. 4092–4101. PMLR (2018)
56. Pinos, M., Mrazek, V., Sekanina, L.: Evolutionary approximation and neural architecture search. *Genet. Program Evolv. Mach.* **23**(3), 351–374 (2022)
57. Prabakaran, B.S., Akhtar, A., Rehman, S., Hasan, O., Shafique, M.: BioNetExplorer: architecture-space exploration of bio-signal processing deep neural networks for wearables. *IEEE Inter. Things J.* 1–10 (2021)
58. Prashanth, H.C., Madhav, R.: Evolutionary standard cell synthesis of unconventional designs. In: Proceedings of the Great Lakes Symposium on VLSI 2022, GLSVLSI ’22, pp. 189–192. ACM (2022)
59. Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y.L., Tan, J., Le, Q.V., Kurakin, A.: Large-scale evolution of image classifiers. In: Proceedings of the 34th International Conference on Machine Learning, ICML’17, vol. 70, pp. 2902–2911 (2017). JMLR.org

60. Sarwar, S.S., Venkataramani, S., Ankit, A., Raghunathan, A., Roy, K.: Energy-efficient neural computing with approximate multipliers. *J. Emerg. Technol. Comput. Syst.* **14**(2), 16:1–16:23 (2018)
61. Satesesan, A., Sinha, S., Smitha, K.G., Vinod, A.P.: A survey of algorithmic and hardware optimization techniques for vision convolutional neural networks on FPGAs. *Neural Process. Lett.* **53**(3), 2331–2377 (2021)
62. Schorn, C., Elsken, T., Vogel, S., Runge, A., Guntoro, A., Ascheid, G.: Automated design of error-resilient and hardware-efficient deep neural networks. *Neural Comput. Appl.* **32**(24), 18327–18345 (2020)
63. Sekanina, L.: Neural architecture search and hardware accelerator co-search: a survey. *IEEE Access* **9**, 151337–151362 (2021)
64. Shafique, M., Naseer, M., Theocharides, T., Kyrkou, C., Muthu, O., Orosa, L., Choi, J.: Robust machine learning systems: challenges, current trends, perspectives, and the road ahead. *IEEE Design & Test* **37**(2), 30–57 (2020)
65. Sipper, M.: Neural networks with à la carte selection of activation functions. *SN Comput. Sci.* **2**(6), 470 (2021)
66. Staudigl, F., Merchant, F., Leupers, R.: A survey of neuromorphic computing-in-memory: architectures, simulators, and security. *IEEE Design & Test* **39**(2), 90–99 (2022)
67. Stewart, R., Nowlan, A., Bacchus, P., Ducasse, Q., Komendantskaya, E.: Optimising hardware accelerated neural networks with quantisation and a knowledge distillation evolutionary algorithm. *Electronics* **10**(4) (2021)
68. Sze, V., Chen, Y., Yang, T., Emer, J.S.: Efficient processing of deep neural networks: a tutorial and survey. *Proc. IEEE* **105**(12), 2295–2329 (2017)
69. Sze, V., Chen, Y.-H., Yang, T.-J., Emer, J.S.: Efficient Processing of Deep Neural Networks. Synthesis Lectures on Computer Architecture. Morgan & Claypool Publishers (2020)
70. Vasicek, Z., Sekanina, L.: Evolutionary approach to approximate digital circuits design. *IEEE Trans. Evol. Comput.* **19**(3), 432–444 (2015)
71. Velasco-Montero, D., Fernandez-Berni, J., Carmona-Galan, R., Rodriguez-Vazquez, A.: Previous: a methodology for prediction of visual inference performance on IoT devices. *IEEE Internet Things J.* **7**(10), 9227–9240 (2020)
72. Venkataramani, S., et al.: Efficient AI system design with cross-layer approximate computing. *Proc. IEEE* **108**(12), 2232–2250 (2020)
73. Wang, T., Wang, K., Cai, H., Lin, J., Liu, Z., Wang, H., Lin, Y., Han, S.: APQ: joint search for network architecture, pruning and quantization policy. In: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2075–2084 (2020)
74. Wang, X., Wang, X., Jin, L., Lv, R., Dai, B., He, M., Lv, T.: Evolutionary algorithm-based and network architecture search-enabled multiobjective traffic classification. *IEEE Access* **9**, 52310–52325 (2021)
75. Wu, Y.N., Emer, J.S., Sze, V.: Accelergy: an architecture-level energy estimation methodology for accelerator designs. In: 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 1–8 (2019)
76. Xia, X., Ding, W.: HNAS: hierarchical neural architecture search on mobile devices (2020)
77. Yuan, Z., Liu, J., Li, X., Yan, L., Chen, H., Bingzhe, W., Yang, Y., Sun, G.: NAS4RRAM: neural network architecture search for inference on rram-based accelerators. *Sci. China Inf. Sci.* **64**, 160407 (2021)
78. Zhang, L.L., Yang, Y., Jiang, Y., Zhu, W., Liu, Y.: Fast hardware-aware neural architecture search. In: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), pp. 2959–2967 (2020)
79. Zhang, X., Wang, J., Zhu, C., Lin, Y., Xiong, J., Hwu, W.-M., Chen, D.: DNNBuilder: an automated tool for building high-performance DNN hardware accelerators for FPGAs. In: 2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), pp. 1–8 (2018)
80. Zhou, Y., Dong, X., Akin, B., Tan, M., Peng, D., Meng, T., Yazdanbakhsh, A., Huang, D., Narayanaswami, R., Laudon, J.: Rethinking co-design of neural architectures and hardware accelerators (2021)

Chapter 13

Adversarial Evolutionary Learning with Distributed Spatial Coevolution



Jamal Toutouh, Erik Hemberg, and Una-May O'Reilly

Abstract Adversarial Evolutionary Learning (AEL) is concerned with competing adversaries that are adapting over time. This competition can be defined as a minimization–maximization problem. Different methods exist to model the search for solutions to this problem, such as the Competitive Coevolutionary Algorithm, Multi-agent Reinforcement Learning, Adversarial Machine Learning, and Evolutionary Game Theory. This chapter introduces an overview of AEL. We focus on spatially distributed competitive coevolution for adversarial evolutionary learning to deal with the Generative Adversarial Networks (GANs) training challenges. A population of multiple individual solutions, parameterized artificial neural networks (ANN), provides diversity to the gradient-based GAN learning and increases the robustness of the GAN training. The computational complexity is reduced by using a spatial topology that decreases the number of evaluations and facilitates scalability. In addition, the topology enables diverse hyper-parameters, objectives, search operators, and data. We present a design and an implementation of an AEL system with spatial competitive coevolution and gradient-based adversarial learning. We demonstrate how the increase in diversity improves the performance of generative learning tasks on image data. Moreover, the distributed population in AEL can help overcome some hardware limitations for ANN architectures.

J. Toutouh (✉)
ITIS Software, University of Malaga, Malaga, Spain
e-mail: jamal@uma.es

E. Hemberg · U.-M. O'Reilly
MIT CSAIL, Cambridge, MA, USA
e-mail: hembergerik@csail.mit.edu

U.-M. O'Reilly
e-mail: unamay@csail.mit.edu

13.1 Introduction

Adversarial Evolutionary Learning (AEL) is concerned with competing adversaries that are adapting over time. One division of this perpetual conflict is to search for solutions to a minimization–maximization problem via competitive coevolutionary algorithm (CCA) [60], Multi-agent Reinforcement Learning (MARL) [23], Adversarial Machine Learning (AML) [109], and Evolutionary Game Theory (EGT) [102]. In AEL, adversarial populations (e.g., some solution representations or agents) adapt against the learning of each other. The *learning* refers to how the adversaries are updated, e.g., training of parameterized functions with some data. A population can help provide robustness and scale to the learning method [78]. One particular instance of interest of adaptive adversaries is in generative modeling with Generative Adversarial Networks (GANs) that aims to learn a function that describes a latent (unknown) distribution [51].

A GAN estimates the distribution from which real data samples from a training dataset are obtained [51]. GANs provide an example for the capabilities of AEL. In a GAN, two artificial neural networks (ANN), a generator and a discriminator, compete as adversaries over multiple iterations to address a minimax optimization. The generator produces data samples from a latent space input, and the discriminator (classifier) discriminates between generated samples (“fake”) and samples from a dataset (“real”). The objective and loss functions are defined to train the generator to synthesize a sample that cannot be discriminated from a real sample by the discriminator. GANs can produce realistic data samples with a few samples due to the generative adversarial interactions. Numerous image applications use GANs, e.g., 3D object generation [24], image-to-image translation [89], multispectral and panchromatic images fusion [41], and medical image generation [112] and data types [67, 83, 120].

A caveat is that GANs trained with gradient-based optimization methods commonly fail to converge to a stable solution where the generator and the discriminator cannot improve their objectives, i.e., to find the Nash equilibrium of the underlying game. Some examples of training misbehaviors, a.k.a. pathologies, that frequently hinder GAN training from converging to this equilibrium are mode collapse [9], discriminator collapse [64], and vanishing gradients [7]. There are black-box machine learning (ML) algorithms that use evolutionary computation [118] to overcome this when training GANs [33, 110].

We aim to show how diversity from AEL improves performance in adversarial evolutionary learning applied to train GANs. Furthermore, we inspect how convergence, resilience, and robustness are enabled by the diversity of: solutions (e.g., ANN weights), hyper-parameters, operators, objectives, and data. For this, we use the process of evolutionary learning (evolutionary computation) as a natural representation to accommodate diversity. In evolutionary learning, multiple solutions and higher performing solutions are selected and varied to create a population of new solutions, i.e., learning is done at the population level. The use of populations provides solution diversity. A spatial topology defines interaction among solutions and provides

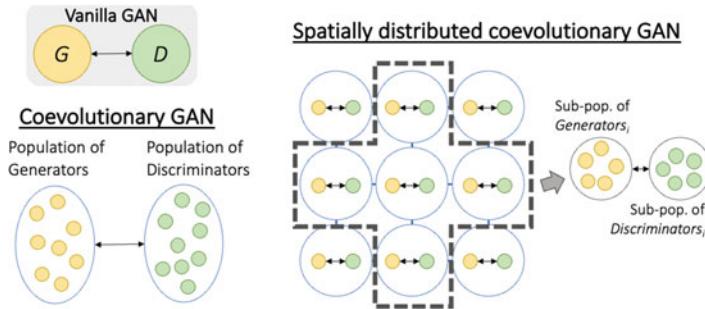


Fig. 13.1 Overview of possible sources of diversity for adversarial evolutionary learning with Lipizzaner, e.g., population

a diversity temporally over generations (training epochs). Furthermore, the nodes in the topology allow hyper-parameter, objective, data, and operator diversity for each node. Finally, the spatial topology provides an asynchronous and scalable system. Figure 13.1 shows possible sources of diversity for adversarial evolutionary learning, i.e., coevolution and spatially distributed coevolution.

The research work on spatial coevolutionary GANs is dispersed [2, 45, 54, 104–108], so this chapter consolidates it. A spatial (two-dimensional toroidal) topology can efficiently control the mixing of adversarial populations in coevolutionary algorithms [54, 108]. Studies showed that the spatially distributed competitive coevolutionary GAN training mitigates pathologies [54, 105]. For example, each cell contains a generator–discriminator pair and randomly selects a loss function to train each cell for each generation [104]. In addition, using distributed cells allows data decomposition to train GANs in each cell with different subsets of data for data diversity [106]. In particular, we demonstrate the impact of diversity on GAN training and different tasks [107]. In this chapter, we discuss studies of the following important adversarial evolutionary learning factors of population, hyper-parameter, variation operator, data, and objective diversity:

- **Spatial Population** Can spatial coevolutionary algorithms improve GAN training? (Sect. 13.4.1)
 - **Topology** What is the effect on the GAN trained when changing topology and communication? (Sect. 13.4.1.1)
 - **Scale** How do large spatial grids impact the accuracy of the generative model trained with a cellular algorithm? (Sect. 13.4.1.2)
- **Variation operator** Do different loss function combinations (applied to optimize the network parameters) result in generative models with better performance? (Sect. 13.4.2)
- **Data** How does the GAN training quality change in spatially distributed grids when the dataset is varied on the grid? (Sect. 13.4.3)

- **Task objective transfer** Can we create high-quality and diverse ensembles? (Sect. 13.4.4)

We focus on **Lipizzaner**, a system specifically designed to reliably address the degenerate GAN behaviors [54]. Lipizzaner combines spatially distributed coevolution and GAN training. Lipizzaner consists of an asynchronous CCA executing on a 2D spatial grid topology of cells with overlapping neighborhoods. Two competitive coevolving sub-populations exist on each cell—generators and discriminators—collected from the cell and its adjacent neighbors. Training is done pairwise in the sub-populations. The ANN models follow ML convention and are updated with stochastic gradient descent (SGD) using the minimax objective. Some training parameters, such as the learning rate of SGD, also use evolutionary learning. Between training epochs, the competing sub-populations are updated with copies of the best ANN models from the cell’s neighborhood. This communication relies upon the overlap of the cell neighborhoods. The solutions at each individual cell improve because of the training, while the spatial propagation of best solutions in each epoch prevents pathologies from persisting.

The contributions in this chapter are

1. An overview and some definitions of adversarial evolutionary learning, with focus on evolutionary GANs.
2. A comprehensive description of spatial coevolutionary GAN training, as an example of adversarial coevolutionary learning.
3. An extensive discussion and demonstration of adversarial evolutionary learning for GANs. We focus on the impact of population, hyper-parameters, objectives, data, and operator diversity.

The chapter proceeds as follows. Section 13.2 presents background and related work. Section 13.3 describes Lipizzaner AEL method and its variations. Section 13.4 summarizes different experimental analyses performed on adversarial coevolutionary learning. Finally, Sect. 13.5 draws the main conclusions and describes the future work.

13.2 Background and Related Work

First, we present an overview of the background and related work for AEL. Note that this chapter is an overview, not a full survey. Figure 13.2 shows a categorization that we follow for AEL. Our division is Competitive Coevolutionary Algorithm (CCA), Multi-agent Reinforcement Learning (MARL), Adversarial Machine Learning (AML), Evolutionary Game Theory (EGT), and Generative Adversarial Networks (GAN). We have included GANs in the figure because we understand that GANs can be seen as a specific case of AEL in which two agents (ANNs) compete with each other.

We base our AEL categories on the properties of: the number of defined adversaries, the dynamics of the adversaries, and the type of learning, see Table 13.1.

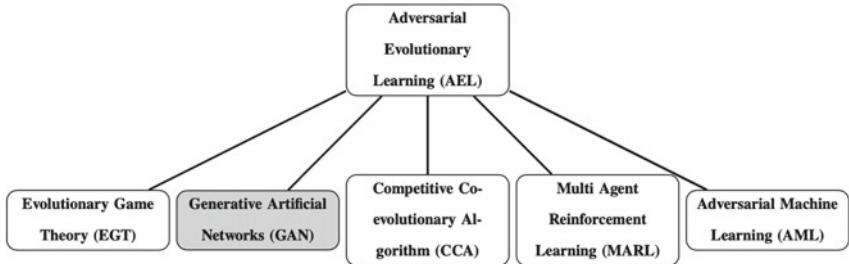


Fig. 13.2 A categorization of Adversarial Evolutionary Learning. To distinguish the categories we use the properties regarding number of adversaries, adversary dynamics, and “Learning”, see Table 13.1. We focus on GANs (gray box)

We categorize learning along the machine learning concepts of supervised (SL), unsupervised (UL), and reinforcement (RL). In addition, there are evolutionary learning process properties: population and selection and variation. Note that this is an overview, and all nuances are not presented.

In Table 13.1, we require that the *number of adversaries* is greater than zero. The dynamics describe that a static adversary is different from a *dynamic*, i.e., an adversary that changes (adapts) over time (between engagements, not only within the engagement). The dynamic adversary can change in different ways, e.g., adapting against each other. The *learning* refers to how the adversaries are updated, e.g., training parameterized functions. The population indicates if there is a population,

Table 13.1 Table of names and key properties in AEL categorization. In the Learning column: SL is supervised learning, UL is unsupervised learning, RL is reinforcement learning. The key evolutionary learning properties are population size, selection, and variation

Name	Adversarial learning			Evolutionary learning	
	#Adversaries	Dynamic	Learning	Population	Variation and selection
Generative adversarial network	1	Yes	SL, UL	No	No
Evolutionary game theory	≥ 1	No	No	∞	No
Competitive coevolutionary algorithm	≥ 1	Yes	RL	≥ 1	Yes
Multi-agent reinforcement learning	≥ 1	No	RL	No	No
Adversarial machine learning	1	No	SL, UL, RL	No	No

and if it is finite or infinite. Moreover, the use of variation and selection operators is indicated. Finally, hyper-parameter search variants of adversarial learning based on Evolutionary Computation (EC) variations are often created when the evolutionary process is added.

Next, we present the formal notation we use for adversarial evolutionary learning and, in particular, GANs.

13.2.1 Notation

We clarify some concepts of AEL with formal notation. We present an overview of learning in Sect. 13.2.1.1 and GANs in Sect. 13.2.1.2. Note we try to provide minimal and meaningful definitions with as simple notation as possible.

In general, input $\mathbf{x} \in \mathbb{R}$ can have a label $(\mathbf{x}, \mathbf{y}), \mathbf{y} \in \mathbb{R}$. Table 13.2 presents an overview of the notation used throughout the chapter. It includes parameters, variables, and constants considered names used for AEL and Lipizzaner [54]. Variables are optimized, parameters also include the initial variable values in Lipizzaner, and constants are used for the computational complexity analysis.

13.2.1.1 Learning Definitions and Notation

We define different types of learning as

Supervised Learning (SL) The input has labels, $\mathbf{y} \neq \emptyset$. Learn a parameterized function $f : \mathbb{R} \rightarrow \mathbb{R}, \hat{\mathbf{y}} = f(\mathbf{x}|\theta), \theta \in \mathbb{R}$. The parameters θ are learned by $h : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}, \theta = h(\mathbf{x}, \alpha), \alpha \in \mathbb{R}$. The objective is to minimize the difference between the model output and the actual value, $\text{argmin}_{\theta} \mathbf{y} - f(\mathbf{x}|\theta)$.

Unsupervised Learning (UL) The input does not have labels, $\mathbf{y} = \emptyset$. Learn a distance function $d : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}, \mathbf{y} = d(\mathbf{x}_0, \mathbf{x}_1|\theta), \theta \in \mathbb{R}$. The parameters θ are learned by $h : \mathbb{R} \times \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}, \theta = h(\mathbf{x}_0, \mathbf{x}_1, \alpha), \alpha \in \mathbb{R}$. The objective is to minimize the distance between similar input, $\text{argmin}_{\theta} d(\mathbf{x}_0, \mathbf{x}_1|\theta)$.

Reinforcement Learning (RL) An agent takes a sequence of actions that transition between states and collect a reward for their action sequence. Define environment states $s \in \mathbb{S}$, agent actions $a \in \mathbb{A}$, transition probabilities $p(s_{t+1} = s' | s_t = s, a_t, a) \in [0, 1], s' \in \mathbb{S}, a \in \mathbb{A}$, and rewards $f : \mathbb{S} \times \mathbb{S} \rightarrow \mathbb{R}, r = f(s, s')$. Objective is to learn a policy to maximize the reward $\pi : \mathbb{A} \times \mathbb{S} \rightarrow [0, 1], p(a_t = a | s_t = s) = \pi(a, s)$. Competitive Multi-agent reinforcement learning (MARL) can be expressed with an increase in state, action space for the multiple agents, and competing rewards for each agent.

Evolutionary Learning An evolutionary learning process selects and varies solutions over time based on the quality (fitness) when compared to other solutions in a population. Fitness is $\phi : \mathbb{R} \rightarrow \mathbb{R}$. The population of solutions at time t is $P_t = \phi(e(v(P_{t-1}))|\theta), P \in \mathbb{R}$. The variation operation is $v : \mathbb{R} \rightarrow \mathbb{R}, P_t = v(P_{t-1})$. The evaluation of the population is $e : \mathbb{R} \rightarrow \mathbb{R}, P_t = e(P_{t-1})$. The objective is to

Table 13.2 Parameters, variables, and constants considered in the chapter

Name	“Learning”
$f(\mathbf{x}, \theta)$	A parameterized function with parameters $\theta \in \mathbb{R}$
$h(\mathbf{x}, \alpha)$	A function for learning parameters with constants $\alpha \in \mathcal{N}$
\mathbf{r}	The red (opponent, test, attacker, predator) adversary, $\mathbf{r} \in \mathbb{R}$
\mathbf{b}	The blue (player, solution, defender, prey) adversary, $\mathbf{r} \in \mathbb{R}$
$\phi(\mathbf{x})$	The fitness of a solution, $\phi(\mathbf{x}) \in \mathbb{R}$
$v(P)$	The variation function of a population, $v(P) \in \mathbb{R}$
$e(P)$	The evaluation function of a population, $e(P) \in \mathbb{R}$
Name	Constants
C	GAN training cost, used for computational complexity analysis, $C \in \mathbb{Z}$
s_n	Maximum neighborhood size, $s_n = 5$ (a von Neumann neighborhood)
f_s^{\min}	Solution concept for selection and replacement that uses the minimum value (“best worst case”)
<i>Randomly Initialized and Evolved GAN Variables</i>	
δ	Learning rate of a neural network (generator/discriminator), a parameter, $\delta \in [0, \dots, 1]$
\mathbf{w}	Ensemble mixture weights of the generator ensemble mixture, a parameter, $\mathbf{w} \in [0, \dots, 1]^s$
<i>Randomly Initialized and SGD-Trained GAN Variables</i>	
g	Parameters of the generator, $g \in \mathbb{G}$.
d	Parameters of the discriminator, $d \in \mathbb{D}$.
<i>Configuration Parameters For Spatial Coevolution</i>	
m	The grid size, $m \in \mathbb{Z}$
s	Neighborhood size, derived from the cardinality of the neighborhood $s = \min(m^2, s_n)$
r	Radius (overlap) of neighborhood, $r \in \mathbb{Z}$
M	Loss functions, $M \in \mathcal{M}$
T	Number of generations (epochs), $T \in \mathbb{Z}$
τ	Tournament size, $\tau \in \mathbb{Z}$
β	Mutation probability for learning rate, $\beta \in \mathbb{R}$
μ	Mutation probability for mixture weights, $\mu \in \mathbb{R}$
G_g	A function parameterized by g , $G_g : \mathbb{R}^l \rightarrow \mathbb{R}^v$
D_d	A function parameterized by d , $D_d : \mathbb{R}^v \rightarrow [0, 1]$
\mathbf{X}	Dataset sample, $\mathbf{X} \in \mathcal{X}$
Θ	Parameters for Lipizzaner $\Theta = [m, s, M, T, \tau, \beta, \mu, \mathbf{w}, \delta, G_g, D_d, f_s^{\min}]$

(continued)

Table 13.2 (continued)

Name	“Learning”
<i>GAN Notation</i>	
φ	Concave measuring function, $\varphi : [0, 1] \rightarrow \mathbb{R}$
\mathcal{G}	Set of generators, $\{G_g, g \in \mathbb{G}\}, \mathbb{G} \subset \mathbb{R}^p$
\mathcal{D}	Set of discriminators, $\{D_d, d \in \mathbb{D}\}, \mathbb{D} \subset \mathbb{R}^p$
l	Dimension of Gaussian distribution that generates generator input, $l \in \mathbb{Z}$
z	Generator input dimension, $z \in \mathbb{Z}$
v	Generator output dimension, this is also the discriminator input dimension, $v \in \mathbb{Z}$
\mathcal{T}_{G_g}	Distribution defined by the generator, G_g
\mathcal{T}_*	Target distribution for the generator, G_g
e	Dataset size, $e = \mathbf{X} $
$\mathcal{L}(g, d)$	Gan training objective function, see Eq. (13.1). Used to calculate fitness for GAN coevolution, see Alg. 13.9
<i>Spatial Coevolution GAN Notation</i>	
ϕ	Fitness of a GAN, see Alg. 13.10, ϕ_g for generator and ϕ_d for discriminator
n	A cell in the grid, $n = [g, d, \delta, \mathbf{w}], n \in \mathbf{N}, \mathbf{N} = \{\mathbb{G} \times \mathbb{D} \times \mathbb{R} \times \mathbb{R}^s\}$
n_k	The k th cell, $1 \leq k \leq N$
\mathbf{n}	The neighborhood contains $[n_1, \dots, n_s], \mathbf{n} \in \mathbf{N}^s$, these are the sub-populations
\mathbf{n}_k	The k th neighborhood, $\mathbf{n}_k \in \mathbf{N}$
$g_{k,1}$	Generator at the center cell of the k th neighborhood, $g_k \in \mathbf{g}, 1 \leq k \leq N$
N	Number of cells, $N = m^2$ (also the population size)
\mathbf{N}	Grid population $\mathbf{N} = [\mathbf{n}_1, \dots, \mathbf{n}_N]$
\mathbf{g}	The sub-population of generators $\mathbf{g} = [g_1, \dots, g_s]$ in a neighborhood
\mathbf{d}	The sub-population discriminators $\mathbf{d} = [d_1, \dots, d_s]$ in a neighborhood
F	Lipizzaner function, see Eq. (13.2), $F : \mathcal{X} \times \mathbb{G}^s \times \mathbb{R} \times \mathbb{R}^s \rightarrow \mathbb{G}^s \times \mathbb{R}^s$
h	Generator measuring function, $h : \mathbb{G}^s \times \mathbb{R}^s \rightarrow \mathbb{R}, q = h(\mathbf{g}, \mathbf{w})$
q	Generator ensemble quality score $q = h(\mathbf{g}, \mathbf{w})$. Fitness for weight evolution, see Alg. 13.8
\mathbf{y}	Variables optimized by Lipizzaner, $\mathbf{y} = [\mathbf{g}, \delta, \mathbf{w}], \mathbf{y} \in \mathcal{Y}, \mathcal{Y} = \{\mathbb{G}^s \times \mathbb{R} \times \mathbb{R}^s\}$
\mathbf{e}	Ensemble of generators, $\mathbf{e} = [\mathbf{g}, \mathbf{w}], \mathbf{e} \in \mathcal{E}, \mathcal{E} = \{\mathbb{G}^s \times \mathbb{R}^s\}$
\mathbf{e}^*	Best ensemble of generators, $\mathbf{e} = [\mathbf{g}, \mathbf{w}]$, see Eq. (13.5)
θ_p	Dataset sampling fraction $\theta_p \in [0, 1]$

maximize fitness $\text{argmax}_{\mathbf{x} \in P} \phi(e(v(P)|\theta))$. Figure 13.3 shows a high-level EC process for learning. This is the basis for the Adversarial Evolutionary Learning (AEL).

Adversarial Learning (AL) Learning with two explicit competing adversaries (player/opponent, test/solution, red/blue, attacker/defender, predator/prey, generator/discriminator) $\mathbf{r}, \mathbf{b} \in \mathbb{R}$. The engagement of the adversaries is $f : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}, \mathbf{y} = f(\mathbf{r}, \mathbf{b}|\theta_r, \theta_b), \theta \in \mathbb{R}$. The parameters θ_r, θ_b are learned by $h : \mathbb{R} \times \mathbb{R} \rightarrow$

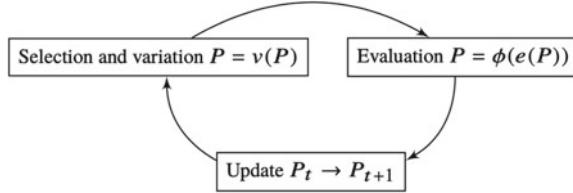


Fig. 13.3 The iterative process used for evolutionary computation (EC). At iteration t , a set of individual (sample) is selected from a population P_t and varied. These individuals are then evaluated. Individual performance is determined. The population is then updated toward the better performing individuals. The process then begins again with the new population P_{t+1}

$\mathbb{R} \times \mathbb{R}, \theta_r, \theta_b = h(\mathbf{r}, \mathbf{b}, C), C \in \mathbb{R}$. The objective is for one adversary to minimize and the other to maximize $\min_{\mathbf{r}} \max_{\mathbf{b}} f(\mathbf{r}, \mathbf{b}|\theta)$. Note, AL often use $\theta_r = \theta_b$.

One minimax-instance of AL is GAN. We focus this chapter on how AEL can help improve a GAN. Next, we provide a more extensive description of GAN training.

13.2.1.2 GAN Notation

We adopt notation similar to [8, 54, 65], see Table 13.2. We first introduce GAN notation, then describe some GAN learning tasks, how to use a topology, and GAN training details.

Let $\mathcal{G} = \{G_g, g \in \mathbb{G}\}$, $\mathbb{G} \subset \mathbb{R}^p$ and $\mathcal{D} = \{D_d, d \in \mathbb{D}\}$, $\mathbb{D} \subset \mathbb{R}^p$ denote sets of generators and discriminators. G_g is a function $G_g : \mathbb{R}^l \rightarrow \mathbb{R}^v$ parameterized by g . D_d is a function $D_d : \mathbb{R}^v \rightarrow [0, 1]$ parameterized by d . The generator G_g defines a distribution \mathcal{T}_{G_g} , this is done by generating z from an l -dimensional Gaussian distribution and then applying G_g on z to generate a sample $x = G_g(z)$ of the distribution G_g . Finally, let \mathcal{T}_* be the unknown target distribution to which we would like to fit our generative model G_g .

Formally, the goal of GAN training is to find parameters g and d in order to optimize the objective function

$$\min_{g \in \mathbb{G}} \max_{d \in \mathbb{D}} \mathcal{L}(g, d) , \text{ where}$$

$$\mathcal{L}(g, d) = \mathbb{E}_{x \sim \mathcal{T}_*} [\varphi(D_d(x))] + \mathbb{E}_{x \sim \mathcal{T}_{G_g}} [\varphi(1 - D_d(x))] , \quad (13.1)$$

and $\varphi : [0, 1] \rightarrow \mathbb{R}$ is a concave *measuring function*. In practice, we have access to a finite number of training samples x_1, \dots, x_v to approximate \mathcal{D}_{G_g} or G_g . Therefore, an empirical version $\frac{1}{v} \sum_{i=1}^v \varphi(D_d(x_i))$ is used to estimate $\mathbb{E}_{x \sim \mathcal{T}_*} [\varphi(D_d(x))]$.

13.2.1.3 GAN Learning Tasks

This section introduces the main adversarial learning tasks that are approached with GANs, i.e., generative modeling and ensemble repurposing.

For the task of generative modeling one way of expressing Lipizzaner is as a function F that returns an ensemble of generator parameters and their ensemble weights (see Eq. 13.2). The variables that are optimized are $\mathbf{y} = [\mathbf{g}, \delta, \mathbf{w}], \mathbf{y} \in \mathcal{Y}, \mathcal{Y} = \{\mathbb{G}^s \times \mathbb{R} \times \mathbb{R}^s\}$, the ensemble size is $s, s \in \mathbb{N}$, the parameters of each generator G_g in the ensemble $\mathbf{g} = [G_{g_1}, \dots, G_{g_s}]$, the learning rate for neural network training $\delta, \delta \in \mathbb{R}$ and the ensemble mixture weights $\mathbf{w}, \mathbf{w} \in \mathbb{R}^s$, and \mathbf{X} is a dataset (a sample from the target distribution \mathcal{T}_*). We focus on finding the best ensemble of generators $\mathbf{e} = [\mathbf{g}, \mathbf{w}], \mathbf{e} \in \mathcal{E}, \mathcal{E} = \{\mathbb{G}^s \times \mathbb{R}^s\}$ according to a measuring function $h : \mathbb{G}^s \times \mathbb{R}^s \rightarrow \mathbb{R}$. This can be stated as it is described in Eq. 13.3 by using the GAN training Eq.(13.1) and Θ is the set of parameters $\Theta = \{m, s, M, T, \tau, \beta, \mu, \mathbf{w}, \delta, G_g, D_d, f_s^{\min}\}$, shown in Table 13.2.

$$\mathbf{e} = F(\mathbf{X}, \mathbf{y}), F : \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{E} \quad (13.2)$$

$$\operatorname{argmax}_{\mathbf{y} \in \mathcal{Y}} h(F(\mathbf{X}, \mathbf{y} | \Theta)) \quad (13.3)$$

by using the GAN training Eq.(13.1) and Θ is the set of parameters $\Theta = \{m, s, M, T, \tau, \beta, \mu, \mathbf{w}, \delta, G_g, D_d, f_s^{\min}\}$, shown in Table 13.2.

We present the task of ensemble repurposing as follows. Given a set of generators \mathbf{g} find the weight vector $\mathbf{w}, |\mathbf{g}| = |\mathbf{w}|$. For the set of generators \mathbf{g} , the optimal ensemble mixture weight vector \mathbf{w} is defined as follows:

$$\mathbf{w}^* = \operatorname{argmax}_{\mathbf{w}} \sum_{i=1}^{|\mathbf{g}|} h((g_i, w_i)), \quad (13.4)$$

where w_i represents the probability that a data point comes from the i th generator, with $\sum_{i=1}^{|\mathbf{g}|} w_i = 1$.

13.2.1.4 Topology

In this chapter, we focus on a spatially distributed CCA applied to train GANs, i.e., Lipizzaner. Lipizzaner uses a square toroidal grid with N cells, $N = m \times m$. A cell $n, n = [g, d, \delta, \mathbf{w}], n \in \mathbb{N}, \mathbb{N} = \{\mathbb{R} \times \mathbb{R}^s \times \mathbb{G} \times \mathbb{D}\}$ has a *neighborhood* $\mathbf{n} = \{n_1, \dots, n_s\}, n_i \in \mathbb{N}\}$ which allows it to form two sub-populations of models: \mathbf{g} and \mathbf{d} . We denote the size of this neighborhood by s and the number of neighborhoods is N . We use a default five-cell von Neumann neighborhood ($s = 5$), with radius (overlap) $r = 1$, see Fig. 13.4a.

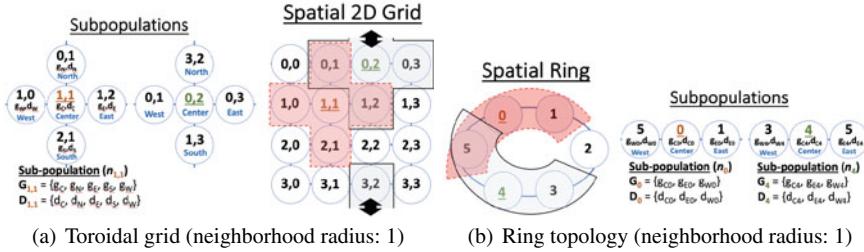


Fig. 13.4 Overlapping neighborhoods and sub-population definitions according to topologies and neighborhood radius

For the k th neighborhood in the grid \mathbf{n}_k , $1 \leq k \leq N$, we refer to the generator in its center cell by $g_{k,1}$ and the generators in the rest of the neighborhood by $g_{k,2}, \dots, g_{k,s}$, respectively. Furthermore, we denote the union of these generators for the k th generator neighborhood by $\mathbf{g}_k = \cup_{i=1}^s \{\mathbf{g}_{k,i}\} \subseteq \{g_{1,1}, \dots, g_{N,1}\}$.

In Lipizzaner, the generators of the N sub-populations each of size s to form N ensembles, the best ensemble is returned by Lipizzaner. For the ensembles of generators \mathbf{g} the s -dimensional ensemble mixture weight vector \mathbf{w} is defined as follows:

$$\mathbf{e}^* = \operatorname{argmax}_{\mathbf{g}_k, \mathbf{w}_k : 1 \leq k \leq N} \sum_{i=1}^s h(g_{k,i}, w_{k,i}). \quad (13.5)$$

13.2.1.5 GAN Training

Multiple loss functions ϕ have been defined for addressing the underlying optimization problem behind GAN training for p_M . The ones we are concerned with are (written for generators): *Binary cross entropy (BCE) loss* where the model's objective is to minimize the Jensen–Shannon divergence (JSD) between the real and fake data distributions *Mean squared error (MSE) loss* where the model's objective is to minimize the average of the squares of the errors, e.g., [74]. *Heuristic loss (HEU)* objective maximizes the probability of the discriminator being mistaken by minimizing the objective function [110] *Wasserstein (WASS) loss* Wasserstein GANs [7] minimize the *Wasserstein Distance* between the two probability distributions [7].

The challenge of training a GAN is increased by some pathologies:

Non-Convergence: The model parameters oscillate, destabilize, and never converge. Proposed remedies are regularization of the discriminator, e.g., add noise to discriminator inputs and penalize discriminator weights.

Mode Collapse: The generator collapses which produces limited varieties of samples. Proposed remedies are unrolled GANs and Wasserstein loss.

Vanishing gradient: The discriminator gets too successful that the generator gradient vanishes and learns nothing. Proposed remedies are Wasserstein loss and modified minimax loss.

Next, we present an overview of adversarial evolutionary learning.

13.2.2 *Adversarial Evolutionary Learning*

In this section, we discuss different AEL aspects, with focus on CCA. We briefly introduce adversarial learning with some evolutionary learning examples, and then we present CCAs.

13.2.2.1 *Adversarial Learning*

There are multiple subfields that have studied learning adversaries. We briefly present them here:

Evolutionary Game Theory (EGT) started as an application of the mathematical theory of games to biological contexts based on the notion that frequency-dependent fitness introduces a strategic aspect to evolution. Game theory is a formal theory of decisions of intelligent rational agents in conflicting situations where the decisions depend on each other. In evolutionary game theory, there is a very large population that are randomly paired to play a given strategy [102]. The interaction of multiple autonomous agents gives rise to highly dynamic and non-deterministic environments in the evolutionary dynamics of multi-agent learning [17]. As an example a combination of elements from deep learning and artificial life demonstrated that coevolutionary neural population models can simulate population dynamics, and that evolutionary game theory can describe the behavior [80].

Multi-agent Reinforcement Learning (MARL) is a subfield of reinforcement learning that focuses on studying the behavior of multiple learning agents that coexist in a shared environment [23]. Each agent has its own rewards and takes actions accordingly. Competing agent interests can result in non-trivial group dynamics. MARL is related to game theory. A comprehensive survey of state-of-the-art methods for integrating EC into RL [12].

Adversarial Machine Learning (AML) study attacks and defenses on machine learning algorithms [16, 49, 94, 109]. For example, artificial neural networks can produce high confidence predictions for unrecognizable images [82]. In another example is generating evasive malware [4, 40, 76, 113].

13.2.2.2 *Competitive Coevolutionary Algorithm*

An Evolutionary Algorithm (EA) can evolve individuals, e.g., fixed length genotypes such as the bit strings used by Genetic Algorithms (GAs) [50]. Biological coevolution

refers to the influence of multiple species on each other's evolution [42, 90]. We are concerned with competitive coevolution, e.g., due to constrained resources under contention or a predator-prey relationship. EAs usually abstract the evolutionary process and evaluate the quality of an individual with an a priori defined fitness function. Coevolutionary algorithms extend EAs and base the fitness of an individual on its interactions with other individuals or the environment. This mimics the coupled interaction of natural coevolution [77, 88, 90, 97].

A basic competitive coevolutionary algorithm evolves two coupled populations, each with selection and variation (e.g., crossover and mutation). In this section, we refer to individuals from the two *adversarial* populations as *adversaries*. Two types of adversaries perform the competitive coevolution: *tests* and *solutions*. In each generation, competitions (engagements) are formed by pairing adversaries from their respective populations. This shared fitness evaluation couples the population. The objective of a test individual is to demonstrate the solution as incorrect. Conversely, the objective of the solution individual is to solve the test correctly. The dynamic of the competitive coevolutionary algorithm, driven by conflicting objectives and guided by performance-based selection and random variation, can gradually produce better and more robust solutions [90, 97]. Generally, competitive coevolutionary algorithms suit interactive domains where performance is relative to others, e.g., games [88]. In addition, there is also some initial runtime analysis of a competitive coevolutionary algorithm named Pairwise Dominance CoEvolutionary Algorithm (PDCoEA) [63] with level-based analysis [30].

An efficient competitive coevolutionary algorithm minimizes the number of competitions per generation while maximizing the accuracy of its fitness estimate of each adversary. Two competition structures are: *one-vs-one*, each adversary competes only once against a member of the opposing population, see Fig. 13.5a, and *all-vs-all*, each adversary competes against all members of the opposing population, see Fig. 13.5b. *One-vs-one* has minimal fitness evaluations¹ ($O(N)$) and strong competition bias. In contrast, *all-vs-all* has a quadratic number of fitness evaluations, thus a higher computational cost, $O(N^2)$ but reduced competition bias [97]. In [79], adversaries are placed on a $m \times m$ grid with a fixed neighborhood (size s). The structure is competition among all competitors in the neighborhood. Fitness evaluations are reduced to $O(ms^2)$ by this. An individual's fitness is calculated over all engagements. At each generation, an individual is assigned a fitness score derived from some function of its performance in the competitions. The fitness assignment is a *solution concept* [88]. Solution concepts include best worst case, maximization of expected utility, Nash equilibrium, and Pareto optimality.

An example of an alternating competitive coevolutionary algorithm [6] is shown in Algorithm 1 (Fig. 13.6). The adversaries evolve in an alternating manner. First, the adversaries are initialized. Then, at each generation, a new test population, \mathbf{T}_t , is selected, mutated, recombined, and evaluated against the current solution population, \mathbf{S}_{t-1} . Based on the evaluation the tests are replaced. Then, control reverts to the solution population so it can evolve.

¹ Computational cost is shown for two populations of size N .

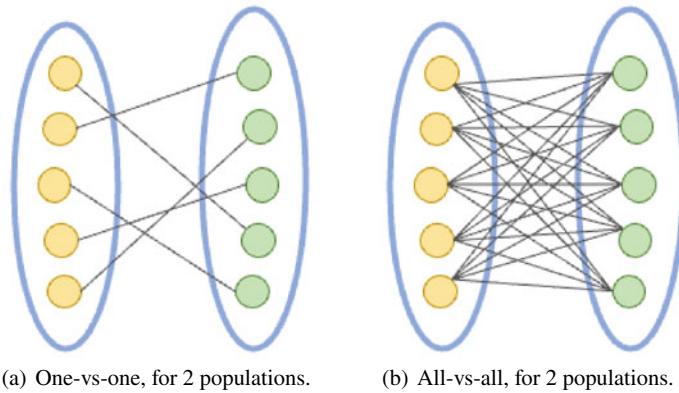


Fig. 13.5 Competition structures between adversaries/competitors

Algorithm 1: Example of alternating CompCA

```

Input : I: number of generations  $\mathcal{L}$ : Fitness function,  $\mu$ : Mutation probability, N: Population size
Return:  $a_*$ : Best tests,  $d_*$ : Best solutions
1  $T_0 \leftarrow [a_{1,0}, \dots, a_{N,0}] \sim \mathcal{U}(\{0,1\})$ //Initialize test population
2  $S_0 \leftarrow [d_{1,0}, \dots, d_{N,0}] \sim \mathcal{U}(\{0,1\})$ //Initialize solution population
3  $t \leftarrow 0$ //Initialize iteration counter
4 while  $t \leq I$  do
    //Iterate over populations
    for  $X \in (T_t, S_t)$  do
         $Y \leftarrow adversary(X)$ //Get the adversary population
         $t \leftarrow t + 1$ //Increase counter
         $X_t \leftarrow select(X_{t-1})$ //Selection
         $X_t \leftarrow mutate(X_t, \mu)$ //Mutation
        if  $X == T$  then
            //Test is the evolving population.
             $a'_*, d'_* \leftarrow \underset{a \in X_t}{\text{argmin}} \underset{d \in Y_{t-1}}{\text{argmax}} \mathcal{L}(a, d)$ //Best individual
        else
            //Solution is the evolving population
             $a'_*, d'_* \leftarrow \underset{a \in Y_t}{\text{argmin}} \underset{d \in X_{t-1}}{\text{argmax}} \mathcal{L}(a, d)$ //Best individual
        if  $\mathcal{L}(a'_*, d'_*) < \mathcal{L}(a_{N,t-1}, d_{N,t-1})$  then
            //Replace worst test
             $a_{N,t-1} \leftarrow a'_*$ //Update population
         $X_t \leftarrow X_{t-1}$ //Copy population
    17  $a_*, d_* \leftarrow \underset{a \in T_I}{\text{argmin}} \underset{d \in S_I}{\text{argmax}} \mathcal{L}(a, d)$ //Best individuals
18 return  $a_*, d_*$ 

```

Fig. 13.6 Algorithm 1

The interactive aspect of fitness implies that there is no *exact* fitness measure for a competitive coevolutionary algorithm, but the fitness value depends on the adversary. In coevolutionary algorithms two members of the same population, during selection, can be imprecisely compared when they did not compete against the same opponents. In addition, regardless of the function that computes a fitness score, an adversary's score may change when it competes against different opponents. These properties imply that the ranking individuals is a noisy estimate. This estimation can lead to

competitive coevolutionary algorithms *pathologies* that limit the effectiveness when modeling an arms race [88]. Furthermore, pathologies can arise from competitive imbalance, particularly when the behavior space of one population is not the same as that of the other [84]. competitive coevolutionary algorithm pathologies include

disengagement—occurring when opponents are not challenging enough or too challenging [25],

cyclic dynamics—generally appearing in transitive domains (A beats B, B beats C, and C beats A) [57],

focusing or overspecialization—arising when an adversary evolves to beat only some of its opponents [22], and

coevolutionary forgetting—occurs when the ability to beat an opponent evolves away [20].

In order to remediate these pathologies proposed methods center on archives/memory, maintaining sub-optimal individuals in the population, using a neighborhood spatial structure, and/or monitoring the progress of the algorithm [18, 20, 22, 25, 44, 57, 60, 115].

13.2.3 Related Work

This section first provides an overview of some of the applications of AEL. Then, it focuses on different variations of EC-based GANs and their applications.

AEL in the form of CCA has been used for search, optimization, design, and modeling problems [77, 88, 90, 97]. Some example domains are

- *Games*, encompassing board games [87], video game playing [58, 98], and social science games [10].
- *Search-based software engineering* [1, 3, 39]. For example, there has been use of competitive coevolutionary algorithms to coevolve programs and unit tests from their specification [5, 6, 114]. Another study used coevolution to distinguish correct behavior from incorrect [15].
- *Security*, in particular, cybersecurity [56, 59, 75, 85, 93, 96]. Other examples are vulnerability testing for malware in mobile applications using coevolution appears in [21]. More examples are [47, 55, 99].

Regarding AEL applied to GAN training, Evolutionary GAN (EGAN) uses multiple loss functions, each associated with a generator and evolves a population of generators against one discriminator by selecting the best generator [110]. EGANs introduce diversity through variation with mutation. Another example is the COEGAN [31–33] approach which evolves the neural network architectures for Generator and Discriminator and then uses coevolution to train the adversarial components of GANs. The t-SNE visualization showed that COEGAN evolves generators and discriminators toward higher quality models [36]. Furthermore, COEGAN was extended by using skill rating, a game-based fitness function, to assess the individuals [35].

In addition, different variants of EC have also been used. For example, the expression-based evolutionary GAN was explored. The main idea was to replace the generator with a population of expressions and evolve them by using GP [11]. Other works extend EGAN to multi-objective evolutionary GAN based on NSGA-II [14]. Another multi-objective approach is using quality-diversity search with COEGAN based on Novelty Search with Local Competition (NSLC) [34]. Furthermore, differential evolution has been used with GANs (DEGAN) to refine the input of the generator of a Wasserstein GAN [121]. Finally, a cooperative coevolutionary algorithm has been proposed to conduct adversarial multi-objective optimization by incorporating dual evolution with respect to the generators and discriminators [27].

There are many investigations of using and improving EC for GANs. For example, the application of GA to evolve fake samples to make them more realistic to train the discriminator [28]. A variation of COEGAN, named COEGAN-v2, which applies new variation operators by including spectral normalization and upsampling layers and has RaSGAN loss function for training and fitness evaluation [37]. An extension of EGAN, which introduces a new metric computed by the discriminator to assess the realness of a sample used to mutate the individuals [81]. Another variation of EGAN, which uses ES to evolve a population of generators that apply an SGD-based mutation operator [68]. EvolGAN [92], which does not modify the training phase, instead evolutionary latent space search based on a quality estimator metric is applied to improve the quality of the produced images when the generator is trained on small, difficult datasets, or both. And AEGAN, which is an extension of EGAN that embeds a normalized self-attention mechanism in the discriminator and generator to adaptively assign weights according to the importance of the samples' features [116]. Other examples are based on partial transfer learning for improving the speed [71], crossover-based knowledge distillation [66], and both knowledge distillation and transfer learning [72].

Regarding the use of evolutionary GANs, a common domain of application is computer vision (i.e., dealing with images). There has been work on evolutionary generative adversarial networks for hyperspectral image classification [13], abnormal electrocardiogram classification [111], data augmentation for cardiac magnetic resonance image [46], and COVID-19 infection segmentation [53, 103]. More general image applications are face image inpainting with evolutionary generators [52], and evolving noise injection in super-resolution GANs [91]. Furthermore, there have also been work with evolutionary GANs in other domains. For example, composite evolutionary GAN for natural language generation [100]. Another example is category-aware GAN with hierarchical evolutionary learning for text generation [70]. Evolving GANs have also been used for sequential data imputation [26]. Finally, EAs have been applied to GAN to evolve the architectures of the generators and discriminators [38, 48, 117].

None of the AEL methods previously discussed have studied spatial coevolution. Here, we focus on the impact of diversity, at population, hyper-parameters, operators and data and objectives, that is provided by this technique. In the next section, we describe the spatial coevolutionary GAN training method applied by Lipizzaner.

13.3 Method

In this section, we consolidate the description of AEL with spatial coevolution applying the Lipizzaner framework and its variants. Lipizzaner is used for GAN training and promotes diversity with a population, topology, operators, data and objectives. The key method components for Lipizzaner are

Gradient-based learning to update the neural networks' parameters of the generator g , discriminator d , and learning rate δ .

Gaussian-based mutations to update the learning rate and weight ensemble parameters δ, \mathbf{w} .

Evolutionary selection and replacement to maintain convergence.

A topology with overlapping cells n_i between neighborhoods $\mathbf{n}_i, \mathbf{n}_j$, with sub-populations of generators \mathbf{g} and discriminators \mathbf{d} in the neighborhoods.

The spatial topology for GAN training is described in Sect. 13.3.1. The algorithmic description of spatially distributed GAN training is in Sect. 13.3.2. Finally, the implementation of spatially distributed GAN training with Lipizzaner is described in Sect. 13.3.3.

13.3.1 Spatially Distributed GAN Training

Lipizzaner uses von Neumann neighborhoods with radius (overlap) 1, see Sect. 13.2.1.4. This defines the migration policy (i.e., the directionality of signal propagation) through the cells in four directions. Figure 13.4a shows an example of a 2D toroidal grid with $Z = 16$ (4×4 grid). The shaded areas illustrate the overlapping neighborhoods of the $(1, 1)$ and $(0, 2)$ cells, with dotted and solid outlines. The updates in the *center* of $(1, 1)$ will be propagated to the $(0, 1), (2, 1), (1, 0)$, and $(1, 2)$ cells.

In the ring topology, the cells are distributed in a one-dimensional grid of size $1 \times Z$ and neighbors are sideways, e.g., left and/or right, i.e., an index position or more away. The best GAN (*center*) after an evolutionary epoch at a cell is updated. Neighborhood cells retrieve this update when they refresh their sub-population at the end of their training epochs, carrying signals in two directions around the ring. Figure 13.4b shows populations of six individuals ($Z = 6$) organized in a ring topology with neighborhood radius one ($r = 1$). The shaded areas illustrate the overlapping neighborhoods of the cells (0) and (4) , with dotted and solid outlines, respectively. The updates in the *center* of (0) will be propagated to the (5) and (1) . Next, we provide a more detailed algorithmic description of the AEL Lipizzaner method.

13.3.2 Algorithmic Description of *Lipizzaner*

Lipizzaner starts with Algorithm 2 (Fig. 13.7)::*Lipizzaner* which runs in parallel on each cell n . Each cell neighborhood \mathbf{n} randomly gets a dataset $\mathbf{s} \subseteq \Theta_D$ and initializes single discriminator d and generator g neural network models (we refer to these as models when discriminators and generators are interchangeable). Thus there are two global adversarial populations of the same size as the topology N , $[g_1, \dots, g_N]$ a population of generators and $[d_1, \dots, d_N]$ a population of discriminators. These populations are a source of diversity in *Lipizzaner*, occurring in the representation space, genome space. Each cell n also initializes its evolvable parameters—learning rate δ and mixture weights \mathbf{w} .

Lipizzaner at the highest level searches for and returns a mixture of generators \mathbf{e} composed from a neighborhood \mathbf{n} . Algorithm 3 (Fig. 13.8)::*MixtureEA* evolves a mixture weight vector \mathbf{w} for each neighborhood \mathbf{n} using an ES-(1+1) algorithm which optimizes for generator ensemble performance q , e.g., Frechet inception distance (FID) and the inception score (IS) [19].

Algorithm 2 (Fig. 13.7)::*Lipizzaner* directs each cell executing in parallel to iterate over a generations (epochs). In each generation t , a cell, n_k , starts by copying the latest versions of its neighbors \mathbf{n}_k to set up its sub-populations \mathbf{g}_k and \mathbf{d}_k . Next, each cell independently executes Algorithm 4 (Fig. 13.9)::*CoevolveAndTrainModels*. This algorithm terminates with the cell neighborhood updating its own models, i.e., $\mathbf{g}_{k,1}, \mathbf{d}_{k,1}$. Two key steps for information propagation are: the update at the start of a generation t when a cell n forms a sub-population \mathbf{n} by copying models from its neighbors. The replacement of the best models in the cell after application of coevolution within the sub-population of the cell.

Upon each cell returning its sub-populations to Algorithm 2 (Fig. 13.7)::*Lipizzaner*, the mixture weights \mathbf{w} are evolved. After all generations or a computational deadline is reached Algorithm 2 (Fig. 13.7)::*Lipizzaner* selects the best performing ensemble \mathbf{e}^* across the entire topology and returns them as *Lipizzaner*'s solution.

Selection promotes fitter models over less fit ones when updating a sub-population \mathbf{n} . *Lipizzaner* uses tournament selection, Algorithm 4 (Fig. 13.9)::*CoevolveAndTrainModels* line 3. First, the latest neighbors models are copied to form the sub-populations and then selection is applied. After all GAN training is completed and all models are reevaluated, the generator and discriminator with the best training loss \mathcal{L} replace the worst in the sub-populations, Algorithm 4 (Fig. 13.9)::*CoevolveAndTrainModels* line 17.

The models' performance \mathcal{L} in sub-populations depends on its adversary, see Algorithm 5 (Fig. 13.10)::*EvaluateGANPairs*. Calculated with some loss function M the performance is expected to vary. The fitness ϕ of a model ($g_i \in \mathbf{g}$ or $d_j \in \mathbf{d}$) is based on a solution concept, f_s^{\min} , i.e., the “best worst case” \mathcal{L} against all its adversaries.

In *Lipizzaner*, GAN training is a black-box component which applies SGD to update the parameters of the models, Algorithm 4 (Fig. 13.9)::

Algorithm 2: Lipizzaner: In parallel, for each cell, initialize settings then iterate over each generation t . Each generation t , retrieve neighbors to build generator \mathbf{g}_k and discriminator \mathbf{d}_k sub-populations, evolve generators \mathbf{g}_k and discriminators \mathbf{d}_k trained with SGD, replace self $n_{k,1}$ with best, and finally evolve weights w for a neighborhood mixture model e^* .

```

Input : T : Total generations, N : Population on topology cells, s : Neighborhood size,
           $\theta_{EA}$  : Parameters for MixtureEA,  $\theta_{COEV}$  : Parameters for
          CoevolveAndTrainModels,  $\theta_D$  : Training dataset,  $\theta_p$  : Dataset sampling
          fraction
Return: e : ensemble of generators and mixture weights
//Asynchronous parallel execution of all cells in topology
1 for  $k \in [1, \dots, |N|]$  do
2    $s \leftarrow \text{getDataSubset}(\theta_D, \theta_p)$  //Create sub-set of the dataset
3    $\mathbf{n}_k, w, f_w \leftarrow \text{initializeNeighborhoodAndMixtureWeights}(s, s)$  //Uniformly
      random initialization of settings
   //Iterate over generations
4   for  $t \in [0, \dots, T]$  do
5      $\mathbf{n}'_k \leftarrow \text{copyNeighbors}(k, s, N)$  //collect neighbour cells individuals
        for the sub-populations
6      $\mathbf{n}'_k \leftarrow \text{CoevolveAndTrainModels}(\mathbf{n}'_k, \theta_{COEV})$  //Coevolve GANs using
        Algorithm 4
7      $\mathbf{n}_k \leftarrow \text{updateNeighborhood}(\mathbf{n}'_k, k, N)$ //Update neighborhood
        (population), i.e. Y
8      $w, f_w \leftarrow \text{MixtureEA}(w, f_w, \mathbf{n}_k, \theta_{EA})$ //Evolve mixture weights,
        Algorithm 3
9    $q_k \leftarrow \text{FinalGeneratorScore}(w, n)$  //Compute the generator ensemble final
      score
10  $q_k \leftarrow \text{CollectGeneratorScore}(w, n)$  //Collect the generator ensemble final
      score
11 return  $e^*$  //Best generator mixture

```

Fig. 13.7 Algorithm 2

Algorithm 3: MixtureEA: Evolve generator mixture weights w with a 1+1 ES.

```

Input :  $\mu$  : Mutation rate,  $n$  : Cell neighborhood sub-population,  $w$  : Mixture weights,
           $f_w$ : Mixture quality
Return: w : mixture weights,  $f_w$ : mixture quality
1  $w' \leftarrow \text{mutate}(w, \mu)$  //Gaussian mutation of mixture weights
2  $f'_w \leftarrow \text{evaluateMixture}(w', n)$ //Evaluate generator mixture score, e.g. FID
    for images
3 if  $f'_w < f_w$  then
   //Replace if new mixture weights are better
4    $w, f_w \leftarrow w', f'_w$  //Update mixture weights and score
5 return  $w, f_w$ 

```

Fig. 13.8 Algorithm 3

Algorithm 4: CoevolveAndTrainModels: Select a new sub-population from the current one. Assign all models a loss function at random. Each mini-batch trains discriminators \mathbf{d} against a randomly drawn generator and generators \mathbf{g} against a randomly drawn discriminator, using SGD. Evaluate all against each other, using the solution concept, f_s^{\min} , of “best worst case” loss value to choose how to replace center $n_{k,1}$. Return this new cell neighborhood \mathbf{n} .

```

Input :  $\tau$  : Tournament size,  $\mathbf{X}$  : Input training dataset,  $\beta$  : Mutation probability,  $\mathbf{n}$  :
Cell neighborhood sub-population,  $M$  : Loss functions
Return:  $\mathbf{n}$  : Cell neighborhood sub-population
1  $\mathbf{B} \leftarrow \text{getMiniBatches}(\mathbf{X})$  //Load minibatches
2  $\phi \leftarrow \text{EvaluateGANPairs}(\mathbf{B}, \mathbf{n})$  //Evaluate all updated GAN pairs, Alg. 5
3  $\mathbf{g}, \mathbf{d} \leftarrow \text{tournamentSelect}(\mathbf{n}, \phi, \tau)$  //Select using loss( $\mathcal{L}$ ) as fitness
4 for  $B \in \mathbf{B}$  do
    //Loop over batches
5    $\mathbf{n}_\delta \leftarrow \text{mutateLearningRate}(\mathbf{n}_\delta, \beta)$  //Update neighborhood learning rate
      with Gaussian mutation
6    $d \leftarrow \text{getRandomOpponent}(\mathbf{d})$  //Get uniform random discriminator
7   for  $g \in \mathbf{g}$  do
        //Evaluate generators and train with SGD
8      $g, d \leftarrow \text{pickRandom}(M)$  //Random draw of loss function for each
        generator and discriminator
9      $\nabla_g \leftarrow \text{computeGradient}(g, d, B)$  //Compute gradient for neighborhood
        center
10     $g \leftarrow \text{updateNN}(g, \nabla_g, B)$  //Update with gradient
11     $g \leftarrow \text{getRandomOpponent}(g)$  //Get uniform random generator
12    for  $d \in \mathbf{d}$  do
        //Evaluate discriminator and train with SGD
13       $g, d \leftarrow \text{pickRandom}(M)$  //Random draw of loss function for each
        generator and discriminator
14       $\nabla_d \leftarrow \text{computeGradient}(d, g, B)$  //Compute gradient for neighborhood
        center
15       $d \leftarrow \text{updateNN}(d, \nabla_d, B)$  //Update with gradient
16  $\phi \leftarrow \text{EvaluateGANPairs}(\mathbf{B}, \mathbf{n})$  //Evaluate all updated GAN pairs, Alg. 5
17  $\mathbf{n} \leftarrow \text{replaceCenterIndividuals}(\mathbf{n}, \phi)$  //Best generator and discriminator are
      placed in the center based on loss
18 return  $\mathbf{n}$ 

```

Fig. 13.9 Algorithm 4

CoevolveAndTrainModels lines 9, 10, 14, and 15. We can set up loss functions (operators) with corresponding optimization objectives and randomly assign one to a model after selection.

13.3.2.1 Dataset Sampling

Instead of using the whole training dataset, each cell of the topology uses a subset of the training dataset which is sampled from the whole training dataset. Figure 13.11

Algorithm 5: EvaluateGANPairs: Evaluate all models against each other using the “best worst case” solution concept f_s^{\min} based on the training loss \mathcal{L}

```

Input :  $\mathbf{B}$  : Minibatches,  $\mathbf{n}$  : Cell neighborhood sub-population
Return:  $\phi$  : Generator and discriminator fitnesses, based on training loss  $\mathcal{L}$ 
1  $B_r \leftarrow \text{getRandomMiniBatch}(\mathbf{B})$  //Get random minibatch
2 for  $g, d \in \mathbf{g} \times \mathbf{d}$  do
   | //Evaluate all GAN pairs
3   |  $\mathcal{L}_{g,d} \leftarrow \text{evaluate}(g, d, B_r)$  //Evaluate GAN
4   |  $\phi_g \leftarrow f_s^{\min}(\mathcal{L}_{g,d})$  //Fitness for generator is the ‘‘best worst case’’
   | loss value ( $\mathcal{L}$ )
5   |  $\phi_d \leftarrow f_s^{\min}(\mathcal{L}_{g,d})$  //Fitness for discriminator is the ‘‘best worst
   | case’’ loss value ( $\mathcal{L}$ )
6 return  $\phi$ 

```

Fig. 13.10 Algorithm 5

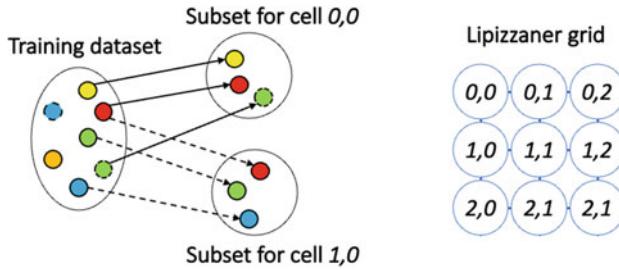


Fig. 13.11 Example of how the whole training dataset is sampled to generate training data subsets to train the different cells of the grid. In this example, a 3×3 grid defines the topology, and it is shown how the subsets for the cells $0, 0$ and $1, 0$ are defined

illustrates how subsets of data are defined for each cell by sampling the entire dataset (for the cells $0, 0$ and $1, 0$). For each cell of the topology, Lipizzaner algorithm searches for and returns a mixture of models composed from the models collected from the cell neighborhood. Thus, the mixture is the fusion set of diverse models in the neighborhood trained by different subsets of the training dataset. Next, we describe an implementation of the Lipizzaner system for AEL.

13.3.3 Implementation of Distributed Computing of Lipizzaner

Figure 13.12 outlines Lipizzaner execution for a given cell. Along the evolutionary process, communication between the processes is performed to exchange relevant information (such as ANNs parameters that define the individuals).

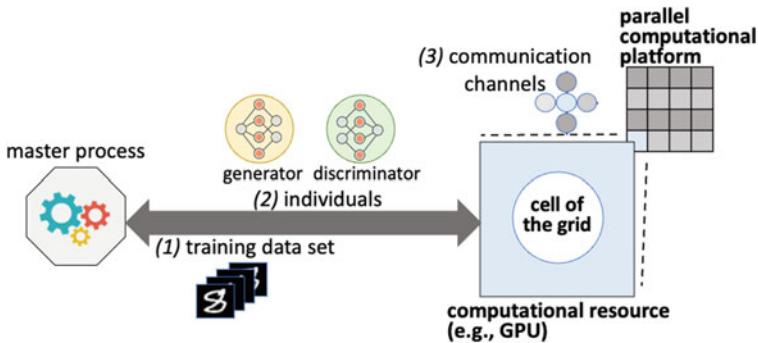


Fig. 13.12 Scheme of how Lipizzaner runs in parallel

With large grid sizes, training iterations could become out of sync, leaving some clients behind the rest. This can lead to the master process waiting too long to return the results. Thus, the Lipizzaner master process is allowed to end the run as soon as a proportion of the client nodes had successfully finished and a specific waiting time had expired.

Handling errors is critical when running Lipizzaner on large grids because the probability of any client to fail. As most of these errors depend on the HPC system, we make Lipizzaner resilient by using a checkpointing procedure. The clients save their running status (i.e., a checkpoint) and send it to the master process to store or update it. A checkpoint consists of the parameters of the ANNs (generator and discriminator) in the center of the cell and the values of the hyper-parameters that drive Lipizzaner. These checkpoints allow a master process to restart any GAN training that halted. We store only information about the central models in the cell and when restarting training, each cell contacts its neighborhood and pulls information from them.

Another requirement for resiliency was to allow the client nodes to continue the training if there is a failure. Thus, the cells in the dead client's neighborhood simply do not try to communicate with it or update their sub-populations up to a maximum number of nodes that can fail during a run. Further, it is difficult to guarantee the exact number of clients that will successfully connect on some HPC so a maximum and minimum number of clients are specified.

13.4 Experiments and Discussion

In this section, we summarize and discuss relevant experimental studies of Lipizzaner for adversarial evolutionary learning in the form of GAN training. In particular, we present investigations regarding:

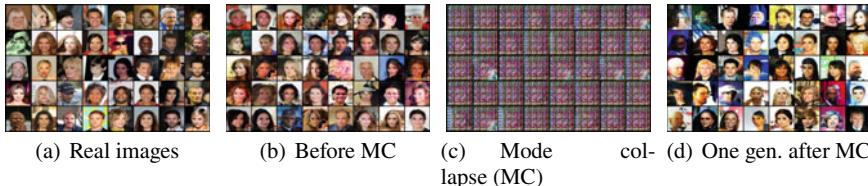


Fig. 13.13 Sequence of CelebA images generated during the competitive coevolutionary GAN training, when a mode collapse (MC) occurs

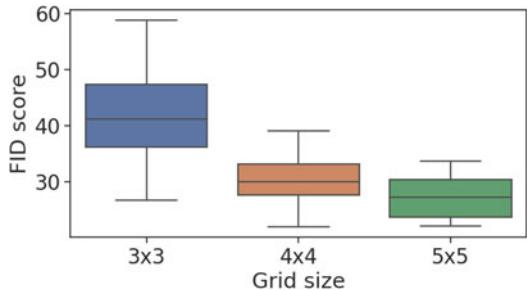
- **Spatial population:** The use of spatial coevolutionary algorithms and diverse parameters to overcome pathologies to improve GAN training is shown in Sect. 13.4.1.
 - **Topology:** The effect on the population diversity for GANs trained with different topologies and communication is shown in Sect. 13.4.1.1.
 - **Scale:** The effect of large spatial grids' impact on the accuracy of the generative model trained is shown in Sect. 13.4.1.2.
- **Operator:** The diversity of loss function combinations at the cells in the spatial topology is shown in Sect. 13.4.2.
- **Data:** The impact of diverse data for spatial coevolutionary GAN training is shown in Sect. 13.4.3.
- **Task objective transfer:** The ability to use the population diversity to create high-quality and diverse ensembles of generative models is shown in Sect. 13.4.4.

13.4.1 Spatial Coevolution for Generative Adversarial Network Training

An initial study showed that AEL with CCA can enhance the performance of gradient-based GAN training methods [2]. Two empirical experiments were considered: (a) a theoretical model was used to analyze the capability of coevolutionary algorithms to solve theoretical GAN problems and (b) empirical experimentation was carried out to evaluate it on commonly used image-generation datasets. The results showed that the use of a competitive coevolutionary GAN training could avoid more frequent degenerate training behaviors with the diversity that comes from a population (phenotypes). For example, Fig. 13.13 illustrates how the competitive coevolutionary approach escaped the mode collapse in the next generation. These results motivated the design and further research on spatially distributed coevolutionary GAN training proposed by Lipizzaner.

As presented in Sect. 13.3.2, the Lipizzaner system combines spatial coevolution with gradient-based learning to improve the robustness and scalability of GAN training. One of the main features of Lipizzaner is that using the cellular

Fig. 13.14 Distribution of FID results on MNIST dataset



(topology) approach based on a grid topology promotes population diversity, i.e., it provides spatial population diversity. The improvement of the GAN training with the population diversity was confirmed with an experimental analysis of *Lipizzaner* by evaluating the quality and diversity of the samples generated by the computed generative models [54]. The experiments considered *Lipizzaner* using different grid sizes (from 2×2 to 5×5) and two image datasets: MNIST [62] and CelebA [69].

The empirical results demonstrated that the samples' quality and diversity improve as the grid size grows. For example, in MNIST experiments, the average FID score decreased (improved) from 40.78 on the 3×3 to 26.78 on the 5×5 (see Fig. 13.14), and the average TVD score decreased (improved) from 0.14 on the 3×3 to 0.12 on the 5×5 . In addition, *Lipizzaner* provided the most competitive results when comparing it against other GAN training approaches (i.e., vanilla GAN, WGAN, and EGAN) with the same computational budget.

Moreover, the diversity in the genome space (i.e., the parameters of the ANNs) increased with the grid size. Thus, the spatial separation of the sub-populations fosters diversity in the whole population. The gain of population diversity leads the generative models to produce better and more diverse samples because the ensembles of generators consist of more diverse generators [54].

Lipizzaner is also scalable, experiments showed that the spatial grid distribution architecture allowed the computational effort to increase linearly instead of quadratically [95]. This claim was supported by the observation that the wall clock time per training epoch grew nearly linear with the grid size. The communication took 0.5 seconds on average. Finally, the asynchronous communication pattern led to the usage of different time slots and therefore reduced high network peak loads.

Previous studies also showed with an ablation study that other evolutionary learning features of *Lipizzaner* had a significant impact, such as learning rate hyper-parameter evolution and ensemble weight optimization [54, 105]. These studies investigated different *Lipizzaner* variations: without learning rate evolution, without local selection pressure, without ensemble weight optimization, without selection and communication, and only keeping ensemble weight optimization. Moreover, the empirical results showed that the impact of ablating evolution of the initial value of the Adam learning rate is the most severe. This indicated that the performance and convergence of both GAN and network model training are

susceptible to initial conditions. However, when selection and communication were eliminated while maintaining learning rate mutation and ensemble weight optimization, `Lipizzaner` achieved an average mean FID score closer to the baseline than any single ablation. Furthermore, retaining only ensemble weight optimization showed the positive impact of learning rate mutation on the results. So, these findings confirmed that communication, selection, learning rate evolution optimization, and ensemble optimization each, as well as in combination, provide a significant contribution.

13.4.1.1 Signal Propagation Through Spatially Distributed Population

Different topologies can be defined on the spatially distributed GAN training when using `Lipizzaner` for AEL. The spatial topology plus the cells neighborhood impacts on the signal propagation through the cells. As presented in Sect. 13.4.1, `Lipizzaner` was designed to consider different topologies, e.g., the toroidal two-dimensional grid and the one-dimensional ring. In `Lipizzaner`, the signal propagation takes the form of copying the best individuals (i.e., ANNs) to neighborhood cells to update the sub-populations, after each training epoch. `Lipizzaner` uses a two-dimensional grid by default (i.e., all the experimental analyses performed used this type of topology).

An empirical investigation of the ring topology in `Lipizzaner` was performed [108]. The primary goal was to investigate the performance quality and efficiency of different directionality and reach of the signal propagation (effectively migration to one or more neighbors on each side of a cell) compared to `Lipizzaner` with grid topology. In addition, the diversity in the populations was evaluated. The analysis considered two different ring topology approaches according to the communication radius (i.e., the number of cells that are affected by the signal propagation in each direction): the ring of radius $r = 2$, named Ring(2), which provides a signal propagation similar to the `Lipizzaner` toroidal grid; and the ring of radius $r = 1$, named Ring(1), which reduces the propagation speed, while accelerating the population's convergence.

The use of Ring(1) has the following main benefits: (a) it mitigates the communication overhead because each cell exchanges information with only two cells (instead of four); (b) the reduction of sub-population size leads to the reduction of the number of operations required for fitness evaluation, selection, and replacement; and (c) it does not require to have a rectangular grid of cells, but the population size (number of cells in the grid) may be any natural number. The experimental analysis considered three different datasets: MNIST, COVID-19-positive 190 chest X-Ray images provided by Cohen et al. [29], and CelebA. Four methods were evaluated: Grid, Ring(2), and Ring(1), with a fixed computational budget, for three different population sizes: 9, 16, and 25.

The empirical analysis of the evaluated spatially distributed coevolutionary GAN training methods showed that using a ring topology instead of a two-dimensional grid maintained the performance of the trained generative models, and it could even

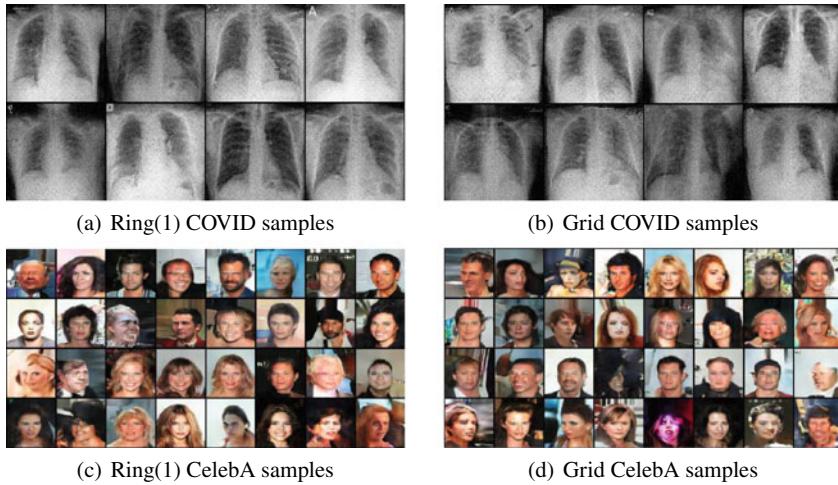


Fig. 13.15 COVID-19 and CelebA images generated by Ring(1) and Grid

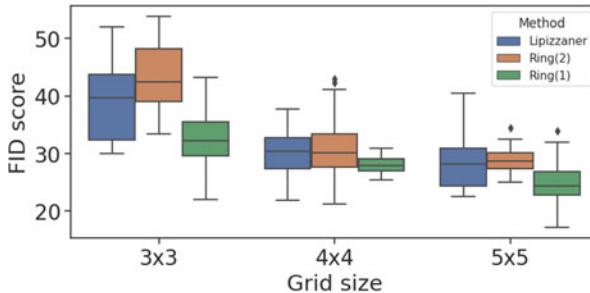


Fig. 13.16 FID results on MNIST dataset for Lipizzaner, Ring(2), and Ring(1)

improve them, depending on the setup. In general, Ring(1) trained the most competitive generative models. Ring(2) and Grid topology provided comparable generative models. Figure 13.15 shows images synthesized by the generative models trained by using Ring(1) and Grid topologies.

Figure 13.16 summarizes the FID results for the MNIST dataset experiments. Ring(1) exhibited the most diverse populations, which diminishes with longer training. When comparing different ring topologies, increasing the migration radius, i.e., Ring(1) versus Ring(2), led to a reduction in diversity. In addition, there was no marked difference in the diversity of populations when comparing migration directionality between Ring(2) and Lipizzaner. Finally, by altering the signal propagation from four to two directions and using a migration radius of one, Ring(1) reduced Lipizzaner's computational time cost by 14.2–41.2% while maintaining comparable quality results.

13.4.1.2 Large-Scale Spatially Distributed Adversarial Learning

Large parallel computation infrastructures enable new distributed methods that take advantage of the computational scale and parallelism. There are studies of the combination of large-scale parallel computation infrastructure, such as high-performance computing (HPC) platforms, with `Lipizzaner` and the ability to scale GAN training to large spatial grids and with datasets with large images [43, 86, 103]. It has been shown that `Lipizzaner` improves the quality of the trained generative models as the grid size increases. However, the results were limited due to the availability of the hardware platforms used for the experiments. Thus, further investigations were performed to evaluate the scalability of `Lipizzaner` by running it on the Summit Supercomputer housed at Oak Ridge National Labs (ORNL) [61]. This HPC environment provided nodes with two IBM POWER9TM processors with 1600GB of non-volatile memory and six GPUs NVIDIA Volta V100s with 32GB memory.

The empirical analysis was reported by Flores et al. [45]. The experiments were performed on two datasets: MNIST and COVID-19 positive 190 chest X-Ray images provided by Cohen et al. [29], considering two image resolutions: 28×28 and 64×64 pixels. Figure 13.18a illustrates some X-Ray images from the COVID-19 dataset. Different ANN architectures were considered: a basic four-layer perceptron (4LP) and two CNNs named CNN28 and CNN64. CNN28 had fewer trainable parameters. Thus, less competitive results were expected for CNN28. The goal of comparing these architectures was to show that scaling `Lipizzaner` may overcome the limitation of using non-optimal ANN architectures, i.e., with fewer parameters, when the nodes of the HPC platform have hardware (memory) limitations.

The MNIST experiments were performed on grid sizes up to 18×18 . The results showed significant improvements in the quality of the generated images when grid sizes increase (see Fig. 13.17a). Thus, the average FID (lower is better) score ranged from 41.23 (when `Lipizzaner` was run on a 3×3 grid) to 17.20 (when a 18×18 grid was used). In the COVID-19 positive chest X-Ray images generation, the results confirmed that increasing the grid size up to 12×12 led to better results, with the same ANN architecture (see Fig. 13.17b). Thus, when increasing the grid size from 3×3 to 12×12 , the CNN28 IS (higher is better) results increased (improved) from 1.31 to 1.49, and the CNN64 results improved from 1.37 to 161. When comparing the results achieved by both CNNs, it was observed that scale may make up for lower ANN complexity. Thus, CNN28 was able to get an IS of 1.49 (when using a 12×12), which is higher than the IS of 1.37 got by CNN64 when using a 3×3 . The less complex CNN provided more accurate generative models than the more complex CNN when increasing the scale. Figure 13.18b and c illustrates samples produced by the generative models trained by using the 12×12 grid to train CNN28 and CNN64.

The improvements in the results, when `Lipizzaner` training was performed on a large-scale HPC platform, had similar run time. This indicates that it is possible to scale the distributed GAN training without significant additional computational time increase. In summary, it was shown that spatially distributed AEL improved performance even when using sub-optimal ANN architectures due to hardware constraints. Next, we study how operator diversity can impact AEL with `Lipizzaner`.

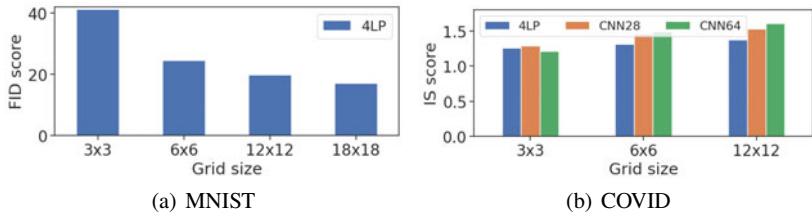


Fig. 13.17 Mean results on MNIST and COVID-19 experiments

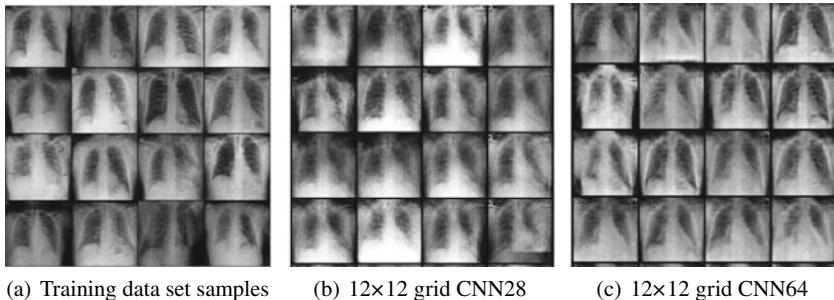


Fig. 13.18 Real and generated COVID-19 chest x-Ray images

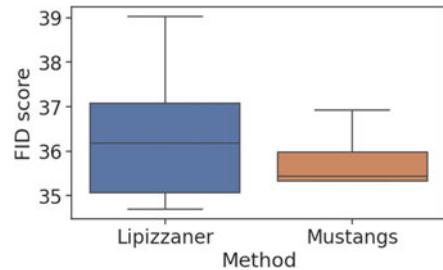
13.4.2 Spatial Adversarial Evolutionary Learning with Operator Diversity

GAN training pathologies, such as instability and mode collapse, generally arise from lack of diversity in the generator and discriminator interactions. AEL with Lipizzaner injects diversity in the network parameters of the ANNs in the population (genome space) by training two populations whose ANNs (individuals) are separated in cells of a toroidal grid. Furthermore, as presented in Sect. 13.3.2 (Algorithm 4 (Fig. 13.9)::CoevolveAndTrainModels lines 8 and 13), Lipizzaner includes a property devised to further increase diversity in the population through the application of a randomized variation (mutation) operator affecting the ANN training loss function. Mutation in Lipizzaner is performed by applying SGD to the networks according to a given loss value. Thus, inspired by EGAN [110], Lipizzaner was extended by randomly picking a loss function from a set of three loss (BCE, MSE, and HEU) at each epoch and each cell of the grid [104].

The impact of operator-driven diversity was analyzed by comparing four variants of Lipizzaner (i.e., three using only one of the tree loss functions and one using the randomized loss function, a.k.a., Mustangs), vanilla GAN, and EGAN. The experimental analysis considered MNIST and CelebA datasets and three grid sizes: 3×3, 4×4, and 5×5.

The quality of the images (FID score) for both datasets was better for all Lipizzaner variations which produced better images compared to vanilla GAN

Fig. 13.19 FID results on CelebA for Lipizzaner and Mustangs on a 2×2 grid



and EGAN. Among the Lipizzaner variations, Mustangs and Lipizzaner with BCE provided the most competitive results. However, the results indicated that Mustangs is robust to the variation of single loss functions (FID results showed reduced standard deviation) and can still find a high-performing mixture of generators (see Fig. 13.19). This provides further empirical evidence that diversity, both in genome (ANN parameters) and operator (loss function) space, provides robust GAN training.

The diversity of the generated samples also showed that the Lipizzaner variants outperformed vanilla GAN and EGAN [104]. In addition, Mustangs trained the generative models that produced the most diverse images. Moreover, the diversity of the evolved ANN parameters was investigated to see if any ANN was replicated on grid. The result showed no significant replication of generator ANNs for all the Lipizzaner variations. Moreover, the diversity varied between setups and Lipizzaner variants. In general, the ANN diversity increased with the grid size, and it was further increased for Mustangs, randomization of loss functions during training. The increase is most likely due to the impact of the loss function on the training convergence of the ANN parameters. To conclude, the experimental evaluation showed that the spatially distributed coevolutionary GAN training combined with the SGD mutation operator (i.e., Mustangs) provided further diversity. Next, the impact of data diversity on AEL performance with Lipizzaner is presented.

13.4.3 Spatial Adversarial Evolutionary Learning with Data Diversity

Generative models consisting of mixtures of generators perform well (see Sect. 13.4.1). However, relying on multiple generators can be resource-intensive. AEL with Lipizzaner mitigates some of this issue by distributing the training of mixtures of GANs on different computational resources to train them in parallel. A complementary strategy is to decrease the amount of training data. A straightforward approach is to train various GANs on smaller subsets of the training dataset. Using less training data entails lower time and storage (disk and memory) requirements

while depending on the ensemble of generators to limit possible loss in performance from the reduction of training data.

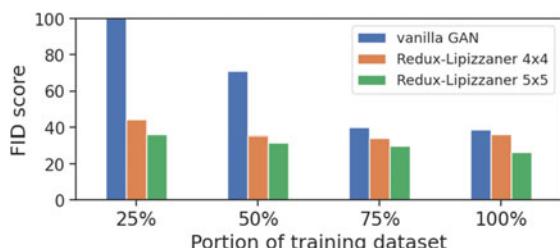
As observed in Sect. 13.4.1.1, Lipizzaner spatial topology with overlapping neighborhood provides signal propagation (i.e., implicit communication) among the cells in the grid. This communication occurs during the update of the sub-population, when the neighbors are copied, since the best pair generator–discriminator in each sup-population is replaced after each epoch for each cell. The research presented in this section concerns the impact of the use of different subsets of training datasets at each cell of the grid topology. This approach was named Redux-Lipizzaner and it empirically studies data sample diversity in the spatially distributed GAN training.

The fundamental aspect of the Redux-Lipizzaner investigation was: the impact on the quality of generative models when training with less data. Instead of training each sub-population with the entire training dataset, per Lipizzaner, Redux-Lipizzaner applies random sampling with replacement of the training data to define different subsets (partitions) of data that are used as a training dataset at each cell (see Fig. 13.11). Thus, each cell has its own *training subset* of data.

The experimental analysis was performed on well-known image datasets: MNIST and CelebA. MNIST experiments were performed on grid sizes of 3×3 and 4×4 , and CelebA on a grid size of 3×3 . A vanilla GAN training method was also considered as baseline. Four different experiments for each dataset and grid size were defined by training the grid cells with 25, 50, 75, and 100% of the data batches. The dataset sampling process is independent for each cell of the grid and consists of randomly picking different batches of the training dataset. For each dataset, all experiments had the same budget of batches while keeping batch size, i.e., the number of images per batch, constant. Thus, when the training dataset size per cell varied, the number of generations was adjusted to maintain the same batch budget.

Figure 13.20 illustrates the mean FID score (lower is better) for MNIST experiments. Focusing on vanilla GAN (baseline), FID increased (performance worsened) as the GAN was trained on less data. The mean FID when the GAN used 25% of data was 574.6 (i.e., the generative models produced basically noise). When the data subset was doubled to 50%, the mean FID score improved to 71.2 (i.e., the generative models created poor-quality images). Mean FID significantly improved when 75% and 100% of data were used with values of 39.8 and 38.3, respectively. Additionally, for the 4×4 grid, the mean FID scores were 44.1, 40.5, 33.0, and 30.7, when the cells

Fig. 13.20 FID results on MNIST experiments. The Y-axis was cut at FID=100 to facilitate comparison of the methods, x-axis shows the fraction of dataset used



were training with 25%, 50%, 75%, and 100% of the dataset, respectively. In the case of the 5×5 grid, the mean FID scores were 36.2, 31.8, 30.1, and 26.3. Finally, for the CelebA experiments, the same outcomes were observed with mean FID scores which were 48.3, 42.4, 38.9, and 42.7 when the cells were training with 25%, 50%, 75%, and 100% of the dataset, respectively.

Three main results were drawn from these results:

1. The application of spatially distributed GAN training of Redux-Lipizzaner dramatically improved the results (in comparison with vanilla GAN).
2. As the grid size increased, the trained generative models produced better quality samples (i.e., 4×4 FID is higher/worse than 5×5 FID). Moreover, 5×5 trained with 25% of the training dataset achieved comparable results to method 4×4 trained with 75 and 100% of the dataset.
3. For all the experiments, generative models trained with 75% of the data produced fair quality samples as the models trained with the whole dataset.

In summary, it was observed that the spatially distributed evolutionary GAN training of Redux-Lipizzaner leverages neighborhood information exchange to produce high-performing generator ensembles. The spatially distributed grids allow training with fewer exemplars from the dataset, since signal propagation facilitates information exchange among the grid. Next, we investigate how the ensembles generated by AEL with Lipizzaner can be re-purposed.

13.4.4 Re-purposing Heterogeneous Generative Ensembles with Optimization Objective Diversity

One approach to enhance the machine learning models' task performance (e.g., classification or prediction) is to combine (fuse) them. This is sometimes known as ensemble learning [119]. There exists a body of work on applying evolutionary computation to learn ensembles of machine learning models known as Evolutionary Ensemble Learning (EEL) [101]. The main idea is to compensate for the limitations of the models trained with the abilities of other methods by combining evolutionary computation with learning algorithms.

Lipizzaner constructs generative models in a mixture of generators that provide competitive results (i.e., high-quality generative models) while mitigating GAN training pathologies. Evolutionary Strategies (ES) [73] is used in Lipizzaner to evolve the mixture weights used to define the ensemble of generators consisting of the sub-populations of generators. The main objective of the ES in Lipizzaner is to maximize the quality of the generated samples (e.g., to optimize IS or FID metrics).

Toutouh et al. [107] combined the spatially distributed GAN training of Lipizzaner with EEL. The principal goal of this research was to improve the existing GAN training methods by applying a methodology inspired by the successful EEL. Applying this approach makes it possible to achieve better results in the task

of constructing generative model ensembles without re-training the generative models from scratch. The methodology combines heterogeneously trained generators to create more competitive and efficient generative models while avoiding the high computational cost associated with re-training them. The focus was on applying EEL to *re-purpose generative models*, i.e., given a set of heterogeneous generators that were optimized to provide the most accurate samples, create ensembles of them for optimizing a different objective (in this case, maximize the diversity of the generated samples). The EEL approach evolved the mixture weights that define the ensemble of generators to optimize the diversity measure (i.e., total variation distance) of generated samples from an ensemble. At the same time, *Lipizzaner* optimizes the network weights to optimize the minimax optimization GAN objective, which maximizes the quality of the generated samples.

For this new task of constructing ensembles of generative models with a different objective, two different optimization problems were defined to optimize (minimize) TVD: Restricted Ensemble Optimization of GENerators (REO-GEN) and Non-Restricted Ensemble Optimization of GENerators (NREO-GEN). REO-GEN restricts the ensemble search to a given specific size, while NREO-GEN relaxes the restriction by defining an upper bound for the size. These two optimization problems were approached with a genetic algorithm, which applies specifically designed crossover and mutation operators, and two greedy algorithms as baselines.

The proposed approaches were empirically evaluated over the MNIST dataset generation problem. The generative models were trained using *Lipizzaner* on a 3×3 grid. The experimental analysis showed that both evolutionary approaches built generative models that significantly improved the diversity of the pre-trained generators while keeping a bounded computational cost. The genetic algorithm addressing NREO-GEN obtained the most competitive TVD results. The degree of freedom in searching for ensembles without specific sizes of NREO-GEN allowed the genetic algorithm to explore the search space more efficiently, leading to better results than the other approaches. Moreover, even though the greedy methods were less competitive than the evolutionary method, they were still able to compute ensembles of generators that improved diversity. However, the effectiveness of the greedy methods decreased when the ensemble sizes increased, due to the associated computational cost of the greedy heuristic. Finally, the constructed ensembles significantly enhanced the TVD and also improved the FID score. Ensembles with four generators produced the best results for generating MNIST digits. Interestingly, 46.7% of the ensembles returned by the genetic algorithm for NREO-GEN comprised four generators. In summary, the adversarial evolutionary learning with a spatial coevolutionary algorithm can help enable high performance on tasks with different objectives. Next, we summarize the AEL experiments performed with *Lipizzaner*.

13.4.5 Summary

The *Lipizzaner* system for AEL demonstrates the ability to create high-performing generative models with reduced resources. This is done with a distributed spatial CCA for GAN training. In addition, *Lipizzaner* can scale to improve the performance even with the use of sub-optimal ANN architectures due to hardware limitation. Furthermore, *Lipizzaner* provides diversity that enables optimization of different generative task objectives. We also demonstrated how an increase in diversity improves performance. In summary, the main advantages of AEL with *Lipizzaner* are

- (a) fast convergence due to gradient-based steps;
- (b) improved convergence due to hyper-parameter evolution;
- (c) robustness due to spatial topology with coevolution and communication;
- (d) computational resource reduction from use of diverse data subset;
- (e) improved samples quality and diversity due to mixture evolution; and
- (f) scalability due to spatial distribution topology and asynchronous parallelism.

13.5 Conclusions and Future Work

We presented an overview of AEL, i.e., evolutionary approaches with competing adversaries which adapt over time. This is often defined as a minimization–maximization problem. We introduced a categorization of AEL based on the number of adversaries, population dynamics, and learning methods. CCA, MARL, AML, and EGT are categories that address AEL. We provided an in-depth demonstration of AEL applied to address the challenges of GAN training by applying a spatially distributed competitive coevolution method named *Lipizzaner*. By using populations of multiple individual solutions (i.e., ANNs), the *Lipizzaner* fosters diversity in the gradient-based GAN learning. Extensive experimental analysis has been summarized to show that diversity increases the robustness of the GAN training. The spatial topology helps to improve the scaling possibilities by reducing the computational complexity and to keep diversity through the grid. In addition, it has been shown how the application of hyper-parameter evolution and the diversity in the learning objectives, search operators, and data can be applied to improve the training further. All these results lead us to consider that the application of AEL methods in which diversity among populations is maintained may facilitate convergence in dealing with problems involving adversarial agents.

Future work includes further investigation on AEL, including the *Lipizzaner* framework, on other ML tasks, such as RL and multi-agent learning. Moreover, other input representations and application areas for AEL should be explored. Finally, the theory regarding AEL needs further elaboration because the current theoretical methods from EA theory do not completely explain or validate AEL methods.

References

- Adamopoulos, K., Harman, M., Hierons, R.M.: How to overcome the equivalent mutant problem and achieve tailored selective mutation using co-evolution. In: Genetic and Evolutionary Computation—GECCO 2004, pp. 1338–1349. Springer (2004)
- Al-Dujaili, A., Schmiedlechner, T., Hemberg, E., O'Reilly, U.-M.: Towards distributed coevolutionary GANs. In: AAAI 2018 Fall Symposium (2018)
- Anand, S., Burke, E.K., Chen, T.Y., Clark, J., Cohen, M.B., Grieskamp, W., Harman, M., Harrold, M.J., McMinn, P., et al.: An orchestrated survey of methodologies for automated software test case generation. *J. Syst. Softw.* **86**(8), 1978–2001 (2013)
- Anderson, H.S., Kharkar, A., Filar, B., Roth, P.: Evading machine learning malware detection. Black Hat (2017)
- Arcuri, A., Yao, X.: Coevolving programs and unit tests from their specification. In: Proceedings of the Twenty-Second IEEE/ACM International Conference on Automated Software Engineering, pp. 397–400. ACM (2007)
- Arcuri, A., Yao, X.: Co-evolutionary automatic programming for software development. *Inf. Sci.* **259**, 412–432 (2014)
- Arjovsky, M., Chintala, S., Bottou, L.: Wasserstein GAN (2017). [arXiv:1701.07875](https://arxiv.org/abs/1701.07875)
- Arora, S., Ge, R., Liang, Y., Ma, T., Zhang, Y.: Generalization and equilibrium in generative adversarial nets (GANs) (2017). [arXiv:1703.00573](https://arxiv.org/abs/1703.00573)
- Arora, S., Risteski, A., Zhang, Y.: Do gans learn the distribution? Some theory and empirics. In: International Conference on Learning Representations (2018)
- Axelrod, R.: The Evolution of Cooperation 10. Basic, NY, New York (1984)
- Baeta, F., Correia, J.A., Martins, T., Machado, P.: Exploring expression-based generative adversarial networks. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO '22, New York, NY, USA, pp. 1862–1869. Association for Computing Machinery (2022)
- Bai, H., Cheng, R., Jin, Y.: Evolutionary reinforcement learning: a survey. *Intell. Comput.* **2**, 0025 (2023)
- Bai, J., Zhang, Y., Xiao, Z., Ye, F., Li, Y., Alazab, M., Jiao, L.: Immune evolutionary generative adversarial networks for hyperspectral image classification. *IEEE Trans. Geosci. Remote Sens.* **60**, 1–14 (2022)
- Baiocletti, M., Coello, C.A.C., Bari, G.D., Poggioni, V.: Multi-objective evolutionary GAN. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion (2020)
- Barr, E.T., Harman, M., McMinn, P., Shahbaz, M., Yoo, S.: The oracle problem in software testing: a survey. *IEEE Trans. Softw. Eng.* **41**(5), 507–525 (2014)
- Biggio, B., Roli, F.: Wild patterns: ten years after the rise of adversarial machine learning. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (2017)
- Bloembergen, D., Tuyls, K., Hennes, D., Kaisers, M.: Evolutionary dynamics of multi-agent learning: a survey. *J. Artif. Intell. Res.* **53**, 659–697 (2015)
- Bongard, J.C., Lipson, H.: Nonlinear system identification using coevolution of models and tests. *IEEE Trans. Evol. Comput.* **9**(4), 361–384 (2005)
- Borji, A.: Pros and cons of GAN evaluation measures. *Comput. Vis. Image Underst.* **215**, 103329 (2022)
- Boyd, R.: Mistakes allow evolutionary stability in the repeated prisoner's dilemma game. *J. Theor. Biol.* **136**(1), 47–56 (1989)
- Bronfman-Nadas, R., Zincir-Heywood, N., Jacobs, J. T.: An artificial arms race: could it improve mobile malware detectors? In: 2018 Network Traffic Measurement and Analysis Conference (TMA), pp. 1–8. IEEE (2018)
- Bucci, A.: Emergent geometric organization and informative dimensions in coevolutionary algorithms. Ph.D. thesis, Brandeis University (2007)

23. Busoniu, L., Babuska, R., De Schutter, B.: A comprehensive survey of multiagent reinforcement learning. *IEEE Trans. Syst. Man Cybern. Part C (Appl. Rev.)* **38**(2), 156–172 (2008)
24. Byrd, R., Damani, K., Che, H., Calandra, A., Kim, D.-K.: A systematic literature review of volumetric 3d model reconstruction methodologies using generative adversarial networks. *J. Inf. Sci. & Eng.* **38**(6) (2022)
25. Cartlidge, J., Bullock, S.: Combating coevolutionary disengagement by reducing parasite virulence. *Evol. Comput.* **12**(2), 193–222 (2004)
26. Chakraborty, H., Samanta, P., Zhao, L.: Sequential data imputation with evolving generative adversarial networks. In: 2021 International Joint Conference on Neural Networks (IJCNN), pp. 1–8 (2021)
27. Chen, S., Wang, W., Xia, B., You, X., Peng, Q., Cao, Z., Ding, W.: CDE-GAN: cooperative dual evolution-based generative adversarial network. *IEEE Trans. Evol. Comput.* **25**(5), 986–1000 (2021)
28. Cho, H.-Y., Kim, Y.-H.: Stabilized training of generative adversarial networks by a genetic algorithm. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO '19, New York, NY, USA, pp. 51–52. Association for Computing Machinery (2019)
29. Cohen, J., Morrison, P., Dao, L.: COVID-19 image data collection (2020). [arXiv:2003.11597v1](https://arxiv.org/abs/2003.11597v1)
30. Corus, D., Dang, D.-C., Eremeev, A.V., Lehre, P.K.: Level-based analysis of genetic algorithms and other search processes. *IEEE Trans. Evolut. Comput.* **22**(5), 707–719 (2018)
31. Costa, V., Lourenço, N., Correia, J.A., Machado, P.: COEGAN: evaluating the coevolution effect in generative adversarial networks. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '19, New York, NY, USA. Association for Computing Machinery (2019)
32. Costa, V., Lourenço, N., Correia, J., Machado, P.: Neuroevolution of generative adversarial networks. In: Deep Neural Evolution, pp. 293–322. Springer (2020)
33. Costa, V., Lourenço, N., Machado, P.: Coevolution of generative adversarial networks. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 11454. LNCS:473–487 (2019)
34. Costa, V., Lourenço, N., Correia, J., Machado, P.: Exploring the evolution of gans through quality diversity. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference (2020)
35. Costa, V., Lourenço, N., Correia, J., Machado, P.: Using skill rating as fitness on the evolution of gans. *EvoApplications* (2020)
36. Costa, V., Lourenço, N., Correia, J., Machado, P.: Demonstrating the evolution of gans through t-sne. *EvoApplications* (2021)
37. Costa, V., Lourenço, N., Correia, J., Machado, P.: Improved evolution of generative adversarial networks. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion (2021)
38. Costa, V., Lourenço, N., Machado, P.: Coevolution of generative adversarial networks. *EvoApplications* (2019)
39. de Oliveira, A.A.L., Camilo-Junior, C.G., Vincenzi, A.M.R.: A coevolutionary algorithm to automatic test case selection and mutant in mutation testing. In: 2013 IEEE Congress on Evolutionary Computation, pp. 829–836 (2013)
40. Demetrio, L., Biggio, B., Lagorio, G., Roli, F., Armando, A.: Functionality-preserving black-box optimization of adversarial windows malware. *IEEE Trans. Inf. Forensics Secur.* **16**, 3469–3478 (2020)
41. Diao, W., Zhang, F., Sun, J., Xing, Y., Zhang, K., Bruzzone, L.: ZeRGAN: zero-reference GAN for fusion of multispectral and panchromatic images. *IEEE Trans. Neural Netw. Learn. Syst.* 1–15 (2022)
42. Ehrlich, P.R., Raven, P.H.: Butterflies and plants: a study in coevolution. *Evolution* **18**(4), 586–608 (1964)

43. Esteban, M., Toutouh, J., Nesmachnow, S.: Parallel/distributed intelligent hyperparameters search for generative artificial neural networks. In: International Conference on High Performance Computing, pp. 297–313. Springer (2021)
44. Ficici, S.G.: Solution concepts in coevolutionary algorithms. Ph.D. thesis, Brandeis University (2004)
45. Flores, D., Hemberg, E., Toutouh, J., O'Reilly, U.-M.: Coevolutionary generative adversarial networks for medical image augmentation at scale. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 367–376 (2022)
46. Fu, Y., Gong, M., Yang, G., Wei, H., Zhou, J.: Evolutionary gan-based data augmentation for cardiac magnetic resonance image. Mater. & Continua Comput. (2021)
47. Garcia, D., Lugo, A.E., Hemberg, E., O'Reilly, U.-M.: Investigating coevolutionary archive based genetic algorithms on cyber defense networks. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO '17, New York, NY, USA, pp. 1455–1462. ACM (2017)
48. Garciamena, U., Santana, R., Mendiburu, A.: Evolved gans for generating pareto set approximations. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 434–441 (2018)
49. Gleave, A., Dennis, M., Kant, N., Wild, C., Levine, S., Russsell, S.: Adversarial policies: attacking deep reinforcement learning. In: Proceedings of the ICLR-20 (2020)
50. Goldberg, D.E.: Genetic Algorithms in Search, 1st edn. Optimization and Machine Learning. Addison-Wesley Longman Publishing Co., Inc, Boston, MA, USA (1989)
51. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Advances in Neural Information Processing Systems, pp. 2672–2680 (2014)
52. Han, C., Wang, J.: Face image inpainting with evolutionary generators. IEEE Signal Proc. Lett. **28**, 190–193 (2021)
53. He, J., Zhu, Q., Zhang, K., Yu, P., Tang, J.: An evolvable adversarial network with gradient penalty for covid-19 infection segmentation. Appl. Soft Comput. **113**, 107947–107947 (2021)
54. Hemberg, E., Toutouh, J., Al-Dujaili, A., Schmiedlechner, T., O'Reilly, U.-M.: Spatial coevolution for generative adversarial network training. ACM Trans. Evol. Learn. Optim. **1**(2) (2021)
55. Hemberg, E., Zipkin, J.R., Skowyra, R.W., Wagner, N., O'Reilly, U.-M.: Adversarial co-evolution of attack and defense in a segmented computer network environment. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, pp. 1648–1655. ACM (2018)
56. Hingston, P., Preuss, M.: Red teaming with coevolution. In: 2011 IEEE Congress on Evolutionary Computation (CEC), pp. 1155–1163 (2011)
57. Hornby, G.S., Mirtich, B.: Diffuse versus true coevolution in a physics-based world. In: Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation, vol. 2, pp. 1305–1312. Morgan Kaufmann Publishers Inc. (1999)
58. Keaveney, D., O'Riordan, C.: Evolving coordination for real-time strategy games. IEEE Trans. Comput. Intell. AI Games **3**(2), 155–167 (2011)
59. Kewley, R., Embrechts, M.: Computational military tactical planning system. IEEE Trans. Syst. Man Cybern. Part C: Appl. Rev. **32**(2), 161–171 (2002)
60. Krawiec, K., Heywood, M.: Solving complex problems with coevolutionary algorithms. In: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion, pp. 687–713. ACM (2016)
61. Laboratory, O.R.N.: SUMMIT oak ridge national laboratory's 200 petaflop supercomputer (2019). <https://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/>. Accessed 30 Dec 2022
62. LeCun, Y.: The mnist database of handwritten digits (1998). <http://yann.lecun.com/exdb/mnist/>
63. Lehre, P.K.: Runtime analysis of competitive co-evolutionary algorithms for maximin optimisation of a bilinear function. In: Proceedings of the Genetic and Evolutionary Computation

- Conference, GECCO '22, New York, NY, USA, pp. 1408–1416. Association for Computing Machinery (2022)
- 64. Li, J., Madry, A., Peebles, J., Schmidt, L.: On the limitations of first-order approximation in GAN dynamics. In: 35th International Conference on Machine Learning, ICML 2018, vol. 7, pp. 4672–4689 (2017)
 - 65. Li, J., Madry, A., Peebles, J., Schmidt, L.: Towards understanding the dynamics of generative adversarial networks (2017). [arXiv:1706.09884](https://arxiv.org/abs/1706.09884)
 - 66. Li, J., Zhang, J., Gong, X., Lü, S.: Evolutionary generative adversarial networks with crossover based knowledge distillation. In: 2021 International Joint Conference on Neural Networks (IJCNN), pp. 1–8 (2021)
 - 67. Li, Y., Sun, M., Zhang, X.: Perception-guided generative adversarial network for end-to-end speech enhancement. *Appl. Soft Comput.* **128**, 109446 (2022)
 - 68. Liang, Y., Han, Z., Nie, X., Ohkura, K.: Improving generative adversarial network with multiple generators by evolutionary algorithms. *Artif. Life Robot.* **27**, 761–769 (2022)
 - 69. Liu, Z., Luo, P., Wang, X., Tang, X.: Deep learning face attributes in the wild. In: Proceedings of International Conference on Computer Vision (ICCV) (2015)
 - 70. Liu, Z., Wang, J., Liang, Z.: Catgan: category-aware generative adversarial networks with hierarchical evolutionary learning for category text generation. In: AAAI Conference on Artificial Intelligence (2019)
 - 71. Liu, Z.Y., Sabar, N.R., Song, A.: Partial transfer learning for fast evolutionary generative adversarial networks. In: 2021 International Joint Conference on Neural Networks (IJCNN), pp. 1–7 (2021)
 - 72. Liu, Z.Y., Song, A., Sabar, N.R.: Kde-gan: enhancing evolutionary gan with knowledge distillation and transfer learning. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion (2022)
 - 73. Loshchilov, I.: Surrogate-assisted evolutionary algorithms. Ph.D. thesis, University Paris South Paris XI; National Institute for Research in Computer Science and Automatic-INRIA (2013)
 - 74. Mao, X., Li, Q., Xie, H., Lau, R.Y., Wang, Z., Smolley, S.P.: Least squares generative adversarial networks. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2794–2802 (2017)
 - 75. McDonald, M.L., Upton, S.C.: Investigating the dynamics of competition: coevolving red and blue simulation parameters. In: Proceedings of the 37th Conference on Winter Simulation, pp. 1008–1012. Winter Simulation Conference (2005)
 - 76. Menéndez, H.D., Bhattacharya, S., Clark, D., Barr, E.T.: The arms race: adversarial search defeats entropy used to detect malware. *Expert Syst. Appl.* **118**, 246–260 (2019)
 - 77. Antonio, L.M., Coello, C.A.C.: Coevolutionary multiobjective evolutionary algorithms: survey of the state-of-the-art. *IEEE Trans. Evol. Comput.* **22**(6), 851–865 (2018)
 - 78. Miikkulainen, R., Forrest, S.: A biological perspective on evolutionary computation. *Nat. Mach. Intell.* **3**(1), 9–15 (2021)
 - 79. Mitchell, M.: Coevolutionary learning with spatially distributed populations. In: Computational Intelligence: Principles and Practice (2006)
 - 80. Moran, N., Pollack, J.: Coevolutionary neural population models. In: Artificial Life Conference Proceedings, pp. 39–46. MIT Press (2018)
 - 81. Mu, J., Zhou, Y., Cao, S., Zhang, Y., Liu, Z.: Enhanced evolutionary generative adversarial networks. In: 2020 39th Chinese Control Conference (CCC), pp. 7534–7539 (2020)
 - 82. Nguyen, A., Yosinski, J., Clune, J.: Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 427–436 (2015)
 - 83. Ohno, H.: Training data augmentation: An empirical study using generative adversarial net-based approach with normalizing flow models for materials informatics. *Appl. Soft Comput.* **86**, 105932 (2020)
 - 84. Olsson, B.: Co-evolutionary search in asymmetric spaces. *Inf. Sci.* **133**(3–4), 103–125 (2001)

85. Ostaszewski, M., Seredyński, F., Bouvry, P.: Coevolutionary-based mechanisms for network anomaly detection. *J. Math. Model. Algorithms* **6**(3), 411–431 (2007)
86. Pérez, E., Nesmachnow, S., Toutouh, J., Hemberg, E., O'Reilly, U.-M.: Parallel/distributed implementation of cellular training for generative adversarial neural networks. In: 2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 512–518. IEEE (2020)
87. Pollack, J.B., Blair, A.D.: Co-evolution in the successful learning of backgammon strategy. *Mach. Learn.* **32**(3), 225–240 (1998)
88. Popovici, E., Bucci, A., Wiegand, R.P., De Jong, E.D.: Coevolutionary Principles, pp. 987–1033. Springer, Berlin, Heidelberg (2012)
89. Qin, Y., Wang, Z., Xi, D.: Tree cyclegan with maximum diversity loss for image augmentation and its application into gear pitting detection. *Appl. Soft Comput.* **114**, 108130 (2022)
90. Rosin, C.D., Belew, R.K.: New methods for competitive coevolution. *Evol. Comput.* **5**(1), 1–29 (1997)
91. Rozière, B., Rakotonirina, N.C., Hosu, V., Rasoanaivo, A., Lin, H., Couprise, C., Teytaud, O.: Tarsier: evolving noise injection in super-resolution gans. In: 2020 25th International Conference on Pattern Recognition (ICPR), pp. 7028–7035 (2020)
92. Rozière, B., Teytaud, F., Hosu, V., Lin, H., Rapin, J., Zameshina, M., Teytaud, O.: Evogan: evolutionary generative adversarial networks. In: Asian Conference on Computer Vision (2020)
93. Rush, G., Tauritz, D.R., Kent, A.D.: Coevolutionary agent-based network defense lightweight event system (CANDLES). In: Proceedings of the Companion Publication of the 2015 on Genetic and Evolutionary Computation Conference, pp. 859–866. ACM (2015)
94. Sadeghi, K., Banerjee, A., Gupta, S.K.: A system-driven taxonomy of attacks and defenses in adversarial machine learning. *IEEE Trans. Emer. Topics Comput. Intell.* **4**(4), 450–467 (2020)
95. Schmiedlechner, T., Yong, I.N.Z., Al-Dujaili, A., Hemberg, E., O'Reilly, U.-M.: Lipizzaner: a system that scales robust generative adversarial network training. In: the 32nd Conference on Neural Information Processing Systems (NeurIPS 2018) Workshop on Systems for ML and Open Source Software (2018)
96. Service, T., Tauritz, D.: Increasing infrastructure resilience through competitive coevolution. *New Math. Nat. Comput.* **5**(02), 441–457 (2009)
97. Sims, K.: Evolving 3d morphology and behavior by competition. *Artif. Life* **1**(4), 353–372 (1994)
98. Sipper, M.: Evolved to Win. Lulu. com (2011)
99. Suarez-Tangil, G., Palomar, E., de Fuentes, J.M., Blasco, J., Ribagorda, A.: Automatic rule generation based on genetic programming for event correlation. In: Computational Intelligence in Security for Information Systems, pp. 127–134. Springer (2009)
100. Sun, F., Tao, Q., Hu, J., Liu, J.: Composite evolutionary gan for natural language generation with temper control. In: 2021 7th International Conference on Computer and Communications (ICCC), pp. 1710–1714 (2021)
101. Telikani, A., Tahmassebi, A., Banzhaf, W., Gandomi, A.H.: Evolutionary machine learning: a survey. *ACM Comput. Surv. (CSUR)* **54**(8), 1–35 (2021)
102. Tomassini, M.: Introduction to evolutionary game theory. In: Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation, pp. 877–890 (2014)
103. Toutouh, J., Esteban, M., Nesmachnow, S.: Parallel/distributed generative adversarial neural networks for data augmentation of covid-19 training images. In: Latin American High Performance Computing Conference, pp. 162–177. Springer (2020)
104. Toutouh, J., Hemberg, E., O'Reilly, U.-M.: Spatial evolutionary generative adversarial networks. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '19, New York, NY, USA, pp. 472–480. ACM (2019)
105. Toutouh, J., Hemberg, E., O'Reilly, U.-M.: Analyzing the components of distributed coevolutionary gan training. In: Bäck, T., Preuss, M., Deutz, A., Wang, H., Doerr, C., Emmerich, M.,

- Trautmann, H. (eds.) Parallel Problem Solving from Nature - PPSN XVI. pp, pp. 552–566. Springer International Publishing, Cham (2020)
- 106. Toutouh, J., Hemberg, E., O'Reilly, U.-M.: Data dieting in gan training. In: Iba, H., Noman, N. (eds.) Deep Neural Evolution: Deep Learning with Evolutionary Computation, pp. 379–400. Springer Singapore, Singapore (2020)
 - 107. Toutouh, J., Hemberg, E., O'Reilly, U.-M.: Re-purposing heterogeneous generative ensembles with evolutionary computation. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '2020, New York, NY, USA. Association for Computing Machinery (2020)
 - 108. Toutouh, J., O'Reilly, U.-M.: Signal propagation in a gradient-based and evolutionary learning system. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 377–385 (2021)
 - 109. Tygar, J.D.: Adversarial machine learning. *IEEE Internet Comput.* **15**, 4–6 (2011)
 - 110. Wang, C., Xu, C., Yao, X., Tao, D.: Evolutionary generative adversarial networks. *IEEE Trans. Evol. Comput.* **23**(6), 921–934 (2019)
 - 111. Wang, G.L.M., Thite, A., Talebi, R., D'Achille, A., Mussa, A.W., Zutty, J.: Evolving simgans to improve abnormal electrocardiogram classification. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion (2022)
 - 112. Wang, H., Won, D., Yoon, S.W.: An adaptive neural architecture optimization model for retinal disorder diagnosis on 3d medical images. *Appl. Soft Comput.* **111**, 107686 (2021)
 - 113. Wang, X., Miikkulainen, R.: Mdea: malware detection with evolutionary adversarial learning. In: 2020 IEEE Congress on Evolutionary Computation (CEC), pp. 1–8 (2020)
 - 114. Wilkerson, J.L., Tauritz, D.: Coevolutionary automated software correction. In: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, GECCO '10, New York, NY, USA, pp. 1391–1392. ACM (2010)
 - 115. Williams, N., Mitchell, M.: Investigating the success of spatial coevolution. In: Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation, pp. 523–530. ACM (2005)
 - 116. Wu, Z., He, C., Yang, L., Kuang, F.: Attentive evolutionary generative adversarial network. *Appl. Intell.* **51**, 1747–1761 (2020)
 - 117. Ying, G., He, X., Gao, B.-B., Han, B., Chu, X.: Eagan: efficient two-stage evolutionary architecture search for gans (2021) [arXiv:abs/2111.15097](https://arxiv.org/abs/2111.15097)
 - 118. Yu, X., Gen, M.: Introduction to Evolutionary Algorithms. Springer Science & Business Media (2010)
 - 119. Zhang, C., Ma, Y.: Ensemble Machine Learning: Methods and Applications. Springer (2012)
 - 120. Zhao, F., Lu, Y., Li, X., Wang, L., Song, Y., Fan, D., Zhang, C., Chen, X.: Multiple imputation method of missing credit risk assessment data based on generative adversarial networks. *Appl. Soft Comput.* **126**, 109273 (2022)
 - 121. Zheng, W., Gou, C., Yan, L., Wang, F.-Y.: Differential-evolution-based generative adversarial networks for edge detection. In: 2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW), pp. 2999–3008 (2019)

Part IV

Evolutionary Computation for Machine Learning

In which we explain how evolutionary computation methods can help improve and augment machine learning algorithms: For AutoML, model validation, explainable AI, and fair machine learning.

Chapter 14

Genetic Programming as an Innovation Engine for Automated Machine Learning: The Tree-Based Pipeline Optimization Tool (TPOT)



Jason H. Moore, Pedro H. Ribeiro, Nicholas Matsumoto, and Anil K. Saini

Abstract One of the central challenges of machine learning is the selection of methods for feature selection, feature engineering, and classification or regression algorithms for building an analytics pipeline. This is true for both novices and experts. Automated machine learning (AutoML) has emerged as a useful approach to generate machine learning pipelines without the need for manual construction and evaluation. We review here some challenges of building pipelines and present several of the first and most widely used AutoML methods and open-source software. We present in detail the Tree-based Pipeline Optimization Tool (TPOT) that represents pipelines as expression trees and uses genetic programming (GP) for discovery and optimization. We present some of the extensions of TPOT and its application to real-world big data. We end with some thoughts about the future of AutoML and evolutionary machine learning.

14.1 Challenges of Machine Learning

A central goal of machine learning is to make predictions from multiple measured variables or features using mathematical functions that accurately model underlying patterns in the data. Ideally, those predictions would be accompanied by human-understandable reasoning. Machine learning excels over other analytical approaches when the relationship between the feature patterns and the outcome is nonlinear and/or heterogeneous among different subsets of the instances. Complex patterns in big data are often encountered in real-world problems such as predicting health outcomes or forecasting economic measures.

Machine learning algorithms fit internal parameters to minimize a specified loss function between their predictions and the observed target feature. Models often come in the form of composite pipelines of several steps, including feature selection,

J. H. Moore (✉) · P. H. Ribeiro · N. Matsumoto · A. K. Saini
Cedars-Sinai Medical Center, Los Angeles, CA, USA
e-mail: jason.moore@csmc.edu

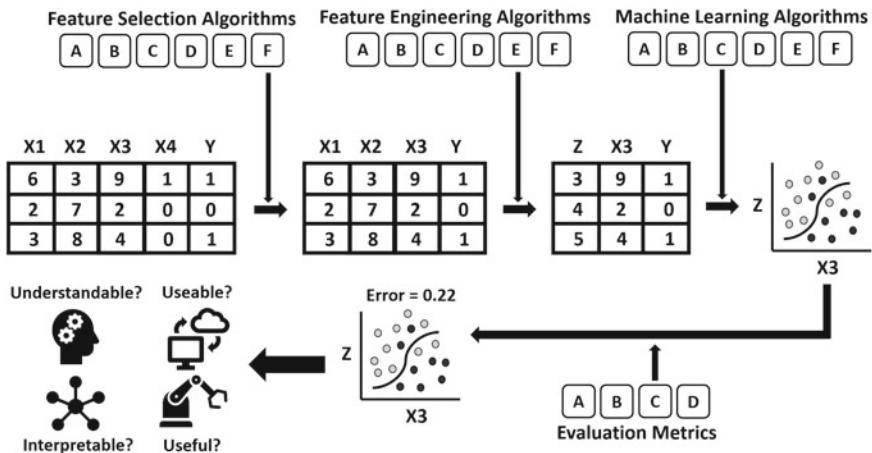


Fig. 14.1 Overview of the many decisions that need to be made when choosing feature selection, feature engineering, machine learning, and model evaluation algorithms. Also shown are the components of model explainability that are often necessary for real-world problem-solving and model deployment. The first three steps are typically targeted for automation

feature pre-processing, feature engineering, and at least one classifier or regressor algorithm such as Decision Trees or Linear Regression. The primary challenge of a data scientist is to discover an optimal set of algorithms and hyperparameters that balance predictive performance and interpretability. This often involves a lengthy and time-consuming process of trial and error that AutoML seeks to automate.

We review here several challenges of designing a machine learning pipeline and then discuss genetic programming (GP) as an innovation engine for machine learning with big data. Figure 14.1 provides an overview of these challenges. More detail on each challenge is provided below.

14.1.1 Designing a Machine Learning Pipeline

Machine learning pipelines are a sequence of steps by which data is analyzed to arrive at a prediction. There are numerous algorithms and methods that could be included in a pipeline including data cleaning, feature selection, feature engineering, and classification and regression methods. Each can have a wide variety of different hyperparameters, and the models generated by these pipelines can be evaluated and explained in many different ways. We briefly review below some of the components and decisions that go into pipeline construction, application, evaluation, and deployment.

14.1.1.1 Data Cleaning and Quality Control

One of the challenges of machine learning is that the data do not always present in a format that is ideal for analysis. Data may come with noise, bias, missing values, outliers, unbalanced labels, and other issues. Data cleaning and quality control is the process of identifying and correcting errors and biases in data. Each of these steps may require different algorithms or statistical methods, and success or failure can have a big impact on the quality of the machine learning results [1].

14.1.1.2 Feature Selection

A significant challenge comes from applying machine learning algorithms to big data where there can be thousands or millions of features. For example, it is common in human genetics to measure more than one million DNA sequence variations in genome-wide association studies (GWAS) to identify the risk factor for common diseases. A large number of features can make it harder to detect the signal from noise, increase the probability of overfitting, and add enormous computational complexity raising concerns about compute time and carbon footprint. Feature selection is an integral first step to addressing these issues in machine learning. Fewer features can yield simpler, more interpretable models, but this must be balanced against the potential cost of predictive performance from training with incomplete data.

Feature selection can take several different forms. The quality or characteristics of features can be used to prune out those that are not of sufficient quality (i.e. too many missing values), lack information (i.e. every instance has the same value), or contain redundant information (i.e. correlated). Domain knowledge can be used to select features most likely to be important based on prior literature. For example, a genetic study of diabetes could focus on DNA sequence variations located in or near genes related to insulin or glucose metabolism as a way to reduce the feature set. Algorithms can estimate the importance of features from preliminary machine learning results or special algorithms such as ReliefF [2] and select a subset for further investigation. Some machine learning algorithms, such as deep learning neural networks or lasso-based logistic regression, can perform feature selection from within the algorithm. This can eliminate the need for this step. Others such as random forests can select features using feature importance scores that assess the relative contribution of each feature to the predictive accuracy. Feature selection is an art in and of itself adding to the complexity of building a machine learning pipeline.

14.1.1.3 Feature Engineering

Feature Engineering is the process of transforming raw data for use in machine learning. Data transformations can increase performance, interpretability, or, in some instances, both. Machine learning algorithms may sometimes be able to account for these issues; however, this often increases the complexity of the models and lowers

interpretability. Feature Engineering can handle these issues more clearly, allowing the algorithm to be focused on the prediction. As with feature selection, care must be taken not to overfit the data if the predicted outcome is used to engineer the features. Validation data may be needed to prevent overfitting in this step.

Transformations can leverage expert knowledge or be done automatically by specialized algorithms. When there is expert knowledge of the interactions of specific features, data scientists can manually transform the data into new features that better represent a biological or physical process being studied. Automated dimensionality reduction methods such as principal component analysis (PCA) or deep learning autoencoders may effectively isolate signals from noisy data. Other feature transformations may be based on simple arithmetic that allows simpler algorithms to model more complex functions. For example, polynomial transformations allow linear regression to quickly model nonlinear trends. Good feature engineering can also reduce the total number of features being studied leading to computational efficiency.

14.1.1.4 Classifier or Regressor Selection

One of the most difficult challenges for experts and non-experts alike is choosing a machine learning model in the form of a classifier (for binary outcomes) or regressor (for continuous outcomes) and setting its hyperparameters. There are dozens of commonly used methods, and each models the features-outcome relationship differently. For example, some methods are better at modeling linear patterns, and some are better with nonlinearities. Some methods produce more explainable models, while some are more computationally efficient. Compounding this problem is that one rarely knows what the patterns in the data are until after extensive analysis and evaluation. Furthermore, each method often has multiple hyperparameters that govern how the algorithm works.

14.1.1.5 Assessing Predictive Performance

Key to any machine learning algorithm is the assessment of model quality. At its core, machine learning seeks to make accurate predictions, as measured by a loss function when applied to a training dataset. Evaluating predictive performance is challenging. Machine learning models can easily memorize entire datasets, leading to perfect in-sample prediction. However, this is often accomplished through an overly large and complex model that memorizes the particularities of the specific dataset rather than finding broader trends. The result is a model that does not generalize well to new data. This is a phenomenon known as overfitting.

Cross-validation is a popular algorithm to estimate the out-of-sample error by averaging the out-of-sample scores on k different training/validation splits of the data. However, evaluating a large number of pipelines can lead to overfitting the cross-validation score itself due to the correlation between the training splits of the data and multiple looks at the testing data. Therefore, the spectrum of models with

low to high cross-validation scores may range from underfit (underperforming) to a good fit to overfit. The data scientist's job is to make this distinction and select an appropriately fit model. Ideally, predictive performance would be assessed on independent data once a final model has been selected. This is sometimes referred to as the testing data.

14.1.1.6 Multi-objective Optimization

Additional measures of model quality beyond predictive accuracy might be needed for some problems or use cases. For example, there has been a growing push for interpretable machine learning models that provide human-understandable reasoning for their predictions. For example, models that predict health outcomes from medical data would be more desirable if the reasoning behind them identifies effective interventions. Improvement in one metric often results in a trade-off, causing performance to decline in another. For example, simpler models may be more interpretable, though usually at the cost of predictive performance. Geng et al. [3] survey objective and subjective measures of model quality. Specifically, they review the idea of model interestingness, which can be domain- and user-specific. Several algorithms, such as Pareto optimization [4] have been developed to tune models toward multiple quality measures efficiently. The selection of the loss function and other measures of model quality can have a big impact on the value of a model.

14.1.1.7 Model Explainability

The growing field of explainable artificial intelligence (XAI) or machine learning aims to develop models that produce human-understandable reasoning for predictions [5]. While the underlying algorithms powering machine learning are often simple and easy to understand with the right mathematical knowledge, their models can be highly complex and inscrutable. These models may produce accurate predictions, but the basis for those decisions is often obscured. Understanding the reasoning within the models and their domain-specific application opens the door to new insights and applications. Combi et al. [5] decompose XAI into model *interpretability, understandability, usability, and usefulness*. We review their definitions below.

The interpretability of a model refers to the ability of the user to have a general understanding or intuition of how a particular decision came about, without necessarily needing to know the exact mathematics leading to the solution. For example, a deep learning model that identifies cancer in MRI images may highlight regions that inform their decisions. From this, a doctor may understand the general patterns the model looks for, without necessarily understanding the complex math behind defining such patterns. In addition, there are methods such as permutation feature importance scores that assess the relative contribution of each feature to the final prediction that can be applied more generally to any machine learning model. This

allows the user to say which features the model focuses on, which may lead to additional hypotheses.

The understandability of a model refers to the user's ability to comprehend and follow the inner workings and mathematics of the underlying algorithm. For example, a human can easily read a decision tree and understand the flow of logic used to arrive at a decision or prediction. However, humans are generally not wired to understand the abstract prediction logic hidden away in the matrix transformations of deep learning models. A model can be understandable but not interpretable. While decision trees can be understandable, they may not be interpretable when the complexity is too high for humans to grasp intuitively. Understandability can be particularly important in the medical domain, where these can be a high cost in terms of patient suffering for incorrect decisions. Clinicians can fact-check the reasoning of the model with their own expert knowledge to make a more informed decision. Understanding a model can lead to new hypotheses and perhaps interventions to perturb the system in a beneficial way. For example, a decision tree model that predicts disease risk from genetic data could identify rules not previously considered by science which could lead to the hypotheses about new gene-based drug targets for disease treatment.

Combi et al. [5] also discuss usability, which refers to the practical ability to easily implement the solution in the real world. There are several factors that inform whether or not a model could be practically implemented. These include, but are not limited to, the practicality of collecting, digitizing, and inputting the required data; the financial cost of installing, running, and maintaining the solution; the ease of training users to utilize the solution and whether it can be incorporated into their workflow. A technology's usefulness may be dependent on context and relative to alternatives and problems at hand.

The final component of XAI is usefulness. This refers to whether the technology meets a user's need. In machine learning, this often means accurate, or at least actionable, predictions. The utility of a model is often enhanced if it is also interpretable, understandable, and usable. Interpretability and understandability both can lead trust to the underlying predictions leading to more efficient evaluation and decision-making as well as providing insights into new directions. Usability is required for people to actually use the technology in the first place, but strong usability can also make workflows more efficient.

14.2 Automated Machine Learning

Automated machine learning or AutoML arose after 2010 to address the complexity of choosing methods, tuning hyperparameters, and building analytical pipelines [6]. One of the first AutoML methods was Auto-WEKA [7] which was built on top of the popular WEKA machine learning package [8]. Auto-WEKA uses a Bayesian optimization algorithm to choose the best machine learning algorithm and hyperparameter settings from the WEKA software package. Auto-WEKA has been used, for

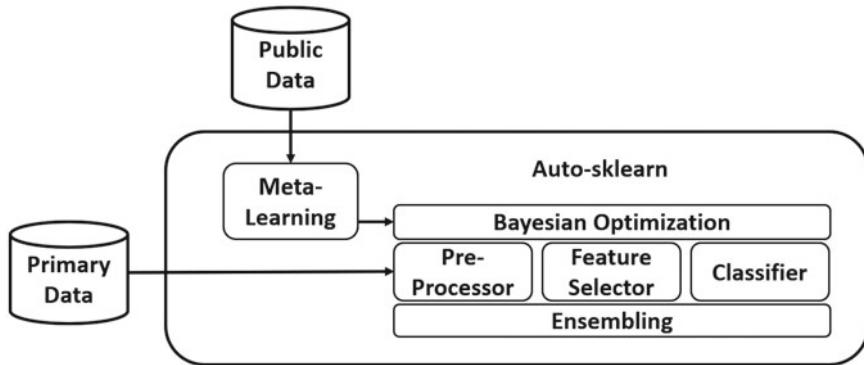


Fig. 14.2 Overview of the Auto-sklearn method that includes a meta-learning algorithm trained on public data and a machine learning pipeline with pre-processor, feature selector, and classifier components. The pipelines are generated using a Bayesian optimization algorithm with prior information from the meta-learner. Ensembles of pipelines are possible

example, to predict intracerebral hemorrhage in patients with features derived from demographics, laboratory tests, and imaging [9].

Another early and widely used AutoML method is Auto-sklearn [10] which uses the scikit-learn machine learning library [11]. Like Auto-WEKA, Auto-sklearn uses a Bayesian optimization algorithm to select a pipeline consisting of a data pre-processing algorithm, a feature selector, and a machine learning classifier (Fig. 14.2). This was the first AutoML method to build a pipeline that is enhanced by meta-learning and ensemble classification. Interestingly, Auto-sklearn uses results from the analysis of a large number of public datasets as meta-data for optimizing its pipelines on new data. Auto-sklearn has been widely used on problems such as prioritization of social media posts about suicide [12].

The third early and widely used method is the Tree-based Pipeline Optimization Tool (TPOT), which represents machine learning pipelines as expression trees with discovery and optimization by genetic programming [13–15]. The TPOT approach also uses the scikit-learn library as a source of algorithms. However, the expression tree representation gives it more flexibility for pipeline construction. We review this approach in detail below.

14.3 The Tree-Based Pipeline Optimization Tool (TPOT)

Central to TPOT is the representation of machine learning pipelines as expression trees. The TPOT discovery and optimization algorithm comes from a subdomain of AI called evolutionary computation, which derives its motivation from evolution by natural selection in biology. Specifically, the application of evolutionary computation to computer programs is referred to as genetic programming or GP [16]. A benefit

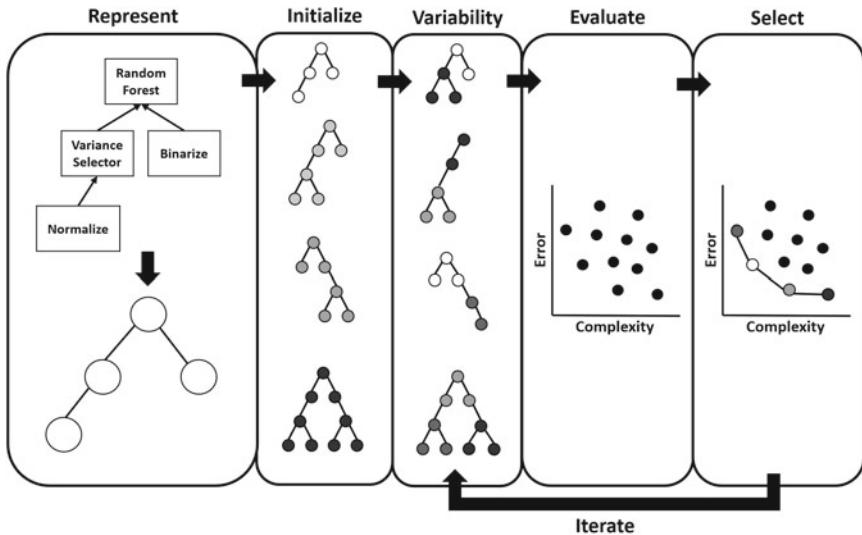


Fig. 14.3 An overview of the TPOT algorithm. The first step is to represent machine learning pipelines using expression trees. Pipeline trees are initialized randomly, diversified using several variation operators, evaluated, and selected using quality metrics such as an error-based loss function and the complexity of the pipelines. This process is iterated until a stopping criterion is reached

of developing TPOT using the GP framework is that there is an extensive base of literature exploring a variety of methods and strategies for applying GP to complex problems. Further, expression trees are easily integrated into a GP framework. Key to TPOT is the use of Pareto optimization [4] which allows pipelines to be evaluated using multiple objectives such as predictive accuracy and pipeline complexity. TPOT is implemented using the open-source Distributed Evolutionary Algorithms in Python (DEAP) software package [17].

The basic components of TPOT are illustrated in Fig. 14.3. The algorithm starts by generating N expression tree-based pipelines randomly from a set of scikit-learn operators (e.g. feature selectors and classifiers) and their hyperparameter settings. New pipelines are generated using variation operators that mutate pipeline components and recombine branches between pipelines. The N old and the N new pipelines are evaluated, and the best N are selected to move forward. The variation, evaluation, selection, and iteration steps continue until a stopping criterion is reached. This is usually set to be G generations or iterations of the algorithm (e.g. $G = 100$ or 1000).

14.3.1 Pipeline Representation

One of the key differences between TPOT and other methods is the representation of machine learning pipelines using expression trees. Tree-based data structures are

ideal for this purpose, given the stepwise data processing that occurs in a pipeline. Here, the tree nodes are selected from feature selection, feature engineering, and machine learning algorithms from the scikit-learn library. The terminals of the tree represent the hyperparameters of the various algorithms and data inputs. At evaluation, a scikit-learn pipeline is constructed by initializing models with their respective hyperparameters and using the feature unions to create the branching structure. To execute a tree, the data are passed into the leaves of the tree and propagated through the nodes of the tree up through the root node, which serves as the final classifier or regressor. We summarize in Fig. 14.3 the steps of the TPOT algorithm. Each step is described in more detail below.

14.3.2 Pipeline Initialization

The first step of TPOT is to initialize a set of N pipelines where N is referred to as the population size (e.g. $N = 100$ or 1000). TPOT has a default set of algorithms from the scikit-learn library. This can be modified or constrained using a configuration file. Each expression tree starts with a machine learning algorithm (e.g. classifier or regressor) at the root node. This ensures that the pipeline's output is a set of predicted values that can be compared to observed values using a loss function. Each child node is selected from the set of all allowable methods, including feature transformation, feature selection, feature engineering, and machine learning algorithms (i.e. the function set). The terminals of each node are used to specify randomly selected hyperparameter values (i.e. the terminal set). Each of the N trees a to b layer trees is generated randomly for the initial population from a distribution of possibilities.

14.3.3 Pipeline Variation

The first step in each generation is to generate new pipelines from the current population. Just as mutation and recombination generate variation for natural selection during evolution, TPOT uses similar operators to generate variation in machine learning pipelines. This is done in a few different ways (Fig. 14.4). New individuals in each population are generated either by mutation of an existing tree, with probability M , or recombination through crossover of selected subtrees (i.e. branches) or hyperparameters between two individuals with probability R . A mutation is equally likely to insert, remove, or replace nodes. In crossover, subtrees of two trees are randomly swapped.

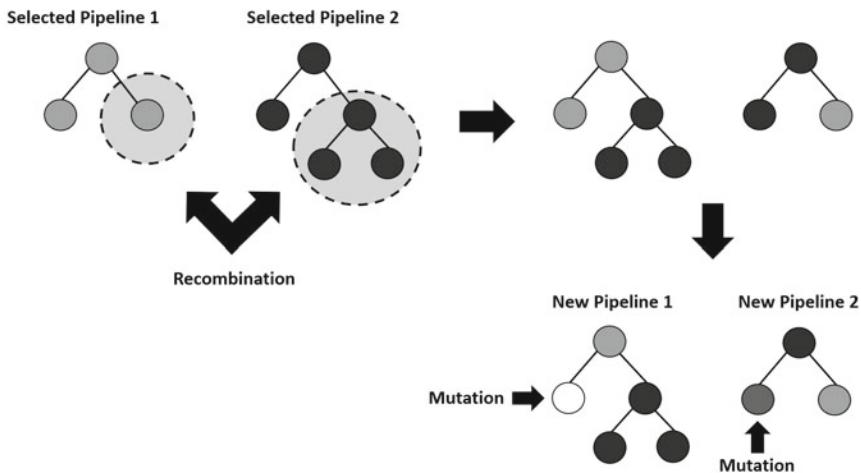


Fig. 14.4 Overview of the process by which variation is introduced into TPOT pipelines via recombination of subtrees and mutation of nodes. Mutation and recombination occur separately on different trees in the same generation

14.3.4 Pipeline Evaluation

Each tree is evaluated based on two objectives. First, a standard loss function is used to evaluate the predictive error of classifiers using k -fold cross-validation. Second, the complexity of the expression tree is approximated by counting the number of nodes in a tree. Other complexity measures could be developed and evaluated that take into account the complexity of the individual algorithms and their hyperparameters. For example, different weights could be given to machine learning methods such as XGBoost that have higher complexity than simpler methods such as decision trees or logistic regression. Assessing complexity allows TPOT to consider simpler pipelines that might be more interpretable, less likely to overfit the data, and more likely to generalize to independent data. Different or additional criteria could be developed within the TPOT software allowing for customized multi-objective optimization.

14.3.5 Pipeline Selection

Key to TPOT is the method by which the best trees or pipelines in the current population are selected to pass into the next population. An ideal model has low complexity but high performance. In practice, there is often a trade-off between interpretability and performance, with simpler models sacrificing some performance. Similarly, there is a trade-off between generalizability and overfitting the objective function performance. Cross-validation addresses this to some degree but is not perfect (TPOT

models can become overly complex and overfit the cross-validation score itself). This is of particular concern when exploring many pipelines. Left unrestricted, TPOT would form overly complex pipelines that greatly overfit the data. If too limited, TPOT may not explore better solutions.

As the balance for interpretability versus performance is subjective and domain-specific, TPOT attempts to optimize a Pareto front of non-dominated models defined by both performance and complexity. During training, TPOT selects the set of non-dominated models from a Pareto front using the non-dominated sorting genetic algorithm II or NSGA-II [18]. These models will then be mutated or recombined in the next generation. Other selection methods, such as lexicase selection [19], should be explored.

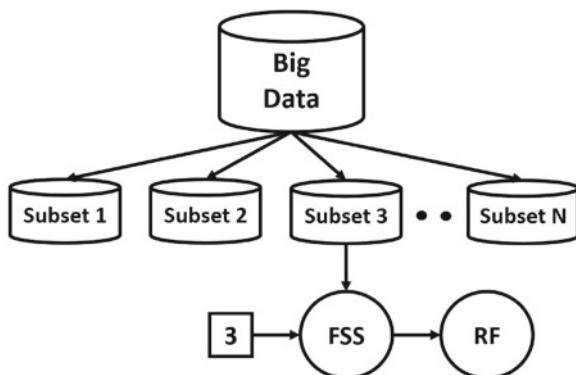
14.3.6 Picking the Best Model

There are several ways to pick the best model following a TPOT run. The easiest is to return the pipeline from the final iteration with the best loss (e.g. predictive accuracy) determined by cross-validation. Another approach is to select the Pareto optimal models and select the one that balances accuracy and complexity according to the wishes of the user.

14.3.7 Scaling TPOT to Big Data

One of the challenges of AutoML is the computational complexity of generating and evaluating millions of machine learning pipelines, which is compounded by big data with many features and instances. Le et al. [20] addressed this with two extensions to TPOT. First, they introduced a template option to specify a fixed linear pipeline structure. The template option can be used to constrain TPOT to just a feature selector and classifier, for example, which would execute faster than more complex trees. Second, they introduced a feature set selector (FSS) operator (see Fig. 14.5), which functions as an expert knowledge feature selector. Here, expert knowledge is used to group features into S subsets. The FSS operator includes a hyperparameter pointing to one of the subsets that is then used to select a subset of the data for the next operator in the tree. Combining the simple template tree structure with smaller feature sets can greatly improve the execution time of TPOT, which is important for working with big data and minimizing the carbon footprint.

Fig. 14.5 The inclusion of a Feature Set Selector (FSS) allows TPOT to select a subset of features for analysis in a pipeline. In the example pipeline shown, the FSS operator has a hyperparameter set to 3 which in turn selects data subset 3. This subset of the data is then passed to the Random Forest (RF) algorithm to develop a predictive model



14.3.8 Neural Networks with TPOT

Deep learning has popularized neural networks (NN) for analyzing big data. An area of current investigation is whether methods such as TPOT can be used to build machine learning pipelines by combining simple NN classifiers or regressors. Romano et al. [21] specifically compared the performance of TPOT, NNs, and an implementation of TPOT with only shallow NN classifiers referred to as TPOT-NN (see Fig. 14.6) applied to several publicly available datasets. This study showed that TPOT-NN performed better on several datasets but did not ever perform worse than standard TPOT, which had access to the full set of scikit-learn classifiers. It is interesting to speculate about the maximum deep learning model complexity TPOT-NN could approximate because it has access to feature transformers, feature selectors, and feature engineering algorithms. It is commonly believed that the first layer or two of a deep learning NN is likely performing some of these functions. A high-performing

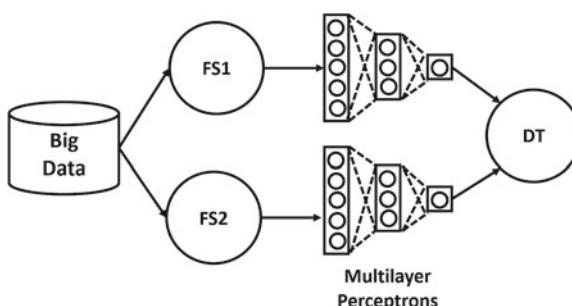


Fig. 14.6 A hypothetical TPOT-NN pipeline. Here, two different Feature Selector operators (FS1 and FS2) select different subsets of features that are then passed to multilayer perceptron neural networks. The predictions made by these feed-forward neural networks are passed to a decision tree that makes the final prediction as a type of ensemble

TPOT-NN pipeline could be easier to interpret and/or understand because the NN classifiers would be simpler and focus more on the classification task. This will be an exciting area to explore in future studies.

14.3.9 Real-World Applications

There have been a number of real-world applications of TPOT with an emphasis in the biomedical domain, especially genetics and genomics [22]. These include predicting depression using genomic data [20], coronary artery disease using metabolomics data [23], schizophrenia using genomics data [24], predicting renal cell carcinoma grade using radiologic images [25], childhood dental carries using metabolomics data [26], and coronary artery disease using genetics data [27]. We focus here on the study by Manduchi et al. [27] that illustrates several of the concepts reviewed above.

Manduchi et al. [27] studied over 19,000 subjects with coronary artery disease (CAD) and more than 320,000 health controls ascertained from the UK Biobank data resource. This resource provided more than one million measured genetic variants from across the human genome. The goal of the study was to develop a predictive model of CAD using more than one million genetic features. The authors faced several machine learning challenges with these big data. First, the CAD class (disease vs. no disease) was severely imbalanced, which can lead to predictive accuracies that favor the larger class. The authors randomly downsampled the larger class to balance the dataset. A total of 50 datasets were created by repeating this process 50 times to prevent chance partitions of the data. Second, the size of the dataset (more than one million features) presents computational and carbon footprint challenges for AutoML. The authors focused their analysis by creating smaller sets of features using expert knowledge about genes believed to be promising drug targets for CAD. Finally, the sample size permitted a 5-fold cross-validation to assess generalizability of the models along with holdout datasets used to assess the predictive ability of final models.

Figure 14.7 shows one of the best pipelines generated by TPOT for predicting CAD. In the first step of the pipeline, a feature selector based on feature percentile is selected for reducing the total number of features. The second operator employs a variance threshold as an additional feature selector. The third operator uses a stacking estimator with a stochastic gradient descent classifier to engineer a new feature which is then added back to the feature list. The fourth operator carries out an additional round of feature selection by employing an extra tree classifier and using feature important scores to select a subset. This final list of selected and engineered features is then passed to the final operator, which uses an extra tree classifier to do the final classification of subjects with and without CAD. This pipeline was statistically significant ($p < 0.05$) based on a permutation test and had a testing accuracy on holdout data of 0.55, which is in line with other predictive studies using genetics-based features. Importantly, TPOT was able to automatically identify a machine learning pipeline consisting of feature selection, feature engineering, and

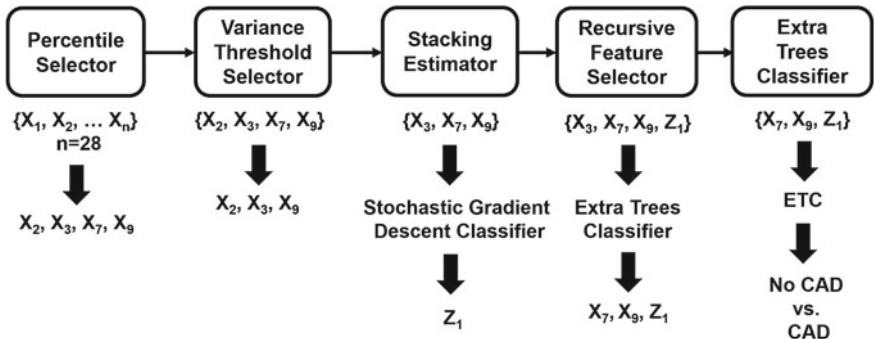


Fig. 14.7 A TPOT-generated machine learning pipeline for predicting risk of CAD. This pipeline contains a combination of feature selector (first, second, and fourth), feature engineering (third), and classification (fifth) algorithms

classification algorithms. It is unlikely that any human user would have constructed a pipeline like this manually for these data.

Interpretation of this model using a game-theoretic approach called Shapley values [28] estimated that different features contributed differently to the predictive values for different subsets of human subjects. This suggests a phenomenon known as genetic heterogeneity, where different subjects have different underlying risk factors for the same disease representing a hypothesis to be investigated in future studies. An advantage of machine learning over linear methods such as regression is that they can detect and model complex relationships between features and outcomes such as this heterogeneity pattern. TPOT was able to find a pipeline that could model these relationships automatically.

14.4 Future Directions

We have reviewed several challenges associated with building a machine learning pipeline. Some of these challenges can be addressed through AutoML algorithms designed to select optimal combinations of algorithms and hyperparameter settings for a particular dataset [6]. Early and popular methods include Auto-WEKA [7], Auto-sklearn [10], and TPOT [13, 14].

The TPOT algorithm is unique in that it represents machine learning pipelines as expression trees. The ability to represent pipelines as trees and to generate new trees using genetic-inspired variation operators such as mutation for changing nodes and recombination for swapping subtrees provides a powerful platform for building complex machine learning pipelines. Despite these advantages, TPOT could be improved in a number of different ways. We briefly mention some of these ideas below.

First, although the tree-based representation has a number of advantages, there are other representations used in GP that should be explored. For example, a graph-

based data structure might reduce the computational complexity of a pipeline by passing the same transformation on to multiple different classifiers. A graph-based representation might also be more amenable to ensembling which requires the outputs of multiple classifiers to be processed.

Second, there is a rich literature base for GP with numerous creative and effective ways to initialize populations, generate variation, evaluate solutions, and perform selection. These are too numerous to list here. However, the Genetic Programming Theory and Practice workshop has explored a number of advances for solving hard problems [29]. The most promising of these GP advances should be evaluated for TPOT.

Third, one of the promises of AutoML is that it is able to bring advanced machine learning methods and technology to non-experts. Although TPOT can build a pipeline from scratch, it does not include a graphic user interface (GUI). An intuitive GUI designed for non-experts could be important for bringing this tool and others to the masses. This will help ensure that everyone who can benefit from machine learning is able to do so with as few barriers as possible. Aliro AI is an example of a user-friendly AutoML [30].

Fourth, TPOT is not directly aware of domain-specific knowledge. Current approaches include using manual selection of features as input for TPOT or the Feature Set Selector mentioned above. Some real-world problems may require a more intimate relationship between a knowledgebase and the AutoML so that a method such as TPOT can automatically use the knowledge to select features, select models, design pipeline architectures, and assist with explainability. There is currently little work in this area across AutoML methods, and domains such as biomedicine could greatly benefit.

Fifth, TPOT does not include any advanced tools for explainability that are designed for non-experts. As mentioned above, connection to a knowledgebase could help with interpretation and understanding by providing a domain basis for the results being considered. Automation of XAI for pipelines generated by TPOT could greatly improve their usefulness.

Finally, TPOT uses GP to discover and optimize pipelines. However, there are other areas in which GP can be integrated into the machine learning process. For example, the Feature Set Selector in TPOT requires users to predefined subsets, and then it randomly selects between them. Instead, the FSS could use GP to dynamically create subsets by crossing over features with other pipelines in a process. TPOT may also benefit from the inclusion of GP-based classification or regression methods such as symbolic discriminant analysis [31] or symbolic regression [16]. It would be interesting to provide scikit-learn compatible GP methods or to include arithmetic operators individually to TPOT as a complement to the many other methods available.

Automated machine learning is in its infancy, having been an active area of investigation for less than 10 years. These methods show tremendous promise for accelerating discovery using machine learning pipelines. Genetic programming is a powerful innovation engine for building AutoML pipelines as has been demonstrated with TPOT. Its flexibility with solution representation through expression trees and genetic operators makes it a tool that could be applied to other problems in this space.

References

1. Chicco, D., Oneto, L., Tavazzi, E.: Eleven quick tips for data cleaning and feature engineering. *PLoS Comput. Biol.* **18**, e1010718 (2022)
2. Urbanowicz, R.J., Meeker, M., La Cava, W., Olson, R.S., Moore, J.H.: Relief-based feature selection: introduction and review. *J. Biomed. Inform.* **85**, 189–203 (2018)
3. Geng, L., Hamilton, H.J.: Interestingness measures for data mining: a survey. *ACM Comput. Surv.* **38** (2006)
4. Smits, G.F., Kotanchek, M.: Pareto-front exploitation in symbolic regression. In: O'Reilly, U.-M., Yu, T., Riolo, R., Worzel, B. (eds.) *Genetic Programming Theory and Practice II*. pp. 283–299 (2006)
5. Combi, C., Amico, B., Bellazzi, R., Holzinger, A., Moore, J.H., Zitnik, M., Holmes, J.H.: A manifesto on explainability for artificial intelligence in medicine. *Artif. Intell. Med.* **133**, 102423 (2022)
6. Hutter, F., Kotthoff, L., Vanschoren, J. (eds.): *Automated Machine Learning: Methods, Systems, Challenges*. Springer International Publishing (2019)
7. Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In: *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 847–855. ACM, New York, NY, USA (2013)
8. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The WEKA data mining software: an update. *ACM SIGKDD Explor. Newsl.* **11**, 10–18 (2009)
9. Wang, H.-L., Hsu, W.-Y., Lee, M.-H., Weng, H.-H., Chang, S.-W., Yang, J.-T., Tsai, Y.-H.: Automatic machine-learning-based outcome prediction in patients with primary intracerebral hemorrhage. *Front. Neurol.* **10**, 910 (2019)
10. Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., Hutter, F.: Efficient and robust automated machine learning. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) *Advances in Neural Information Processing Systems*, vol. 28. pp. 2962–2970. Curran Associates, Inc. (2015)
11. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E.: Scikit-learn: machine learning in Python. *J. Mach. Learn. Res.* **12**, 2825–2830 (2011)
12. Howard, D., Maslej, M.M., Lee, J., Ritchie, J., Woollard, G., French, L.: Transfer learning for risk classification of social media posts: model evaluation study. *J. Med. Internet Res.* **22**, e15371 (2020)
13. Olson, R.S., Urbanowicz, R.J., Andrews, P.C., Lavender, N.A., Kidd, L.C., Moore, J.H.: Automating biomedical data science through tree-based pipeline optimization. In: Squillero, G., Burelli, P. (eds.) *Applications of Evolutionary Computation*, pp. 123–137. Springer International Publishing, Cham (2016)
14. Olson, R.S., Bartley, N., Urbanowicz, R.J., Moore, J.H.: Evaluation of a tree-based pipeline optimization tool for automating data science. In: *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pp. 485–492. ACM, New York, NY, USA (2016)
15. Olson, R.S., Moore, J.H.: TPOT: a tree-based pipeline optimization tool for automating machine learning. In: Hutter, F., Kotthoff, L., Vanschoren, J. (eds.) *Automated Machine Learning: Methods, Systems, Challenges*, pp. 151–160. Springer International Publishing, Cham (2019)
16. Koza, J.R.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA (1992)
17. Fortin, F., De Rainville, F., Gardner, M.A., Parizeau, M., Gagné, C.: DEAP: evolutionary algorithms made easy. *J. Mach. Learn. Res.* **13**, 2171–2175 (2012)
18. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* **6**, 182–197 (2002)

19. Helmuth, T., McPhee, N.F., Spector, L.: Lexicase selection for program synthesis: a diversity analysis. In: Riolo, R., Worzel, W.P., Kotanchek, M., Kordon, A. (eds.) *Genetic Programming Theory and Practice XIII*, pp. 151–167. Springer International Publishing, Cham (2016)
20. Le, T.T., Fu, W., Moore, J.H.: Scaling tree-based automated machine learning to biomedical big data with a feature set selector. *Bioinforma. Oxf. Engl.* **36**, 250–256 (2020)
21. Romano, J., Le, T., Fu, W., Moore, J.: TPOT-NN: augmenting tree-based automated machine learning with neural network estimators. *Genet. Program. Evolvable Mach.* 1–21 (2021)
22. Manduchi, E., Romano, J.D., Moore, J.H.: The promise of automated machine learning for the genetic analysis of complex traits. *Hum. Genet.* **141**, 1529–1544 (2022)
23. Olenko, A., Kofink, D., Lytykäinen, L.-P., Nikus, K., Mishra, P., Kuukasjärvi, P., Karhunen, P.J., Kähönen, M., Laurikka, J.O., Lehtimäki, T., Asselbergs, F.W., Moore, J.H.: Model selection for metabolomics: predicting diagnosis of coronary artery disease using automated machine learning. *Bioinforma. Oxf. Engl.* **36**, 1772–1778 (2020)
24. Manduchi, E., Fu, W., Romano, J.D., Ruberto, S., Moore, J.H.: Embedding covariate adjustments in tree-based automated machine learning for biomedical big data analyses. *BMC Bioinform.* **21**, 430 (2020)
25. Purkayastha, S., Zhao, Y., Wu, J., Hu, R., McGirr, A., Singh, S., Chang, K., Huang, R.Y., Zhang, P.J., Silva, A., Soulén, M.C., Stavropoulos, S.W., Zhang, Z., Bai, H.X.: Differentiation of low and high grade renal cell carcinoma on routine MRI with an externally validated automatic machine learning algorithm. *Sci. Rep.* **10**, 19503 (2020)
26. Heimisdóttir, L.H., Lin, B.M., Cho, H., Olenko, A., Ribeiro, A.A., Simon-Soro, A., Roach, J., Shungin, D., Ginnis, J., Simancas-Pallares, M.A., Spangler, H.D., Zandoná, A.G.F., Wright, J.T., Ramamoorthy, P., Moore, J.H., Koo, H., Wu, D., Divaris, K.: Metabolomics insights in early childhood caries. *J. Dent. Res.* **100**, 615–622 (2021)
27. Manduchi, E., Le, T.T., Fu, W., Moore, J.H.: Genetic analysis of coronary artery disease using tree-based automated machine learning informed by biology-based feature selection. *IEEE/ACM Trans. Comput. Biol. Bioinform.* **19**, 1379–1386 (2022)
28. Lundberg, S.M., Lee, S.-I.: A unified approach to interpreting model predictions. In: *Advances in Neural Information Processing Systems*. Curran Associates, Inc. (2017)
29. Sipper, M., Moore, J.H.: Genetic programming theory and practice: a fifteen-year trajectory. *Genet. Program. Evolvable Mach.* **21**, 169–179 (2020)
30. La Cava, W., Williams, H., Fu, W., Vitale, S., Srivatsan, D., Moore, J.H.: Evaluating recommender systems for AI-driven biomedical informatics. *Bioinforma. Oxf. Engl.* **37**, 250–256 (2021)
31. Moore, J.H., Parker, J.S., Olsen, N.J., Aune, T.M.: Symbolic discriminant analysis of microarray data in autoimmune disease. *Genet. Epidemiol.* **23**, 57–69 (2002)

Chapter 15

Evolutionary Model Validation—An Adversarial Robustness Perspective



Inês Valentim, Nuno Lourenço, and Nuno Antunes

Abstract When building Machine Learning models, either manually or automatically, we need to make sure that they are able to solve the task at hand and generalize, i.e., perform well on unseen data. By properly validating a model and estimating its generalization performance, not only do we get a clearer idea of how it behaves but we might also identify problems (e.g., overfitting) before they lead to significant losses in a production environment. Model validation is usually focused on predictive performance, but with models being applied in safety-critical areas, robustness should also be taken into consideration. In this context, a robust model produces correct outputs even when presented with data that somehow deviates from the one used for training, including adversarial examples. These are samples to which small perturbations are added in order to purposely fool the model. There are, however, limited studies on the robustness of models designed by evolution. In this chapter, we address this gap in the literature by performing adversarial attacks and evaluating the models created by two prominent NeuroEvolution methods (DENSER and NSGA-Net). The results confirm that, despite achieving competitive results in standard settings where only predictive accuracy is analyzed, the evolved models are vulnerable to adversarial examples. This highlights the need to also address model validation from an adversarial robustness perspective.

15.1 Introduction

Successful applications of Machine Learning (ML) models are nowadays found across a variety of fields, from computer vision [16, 23] to natural language processing [42]. A primary goal while building these models is to have them perform well

I. Valentim · N. Lourenço (✉) · N. Antunes
University of Coimbra, CISUC, DEI, Coimbra, Portugal
e-mail: naml@dei.uc.pt

I. Valentim
e-mail: valentim@dei.uc.pt

N. Antunes
e-mail: nmsa@dei.uc.pt

on unseen data, which is typically referred to as their generalization performance. Since Evolutionary Machine Learning (EML) has ML at its core, this objective also applies to the models evolved by such approaches.

Validating the models before their deployment allows us to estimate their generalization performance and helps us keep any expectations regarding their performance in check.¹ Appropriate validation methods guarantee that that estimate is as unbiased as possible (e.g., by avoiding leaking test data during training) and help identify issues like overfitting. In addition, we may uncover scenarios under which a model still needs improvement (e.g., samples that are repeatedly being incorrectly classified). On the contrary, adopting incorrect validation practices, or not validating a model at all, can result in overoptimistic expectations that can ultimately lead to significant losses, which may even be fatal in the case of safety-critical systems like autonomous vehicles.

Although most EML approaches are concerned with the predictive performance of the models they evolve [54, 61], there are other aspects that should not be neglected when considering real-world applications. Namely, it is likely that a model has to process data that deviates from the one seen during training. Even under those circumstances, a model should be able to provide correct outputs, i.e., it should be robust. Model robustness has many possible interpretations, but we focus on adversarial robustness, i.e., robustness against inputs tampered with small perturbations that cause an attacked model to produce incorrect outputs [9, 15]. These perturbed inputs are commonly known as adversarial examples [21, 52].

In this chapter, we present general recommendations and methods to conduct model validation in the scope of EML. More importantly, we argue that it is crucial to look beyond the standard setting where predictive performance is the core concern and also evaluate the robustness of the models, specifically their robustness against adversarial examples. Thus, we conducted a study where we assessed the adversarial robustness of Artificial Neural Networks (ANNs) designed by NeuroEvolution (NE) approaches. More concretely, we evaluated Convolutional Neural Networks (CNNs) designed by DENSER [3] and NSGA-Net [35] for the CIFAR-10 image classification task [31].

Our results suggest that models designed by evolution are susceptible to adversarial attacks, much like ANNs designed by hand, whose vulnerability to adversarial examples has been widely investigated. Nevertheless, we also found that one of the models under evaluation shows some resistance to adversarial examples in one of the threat models considered in the experiments. On the one hand, our findings corroborate the need to look at model performance in a broader sense, as even highly accurate models can perform poorly under scenarios that deviate from in-lab environments with ordinary data. On the other hand, these results are also promising when it comes to leveraging NE to search for ANNs that are inherently more robust.

¹ In the ML literature, it is common to find references to a *validation set* of data. Typically, that validation set is used to tune model hyperparameters and perform *model selection*. Model selection is, to a certain degree, related to model validation, but, ultimately, the goal of each of these tasks differs. Model validation is concerned with fully specified models.

The remainder of this chapter is organized as follows. In Sect. 15.2, we give general recommendations for validating models designed by evolution and present related work on benchmarking Evolutionary Neural Architecture Search (ENAS) approaches. In Sect. 15.3, we introduce key concepts related to model robustness, with a special emphasis on adversarial robustness. Section 15.4 is dedicated to our study of the adversarial robustness of NE approaches, including the methodology, experimental setup, and findings. Open challenges are identified and summarized in Sect. 15.5. We discuss the main findings and future directions in Sect. 15.6, which also concludes the chapter.

15.2 Predictive Performance Assessment

A vast majority of EML approaches focus on optimizing predictive performance [54, 61]. That is the case of NE approaches for classification tasks, which often use predictive accuracy as the target metric for the fitness of each individual. The experimental design of such approaches must be done with caution so as to be able to get reliable estimates of the generalization performance of the evolved models. Given that EML approaches are, in general, computationally expensive, some standard validation methods used in ML (such as nested cross-validation [44]) may be unviable. Be that as it may, some general best practices should still be followed.

In this section, we present a general framework for the validation of models designed by evolution. The suggested method is particularly suitable for approaches that evolve models for supervised tasks, specifically ANNs. Having said that, the recommendations are general enough to be easily adapted to different scenarios and application domains. We also present a benchmarking platform aimed at ENAS proposals, an important step toward allowing fairer comparisons between models found by different approaches.

15.2.1 General Framework

To better estimate the generalization performance of an evolved model, different sets of data should be used at different stages of the EML pipeline, as illustrated in Fig. 15.1. As in any standard ML application, we suggest a first partitioning of a dataset into a training set and a test set. Common benchmark datasets (e.g., CIFAR-10) usually have pre-defined sets of training and test data. Only the training set should be used within the evolutionary process. It is imperative that the original test set is kept aside and is only used at the end of the evolutionary process, so as to obtain an unbiased estimate of the performance of the best individuals on unseen data. Moreover, by keeping the original partitioning intact, comparisons with other approaches are more likely to be feasible, since the test performance is measured in the same subset of data which, in turn, has not been used during training or

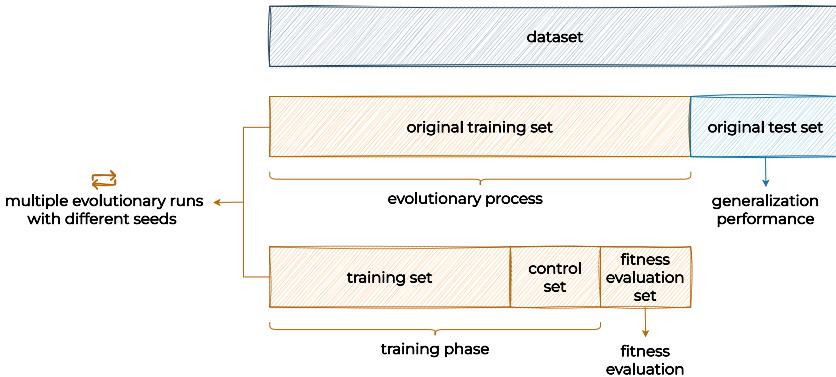


Fig. 15.1 Model validation method for evolutionary machine learning

model selection. Making the partitioning of the dataset publicly available can further facilitate proper validation and comparisons.

In addition to this partition, we advise dividing the original training set into three data subsets, as follows:

1. Evolutionary training set

It is used during the training phase, e.g., to learn the parameters of an ANN in the context of an ENAS approach.

2. Evolutionary control set

It is also used during training, but to keep track of the progress made. A concrete example is the use of this subset to control an early stopping mechanism.

3. Fitness evaluation set

It is used to evaluate the fitness of the trained individuals, thus guiding the evolutionary process.

It is also important to mention that any data augmentation technique, where additional data is introduced, should only be applied to the training data. The same applies to computing summary statistics needed for data pre-processing methods like feature normalization. As such, the data subsets should first be created, and only then should the augmentation methods be applied. Failing to do so can cause data leakage, which jeopardizes the validity of the results. Nonetheless, it is possible to perform test-time augmentation by using and combining the model predictions for multiple modified versions of each sample in the original test set to get a final outcome. Again, the key point to follow is to only perform augmentation after splitting the data.

Due to the stochastic nature of evolutionary approaches, the reported results should also take more than one evolutionary run into account. Different random seeds should be used for the partition of the original training set across these runs. Besides computing average metrics of the best individuals found across evolutionary runs, tracking how fitness evolves through generations can also give important insights into the effectiveness and efficiency of the evolutionary process. After the best individual

across runs has been selected, a final training can take place with the original training set being merged back together. In an ENAS approach, the last training could use gradient methods to optimize the parameters (i.e., weights) of the best architecture (as measured on the fitness evaluation set).

15.2.2 *Benchmarking Evolutionary Neural Architecture Search*

Besides looking at an evolved model in isolation to assess how well it performs at the task at hand, it is also fundamental to compare different approaches. By doing so, one can get a better understanding of how they position themselves in the literature, as well as track advances in the field.

Nevertheless, making these comparisons in a fair and reliable manner is not trivial. Firstly, most implementations are not publicly available, making it difficult to replicate the results of the original papers [61]. Moreover, the approaches are often complex and incorporate specific details and technicalities, which further adds to the difficulty of implementing them faithfully [34]. Additionally, the original works might not clearly specify the parameters used to obtain the reported results. There is also the issue of comparing the efficiency of the evolutionary processes. Given the differences across approaches, for instance, in terms of population size or the number of generations [61], it is not straightforward to make this comparison.

Variations in how the dataset is partitioned (as described in Sect. 15.2.1) and, consequently, the size of each data subset, also contribute to it being difficult to make unbiased and fair comparisons between methods. Some EML problems can pose even further challenges. For instance, in NE there are differences between the data preparation and pre-processing techniques adopted by each approach, not to mention the distinct details pertaining to the training of the ANNs themselves (e.g., number of epochs, batch size, and optimizer). Besides predictive performance, another factor commonly considered in this scenario is the search cost (expressed by the number of GPU days). Given the differences in the hardware used to run the experiments, this is yet another aspect to take into consideration [61].

BenchENAS [61] addresses these issues by proposing a framework to benchmark ENAS approaches. In particular, the settings regarding the data and the training strategy can be fixed across approaches. The number of function evaluations, which is often used to assess the efficiency of the evolutionary search, is also the same for the different methods. Moreover, BenchENAS employs strategies to reduce the time needed for fitness evaluations, namely by taking advantage of the use of multiple GPUs, when available, to train ANNs in parallel.

In sum, BenchENAS promotes sound comparisons of different ENAS methods by providing a platform under which models are trained and evaluated under the same experimental conditions. Nevertheless, BenchENAS makes these comparisons based on a fixed set of objectives and metrics (e.g., accuracy, number of model

parameters, and GPU days for each evolutionary run). Therefore, it is not suitable for more complex evaluations, e.g., robustness assessments. Furthermore, the existing platform does not seem to offer the option to implement a customized validation strategy in terms of dataset partitioning.

15.3 Beyond Predictive Performance: Robustness

The success of ML models led to their adoption in scenarios where concerns other than predictive performance must be addressed. One such concern pertains to how models behave at test time when processing atypical or peculiar data, for instance, as a result of distributional shifts or perturbations. In this context, the robustness of a model refers to its ability to provide correct outputs even if presented with such data.

Given the variety of peculiarities the test data can show, robustness has many facets. In this chapter, we focus on adversarial robustness, more precisely robustness against small L_p -norm perturbations. Moreover, we mainly discuss robustness in the computer vision domain.

15.3.1 Adversarial Robustness

Adversarial examples [21, 52] are carefully crafted samples that make a model produce incorrect outputs [15, 28]. Throughout this chapter, we adopt the definition of adversarial examples described in Szegedy et al. [52], where the authors found that it was possible to create an input x^{adv} similar to a valid data point x to which a model gives a highly different prediction [20]. Figure 15.2 illustrates this phenomenon.

Throughout the years, several methods have been proposed in the literature to craft such examples, under different threat models. From a security perspective,

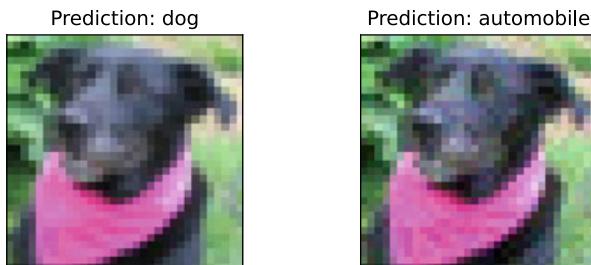


Fig. 15.2 An adversarial example. On the left is a clean example from CIFAR-10, correctly classified by a CNN; on the right is an adversarial example where perturbations were added to the benign image making the same model misclassify it

these threat models can be defined based on the goals, knowledge, and capabilities of the adversary [8].

15.3.1.1 Threat Models

A distinction can be made based on the adversary's knowledge about the model: architecture and parameters, training algorithm and training data, randomness at test time, and allowed level of query access [4]. *White-box attacks*, like gradient-based attacks [36], need full access to the model. Others, like decision-based attacks [6], are *black-box attacks* that solely need access to the final output, such as the predicted label. Yet another type of black-box attacks, transfer-based attacks [43], use a substitute model to craft adversarial examples which are then used to attack the real target model. These transfer-based attacks exploit the fact that adversarial examples generated for one model have a high probability of also being misclassified by another model (trained with a different training set or even with a different architecture) [21, 33, 36, 52]. This property is known as transferability.

We can further distinguish between untargeted and targeted attacks. In the case of image classification, the goal of an *untargeted attack* is simply to make the model predict a class different from the true label of a given instance, while the goal of a *targeted attack* is for the misclassification to be into a specific class [9]. Formally, x is an input with correct label y and C is the classifier under attack. In the untargeted setting, an adversarial example x^{adv} is such that $C(x^{adv}) \neq y$. On the other hand, given a target $t \neq y$, a targeted attack would aim at crafting x^{adv} such that $C(x^{adv}) = t$.

An adversarial example is usually created by adding some perturbation δ to a benign input [28]. Constraints are usually imposed on the capabilities of the adversary in terms of the maximum perturbation that can be added, so that x^{adv} remains close to the original input and its true label remains unchanged [8].

15.3.1.2 Attack Design

In the image domain, a common approach is to use an L_p -norm to bound the perturbations such that $\|x - x^{adv}\|_p \leq \epsilon$, where ϵ is the perturbation budget and usually $p \in \{0, 1, 2, \infty\}$ [8, 14]. The L_0 -norm limits the number of pixels that can be modified, but does not restrict the magnitude of the changes. The L_1 -norm imposes limits on the sum of the magnitudes of the changes. The L_2 -norm measures the Euclidean distance between x^{adv} and x , and allows any number of pixels to be perturbed. The L_∞ -norm limits the magnitude of the changes, but allows any number of pixels to be altered.

While some white-box attacks try to find minimal adversarial perturbations, others try to maximize some loss function with respect to the true label (typically the cross-entropy loss) [4, 14]. We present some of the latter methods assuming an untargeted



Fig. 15.3 Adversarial examples generated with FGSM and FGM. On the left, the original image without any perturbation

setting, which corresponds to the threat model used in the experiments described in Sect. 15.4. One can argue that untargeted attacks are less powerful than targeted ones [9], but on the other hand, it is harder to defend against attacks with no specific target [4].

The Fast Gradient Sign Method (FGSM) [21], a one-step gradient-based attack optimized for the L_∞ -norm, generates an adversarial example as

$$x^{adv} = x + \epsilon \cdot \text{sign}(\nabla_x L(x, y))$$

where $\nabla_x L(x, y)$ is the gradient of the cross-entropy loss with respect to the input image. When this method is optimized for the L_2 -norm, we get the Fast Gradient Method (FGM) which generates an adversarial example as

$$x^{adv} = x + \epsilon \cdot \frac{\nabla_x L(x, y)}{\|\nabla_x L(x, y)\|_2}$$

In both cases, the perturbed image is clipped to the valid data range. Figure 15.3 shows examples of adversarial examples generated with the FGSM and the FGM attacks on the same clean image from CIFAR-10.

The methods that follow consider L_∞ adversaries, but these definitions can be easily adapted to the case where L_2 bounds are imposed, similar to what is done with FGM.

A straightforward extension of the FGSM attack is to take multiple small steps (with step size α) and clip the result by ϵ at each iteration. The clipping operation also takes into account the valid range of data values. This results in the Basic Iterative Method (BIM) [32] which can be defined as

$$\begin{aligned} x_0^{adv} &= x, \\ x_{i+1}^{adv} &= \text{clip}_\epsilon \left\{ x_i^{adv} + \alpha \cdot \text{sign}(\nabla_x L(x_i^{adv}, y)) \right\} \end{aligned}$$

The Projected Gradient Descent (PGD) method [36] is another iterative attack which only differs from BIM on how x_0^{adv} is set. Instead of starting from the original input, a random perturbation bounded by ϵ is generated and added to x .

In an attempt to stabilize the update directions and escape from local optima, the Momentum Iterative Fast Gradient Sign Method (MI-FGSM) [14] incorporates momentum [51] into the BIM method. The Auto-PGD (APGD) method [11] is a variation of the PGD attack which adjusts the step size in an automated way. The authors of APGD also proposed an alternative to the cross-entropy loss called Difference of Logits Ratio (DLR) loss. In addition to being invariant to shifts of the logits, the DLR loss is rescaling invariant [11].

One of the most notorious attacks aimed at finding minimal perturbations is the Carlini and Wagner attack with L_2 distortions [9]. The targeted version of the attack searches for w that solves the following objective:

$$\min \left\| \frac{1}{2}(\tanh(w) + 1) - x \right\|_2 + c \cdot f\left(\frac{1}{2}(\tanh(w) + 1)\right)$$

where a binary search over c is used to find the minimal perturbation that results in a successful attack and where f is defined as

$$f(x') = \max \left(\max_{i \neq t} (Z(x')_i) - Z(x')_t, 0 \right)$$

with t being the target class and $Z(x) = z$ being the output of the layer before the softmax layer, i.e., z are the logits.

Regarding attacks that require less knowledge about the target model than white-box ones, we highlight the Boundary Attack [6] and the Square Attack [2].

Another class of attacks, known as Universal Adversarial Perturbations (UAPs), tries to find a single perturbation vector that can fool an ANN on most samples of the input distribution [38]. Crafting adversarial examples using UAPs is more efficient than attacks that operate on an instance basis, which require the computation of a perturbation from scratch for each sample. The fact that UAPs are also transferable across different model architectures poses additional threats to the robustness of ANNs [38]. However, the fooling rates of methods based on UAPs tend to be lower in comparison to per-instance attacks [38].

There is also a line of work that uses evolution-based methods to generate adversarial examples. Two noteworthy works are the One-Pixel Attack [49] and the GenAttack [1]. The One-Pixel Attack proposed by Su et al. [49] uses Differential Evolution (DE) [48] to generate attacks where a single pixel of an image can be modified. GenAttack [1] uses Genetic Algorithms (GAs) to generate L_∞ adversarial samples in a targeted setting. Both of these approaches are gradient-free methods, but they do require access to the class probabilities (output of the softmax layer).

While the vast majority of the literature studies norm-constrained perturbations, where small norms act as proxies of human perception, other notions and threat models have been discussed and deserve to receive attention from the community [19]. As

examples of such alternative scenarios, we have the attack on traffic signs proposed by Eykholt et al. [17], as well as the adversarial patches from Brown et al. [7].

15.3.1.3 Defenses and Assessment

Several defenses against adversarial examples have been proposed in the literature. Such defenses tend to be designed to be robust to one specific threat model [8]. Adversarial training [36] and its variants have shown the most promising results when it comes to increasing the robustness of models. The main idea behind these methods is to incorporate adversarial examples in the training procedure of a model. Another family of defenses comprises methods that, instead of making the ANNs themselves more robust, aim at detecting adversarial examples during inference time [37]. Many more proposals and approaches can be found in the literature, and so, this is by no means an exhaustive review of all existing work on adversarial defenses. Although more advances have been made in recent years, the 2019 survey by Yuan et al. [64] gives a comprehensive overview of the topic.

Adversarial robustness evaluations typically consist of heuristic approaches that estimate the robustness of a model by performing adversarial attacks [10]. One resorts to this approximation since computing it exactly is usually intractable [8, 10]. Nevertheless, there is a line of work regarding provable approaches that compute exact bounds of the adversarial robustness of a model [30, 55]. These methods fall outside the scope of this chapter.

It is important to mention that performing a correct robustness evaluation is a difficult and error-prone task, especially when dealing with models that incorporate some kind of defense. Many defense proposals conducted incorrect or incomplete evaluations and, as a consequence, were circumvented shortly after publication [4, 57]. In particular, some defenses are able to cause gradient-based attacks to fail, but can be broken with the methods proposed by Athalye et al. [4] or with gradient-free attacks, such as the GenAttack [1]. In this scenario, an evaluation that only uses gradient-based attacks would be misleading.

RobustBench [10] is a relatively recent initiative to keep track of the progress made in adversarial robustness. It aims at benchmarking adversarial robustness under different threat models, namely L_p -robustness with $p \in \{2, \infty\}$. The authors adopt AutoAttack [11], a heuristic evaluation method which relies on an ensemble of white-box and black-box attacks. Although this method can be used to estimate the adversarial robustness of ANNs trained in normal regimes without adversarial considerations, the main focus of the work are models that incorporate some kind of defense against adversarial attacks. In addition to a leaderboard aggregating the evaluations of several robustness-enhancing proposals, a Model Zoo containing pre-trained models from top entries of the leaderboard is also available. Nevertheless, some restrictions are introduced on the considered models, mainly to avoid overestimating robustness due to adaptive evaluations not being performed [10].

In Benz et al. [5], a comparison is made between CNNs and more recent architectures, such as the Vision Transformer [16] and the MLP-Mixer [56], which have

achieved promising results in computer vision tasks. This work relates to ours in that architectural differences are at the core of the analysis.

The work by Huang et al. [27] focuses on evaluating the impact of network width and depth on the adversarial robustness of a Wide Residual Network. This ANN is adversarially trained, and its width and depth are modified at different stages of the model. Their results suggest that reduced width and depth at the last stage of the model can be beneficial.

The experiments conducted by Devaguptapu et al. [12] also look at adversarial robustness from an architectural perspective and include both manually designed architectures and architectures found by NAS approaches (such as an NSGA-Net model). However, only L_∞ -robustness is considered in this work. Surprisingly, the accuracy of the handcrafted models for CIFAR-10 does not drop to zero under attack, which raises questions in terms of the adequacy of the evaluation methodology.

There is also a growing body of work that uses NAS approaches, including NE, to find robust models [18, 22, 46, 59]. As pointed out by Devaguptapu et al. [12], adversarial training is often incorporated in these studies, making it difficult to assess the role of architectural aspects on the robustness exhibited by the models. In the case of Vargas and Kotyan [59] and Geraeinejad et al. [18], adversarial robustness is explicitly included in the objective function. Thus, it is difficult to understand if these NE approaches would be able to find inherently more robust models than the ones crafted by humans without specifically driving the search toward that goal.

15.3.2 Other Forms of Robustness

In the examples seen so far, small perturbations are added to benign data samples with the goal of producing labeling errors. An underlying idea is that the perturbed examples remain semantically close to their clean counterparts, meaning that, for instance, we want a perturbed image of a dog to still look like a dog to a human. However, there are other possibilities when it comes to generating images that are capable of fooling a model. Namely, it is possible to generate images unrecognizable to humans or that seemingly look like noise, but that a model classifies as some known class with high confidence. A noteworthy example is the work by Nguyen et al. [40], where such images were generated using evolutionary algorithms and one of two possible representations: a direct encoding of the pixel values as integers, and an indirect encoding based on Compositional Pattern Producing Networks (CPPNs) [47].

Adversarially perturbed inputs are not the only cause for concern regarding model robustness. Other relevant data distortions include the common image corruptions of the benchmark designed by Hendrycks et al. [25]. It comprises 15 types of corruption from four categories, namely noise, blur, weather, and digital distortions. Each corruption type has five severity levels. Gaussian noise, motion blur, fog, and brightness are examples of distortions from each category. RobustBench [10] also covers this type of image perturbation.

One can argue that these common corruptions and perturbations are unrealistic. Bearing in mind this limitation, the 3D Common Corruptions (3DCC) set [29] has recently been proposed. The 3DCC set is more targeted toward datasets with full scene geometry information, but can also be applied to datasets lacking this 3D information, like ImageNet [45].

An underlying problem of the aforementioned data distortions is that they are synthetic, meaning that the data samples are obtained by applying well-defined transformations that are generated computationally [53]. In Taori et al. [53], the authors argue that the models should also be robust to distribution shifts that occur naturally in the real world, such as changes in lighting or scene compositions. To assess model robustness against these natural distribution shifts, they then use datasets of unmodified images grouped into three categories: consistency shifts, dataset shifts, and adversarially filtered shifts.

Another problem plaguing the robustness of ML models is the open-world nature of real scenarios, given that a closed-world view is often adopted during training [62]. Taking the case of image classification tasks, like the one associated with the CIFAR-10 dataset, models will always output a class from a closed set of possibilities. They are often highly confident about that prediction even if the input image does not fit in any of the classes the models are expected to identify [62].

In real-world applications, it is most likely that changes will occur over time, namely when it comes to the relationships between the inputs and the target outputs of a model [39]. Such changes, often called concept drift [39, 60], can cause model performance to deteriorate, in part due to the model typically being static and trained in historical data where the new relationships were not present [60]. Methods to detect concept drift and address it remain open challenges.

15.4 An Empirical Evaluation of the Adversarial Robustness of NeuroEvolution Methods

Models generated by NE approaches achieve competitive results performance-wise, but their robustness to adversarial examples has received limited attention. We can assume that these evolved models also suffer from this vulnerability, but we cannot be sure whether the evolutionary search yields ANNs that offer some kind of resistance to adversarial examples, and, if that is the case, to what extent it goes.

For that reason, we conducted an empirical evaluation of the adversarial robustness of models designed by two NE approaches [58]. The experiments and results presented herein are largely based on our previous work [58]. The main goal of this study was to understand whether these evolved models show any inherent robustness even when that objective is not included in the evolutionary search procedures. We focused on image classification tasks and CNNs, and performed such an analysis by attacking pre-trained models made publicly available by the authors of the

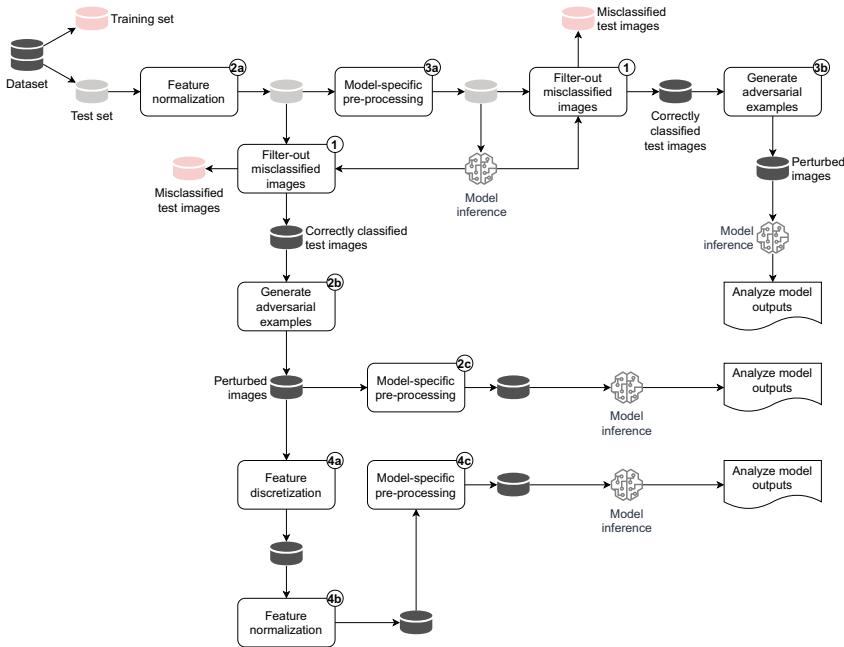


Fig. 15.4 Approach to assess the adversarial robustness of NeuroEvolution methods

corresponding NE approaches. Two threat models based on L_p perturbations were considered.

Besides adversarial robustness not being included in the fitness functions of any of the selected approaches, no defense mechanism (e.g., adversarial training) is applied to the models in any phase of the process. As such, the models were not specifically designed to be adversarially robust.

15.4.1 Methodology

We followed the approach depicted in Fig. 15.4 to assess the adversarial robustness of the selected models. Only the test set of a dataset is used to generate adversarial examples. Furthermore, the attacks are only performed on test samples that, without being perturbed, are correctly classified by the model being evaluated, as depicted by the steps marked with (1).

Different models expect the data to undergo distinct pre-processing procedures. We generate adversarial examples in a more general scenario where a single pre-processing step is applied to the images to normalize the pixel values to the interval $[0, 1]$, as shown by steps (2a) and (2b). In this case, additional pre-processing procedures are only applied after the adversarial perturbations have been added to the

images, as illustrated by step **(2c)**. Additionally, we consider a scenario in which the adversarial examples are crafted after all the pre-processing steps are applied to the images, as depicted by steps **(3a)** and **(3b)**. Under these conditions, the perturbation budget refers to the distance in the space in which the first layer of a network expects the data to be.

Pixel values are typically represented by integers between 0 and 255, but we model them as real numbers. The discretization of the perturbed images can decrease the success rate of the attacks mainly if the perturbations are too small. Thus, our methodology also accounts for the impact of converting the pixel values of the perturbed images back to valid integers, as represented by steps **(4a)**, **(4b)**, and **(4c)**.

The analysis of the models under attack includes computing their accuracy on the perturbed images (which is closely related to the success rate of the attacks), as well as studying the corresponding confusion matrices.

15.4.2 Experimental Setup

In the remainder of this section, we describe the dataset used in the experiments, as well as the models whose adversarial robustness was evaluated. Additionally, we specify the threat models considered and detail the attacks used under each scenario. We also clarify how the metrics used to compare the different models are computed.

15.4.2.1 Dataset and Target Models

All the experiments were conducted on the CIFAR-10 dataset [31], which consists of 32×32 RGB images divided into 10 classes. The test set, which was used to generate adversarial examples, has 10000 images equally distributed across classes (i.e., the dataset is balanced). The original pixel values are in $[0, 255]$, but all models operate with pixels modeled as real numbers. Thus, a pre-processing step is applied across models to normalize the pixel values to the interval $[0, 1]$.

We evaluated models designed by two NE approaches, DENSER [3] and NSGA-Net [35], and compared them against one handcrafted architecture, WRN-28-10 [65]. This choice was mainly based on two criteria: firstly, models had to be directly trained on CIFAR-10, and secondly, pre-trained models (i.e., network weights) had to be publicly available, so as to introduce as little bias as possible from our end.

All the pre-trained models were used directly, without any form of re-training or tuning. Nevertheless, we describe some relevant differences in the training procedures of each model in what follows. These design choices were made by the original authors of the NE approaches or by the authors of the Model Zoo of RobustBench [10], the source of the pre-trained WRN-28-10 network used in our experiments.

The WRN-28-10 architecture [65], a manually designed Wide Residual Network, was used as the baseline model in our experiments for multiple reasons: its performance on CIFAR-10 is similar to that of the remaining models under evaluation, the work which proposes NSGA-Net [35] also uses it as a baseline, and some of the defenses from the RobustBench leaderboard [10] are based on this architecture. As previously mentioned, we used the pre-trained model from the Model Zoo of RobustBench,² which was trained with the 50000 training images of CIFAR-10, without any data augmentation. The only pre-processing applied to the data is the conversion of the pixel values from [0, 255] to [0, 1].

In what concerns DENSER [3], we selected the network that achieved the best accuracy on the CIFAR-10 test set over 10 evolutionary runs. This architecture consists of 10 convolutional layers and 2 pooling layers, followed by 2 fully connected layers [3]. The models resulting from 5 independent training runs of this network are publicly available,³ but, again, we solely attacked the one with the highest test accuracy. In these training runs, the original work used the complete training set of CIFAR-10 and applied a data augmentation method which includes padding, horizontal flipping, and random crops (similar to what is done in Suganuma et al. [50]). In addition to converting the pixel values to real numbers, the data is expected to be centered around zero before being fed to the first layer of the network. Following Assunção et al. [3], this is accomplished through the removal of the mean pixel value per location and color channel (calculated on the entire training set).

As far as the NSGA-Net approach [35] is concerned, we conducted experiments with a pre-trained model (NSGA-M) from a macro search space of networks with 3 phases and fixed changes in spatial resolution between them. Moreover, each phase had a maximum of 6 nodes, each composed of the same sequence of operations. In addition to that model, we also attacked three variants of an architecture obtained by using the cells (normal and reduction) found by NSGA-Net on the NASNet micro search space [66] (NSGA-mA, NSGA-mB, and NSGA-mC with an increasingly higher number of model parameters, as shown in Table 15.1). In the original work, cutout [13] was used to train these models, together with a data augmentation strategy similar to the one adopted by DENSER, which includes padding, random crops, and horizontal flipping. For the three models from the micro search space, the scheduled path dropout technique [66] was also adopted, together with an auxiliary head classifier, whose loss is aggregated with the loss from the main network [35]. After converting the pixel values to real numbers, the data is expected to be normalized using pre-calculated means and standard deviations for each color channel. Further details about the architectures and training procedure can be found in the original paper [35] and the source code repository.⁴

Table 15.1 presents an overview of the size of the models used in our experiments, as given by the number of trainable parameters. We reiterate that all models used a

² <https://github.com/RobustBench/robustbench>.

³ <https://github.com/fillassuncao/denser-models>.

⁴ <https://github.com/ianwhele/nsga-net>.

Table 15.1 Overview of the models in terms of number of parameters and accuracy on the clean examples of the CIFAR-10 test set

Model	No. of parameters	Clean accuracy (%)
WRN-28-10	36.48 M	94.78
DENSER	10.81 M	93.70
NSGA-M	3.37 M	96.27
NSGA-mA	1.97 M	97.57
NSGA-mB	2.20 M	97.78
NSGA-mC	4.05 M	97.98

standard training procedure and no defensive method against adversarial examples was applied.

15.4.2.2 Threat Models and Adversarial Attacks

We considered the scenario in which an attacker has full access to the target model and performed white-box attacks, a choice mainly motivated by the fact that all models are publicly available and details about their training are easily accessible.

Furthermore, we considered untargeted attacks, where the adversarial perturbations were bounded by $\epsilon = 8/255$ under the L_∞ -norm or, in the case of the L_2 -norm, by $\epsilon = 0.5$. These perturbation budgets ϵ were chosen based on threat models used in previous works, namely in the RobustBench benchmark [10].

Instead of finding minimal adversarial perturbations, we focused on attacks that craft adversarial examples by maximizing the loss with respect to the true label while solving the constrained optimization problem. Therefore, we attacked the chosen models using different configurations (number of iterations and number of random initializations) of the following attacks:

- the FGSM and the FGM attacks, for the L_∞ and the L_2 threat model, respectively;
- the BIM attack, for both threat models;
- the PGD attack, for the L_2 threat model.

For the iterative attacks (i.e., BIM and PGD), we set the step size to $\alpha = \epsilon/4$. We used the Python implementations of the attacks by the Adversarial Robustness Toolbox (ART) library [41].

15.4.2.3 Metrics and Baseline Performance

Table 15.1 shows the accuracy of the models on the clean examples of the CIFAR-10 test set. For a fair comparison, adversarial examples were only generated on samples that, before being perturbed, are correctly classified by the model under evaluation. Consequently, the higher the clean accuracy of a model (e.g., NSGA-mC), the larger

the number of generated attacks, i.e., the number of images that are adversarially perturbed. In an untargeted setting, an attack succeeds if the model produces a misclassification, regardless of the predicted class. For this reason, a sample incorrectly classified can be considered a successful attack where no adversarial perturbation had to be added to the original input. Notwithstanding, the models' accuracy on adversarially generated samples is reported taking the complete test set (10000 images) into account.

15.4.3 Experimental Results for the Standard Scenario

We first look into the results for the scenario where the attacks operated in $[0, 1]$, and any additional model-specific pre-processing was only applied after the adversarial perturbations were added to the images. Table 15.2 shows the accuracy of the models on the images with L_∞ perturbations. We present the results for the FGSM attack, for FGSM with 10 random initializations (FGSM-10), and for the BIM attack with 10 and 50 iterations (BIM-10 and BIM-50, respectively).

A brief perusal of the results reveals that the DENSER model is the most susceptible to L_∞ attacks. Even in the case of single-step attacks like FGSM, the accuracy falls below 10% when random restarts are incorporated. On the other hand, the models designed by NSGA-Net on the NASNet search space are the most resistant to these attacks, even achieving higher accuracy on the adversarially perturbed images than the baseline model. However, the four NSGA-Net models are also the ones where the most drastic drops in accuracy are observed when random restarts are incorporated in FGSM.

Despite these observations, the accuracy of all models eventually drops to zero under this threat model if we resort to attacks with enough iterations (BIM-50). If L_∞ robustness is not explicitly included in the evolutionary process, these NE approaches do not seem to find highly accurate models that are also robust against this type of perturbations.

Table 15.2 also shows the adversarial accuracy of the models when attacked with L_2 -bounded perturbations. We present the results for the FGM attack, for the BIM attack with 10, 50, and 100 steps (BIM-10, BIM-50, and BIM-100, respectively), as well as for the PGD attack with 10 random restarts and 50 iterations (PGD-50-10).

Under the single-step FGM attack, the DENSER model remains the most susceptible, while the NSGA-Net models from the micro search space remain the most robust. Nevertheless, the strongest L_2 attacks in our analysis (BIM-100 and PGD-50-10) still bring the accuracy of most models to below 1% and, in the case of NSGA-M and NSGA-mB, to zero. Surprisingly, especially given the observations with L_∞ distortions, this does not hold true for DENSER, whose accuracy just drops to around 20% under this threat model.

Overall, we consider this to be a promising result and a possible indicator that more robust models may be found in the search space used by DENSER. Moreover, the discrepancies in the relative robustness of the models between the two distance

Table 15.2 Accuracy on the CIFAR-10 test set when the attacks operate in $[0, 1]$. The highest reported accuracy under each attack is in bold

		WRN-28-10 (%)	DENSER (%)	NSGA-M (%)	NSGA-mA (%)	NSGA-mB (%)	NSGA-mC (%)
L_∞ $\epsilon = 8/255$	FGSM	28.85	16.37	35.08	52.09	51.86	55.06
	FGSM-10	11.03	6.19	9.28	25.02	22.49	26.92
	BIM-10	0.02	0.00	0.00	0.16	0.00	0.02
	BIM-50	0.00	0.00	0.00	0.00	0.00	0.00
L_2 $\epsilon = 0.5$	FGM	47.61	44.76	48.51	61.34	60.61	64.06
	BIM-10	2.01	30.76	0.23	3.04	0.73	2.57
	BIM-50	0.16	24.13	0.00	0.26	0.01	0.35
	BIM-100	0.09	21.76	0.00	0.12	0.00	0.23
	PGD-50-10	0.08	18.10	0.00	0.11	0.00	0.21

metrics demand further analysis. We are especially interested in understanding what characteristics of the DENSER model make it more L_2 -robust and why those do not seem to help the model against L_∞ attacks. This is the focus of the work we are currently developing.

A comparison between the three NSGA-Net models from the micro search space also reveals that NSGA-mB is marginally less robust than the other two, even though it is more complex (i.e., it has a higher number of parameters) than NSGA-mA. This is observed under both distance metrics, but the susceptibility of NSGA-mB is particularly higher than that of NSGA-mA and NSGA-mC under the BIM-10 attack with L_2 perturbations. A distinguishing factor between the three networks is that, unlike NSGA-mA and NSGA-mC, NSGA-mB does not use Squeeze-and-Excitation blocks [26]. Thus, it seems that Squeeze-and-Excitation may help improve the robustness of an ANN, a hypothesis worth investigating in future work.

15.4.4 Impact of Data Pre-processing

In Sect. 15.4.2.1, we described the different pre-processing steps applied to the images before they reach the first layer of each model. Contrary to WRN-28-10 and DENSER, the pre-processing for the NSGA-Net models changes the scale of the data (i.e., the difference between the maximum and minimum values of a feature is larger than 1). Therefore, when the attacks operate in $[0, 1]$ as in the scenario considered thus far, the NSGA-Net models perceive the adversarial perturbations as approximately 4 times larger than the perturbation budget of the threat model.

To show this effect, we considered a scenario where the perturbation budget remains the same but refers to the distance in the space in which the first layer of a network expects the data to be. This is accomplished by crafting the adversarial

Table 15.3 Accuracy on the CIFAR-10 test set when all model-specific pre-processing is applied to the original inputs before performing the attacks. The highest reported accuracy for each attack is in bold

		DENSER (%)	NSGA-M (%)	NSGA-mA (%)	NSGA-mB (%)	NSGA-mC (%)
L_∞ $\epsilon = 8/255$	FGSM	16.37	46.65	60.61	61.57	63.64
	FGSM-10	6.16	40.82	58.41	58.88	60.97
	BIM-10	0.00	2.70	12.22	8.36	12.70
	BIM-50	0.00	0.81	6.87	3.86	6.96
L_2 $\epsilon = 0.5$	FGM	44.75	67.96	78.11	77.70	80.12
	BIM-10	30.77	25.98	48.60	44.80	49.61
	BIM-50	24.13	17.57	41.63	37.43	42.04
	BIM-100	21.76	16.86	40.72	36.46	41.09
	PGD-50-10	18.10	15.99	40.10	35.38	40.05

examples after all the pre-processing steps are applied to the data, instead of performing the attacks in the $[0, 1]$ range. We are aware that it is misleading to claim that the perturbation budget is the same, but our goal is to show that failing to specify this simple detail may jeopardize the validity of the conclusions taken from this type of study. This issue becomes even more relevant when conducting a study with pre-trained models from different sources, since there is a high chance the pre-processing procedures are not consistent across them.

The results of this experiment are shown in Table 15.3 for both the L_∞ and the L_2 attacks. The baseline model is excluded from this analysis since it only requires the data to be in $[0, 1]$, without centering or standardizing it.

As expected, for the DENSER model there is no significant difference between these results and those from Table 15.2. In contrast, the robustness of the NSGA-Net models appears to be much higher now, especially in the case of L_2 -bounded perturbations.

On the one hand, this experiment shows that the choice of data pre-processing may have a negative impact on adversarial robustness. Therefore, this decision should not be overlooked while designing networks under scenarios where adversarial examples may be a concern. In the particular case of NE approaches, one might even consider including this design choice in the evolutionary process, even though there is not always a straightforward way to do so (e.g., NSGA-Net). On the other hand, this experiment highlights the importance of clearly specifying the conditions under which the attacks were generated and the target models were trained (including any model-specific pre-processing step) to avoid overestimating robustness when performing these evaluations.

Table 15.4 CIFAR-10 test set accuracy when the attacks operate in $[0, 1]$, but the generated images are post-processed. The highest reported accuracy for each attack is in bold

		WRN-28-10 (%)	DENSER (%)	NSGA-M (%)	NSGA-mA (%)	NSGA-mB (%)	NSGA-mC (%)
L_∞ $\epsilon = 8/255$	FGSM	28.85	16.38	35.08	52.09	51.86	55.06
	FGSM-10	11.15	6.28	9.45	25.19	22.70	27.13
	BIM-10	0.02	0.00	0.00	0.16	0.00	0.02
	BIM-50	0.00	0.00	0.00	0.00	0.00	0.00
L_2 $\epsilon = 0.5$	FGM	47.69	44.86	48.62	61.46	60.82	64.24
	BIM-10	2.02	30.77	0.23	3.10	0.75	2.67
	BIM-50	0.16	24.13	0.00	0.26	0.01	0.38
	BIM-100	0.09	21.76	0.00	0.12	0.00	0.24
	PGD-50-10	0.08	18.10	0.00	0.14	0.01	0.23

15.4.5 Impact of Discretization Post-processing

We also evaluated the impact of converting the pixel values of the perturbed images back to integers between 0 and 255. We just considered the case in which the attacks operated in the range $[0, 1]$ and any additional pre-processing was applied after the perturbations were added to the images. Therefore, we multiplied each pixel value by 255 and rounded to the nearest integer (nearest even integer for values exactly halfway between two integers). We then re-applied all pre-processing steps required by the model and reported the accuracy on the post-processed examples. Table 15.4 shows the results for the attacks under both distance metrics.

For the L_∞ attacks, differences are mainly detected when random restarts are incorporated (FGSM-10). The attack success rate slightly deteriorates, but the differences are of less than 0.25% and seem negligible. As far as the L_2 attacks are concerned, the largest differences occur with FGM but also seem negligible (always of less than 0.25%). The success of the attacks is mostly affected by this post-processing procedure when their target is an NSGA-Net model from the NASNet search space.

As pointed out in Carlini et al. [9], this scenario is particularly relevant when dealing with attacks that can produce considerably smaller perturbations, as may be the case with attacks that find the minimal perturbations that fool a model.

15.4.6 Analyzing Misclassifications Through Confusion Matrices

To complement our analysis, we looked into the confusion matrices of each model under different attacks and found that even when the same attack brings the accuracy

of all models to zero (e.g., the L_∞ -constrained BIM-50 attack), different patterns in their misclassifications can still emerge.

As shown in Fig. 15.5a, under an L_∞ -bounded BIM-50 attack, the misclassifications produced by WRN-28-10 for each class are spread out across the remaining classes. Moreover, this model seems to favor mainly two classes, bird and cat, with a large number of examples being misclassified as such (14.87 and 18.29 %, respectively). Under the L_2 -constrained BIM-100 attack (Fig. 15.5b), the misclassifications of images from classes that represent a means of transportation (airplane, automobile, ship, and truck) are more clustered together.

Figure 15.6a shows that, under the L_∞ -constrained BIM-50 attack, the predictions of DENSER can be clearly grouped into clusters, with most examples from one class being misclassified into a smaller subset of the other classes than with the baseline model. Images that represent an animal are mainly misclassified as another animal, while images of a means of transportation are misclassified as some other vehicle. Another common mistake made by DENSER is to classify birds as airplanes, and vice versa. The confusion matrix of DENSER under the BIM-100 attack with L_2 perturbations is shown in Fig. 15.6b. Contrary to the L_∞ attack, the BIM-100 attack is unable to decrease the accuracy to zero, and so, some perturbed images are correctly classified. Other than that, similar patterns emerge between the two types of perturbations. The automobile class is the most difficult to attack with L_2 perturbations, while it is easier to cause a misclassification of airplane instances.

According to Fig. 15.7a, and similar to the baseline model, most examples are also misclassified as bird and cat with NSGA-M. However, misclassifications of a single class are less spread out between the remaining ones, especially in the case of bird, cat, ship, and truck. The three NSGA-Net models from the NASNet search space show similar patterns in their confusion matrices. The main distinguishing factor is the spread of the misclassifications of each class: NSGA-mB misclassifies the majority of the examples from one class into fewer classes than NSGA-mC (check,

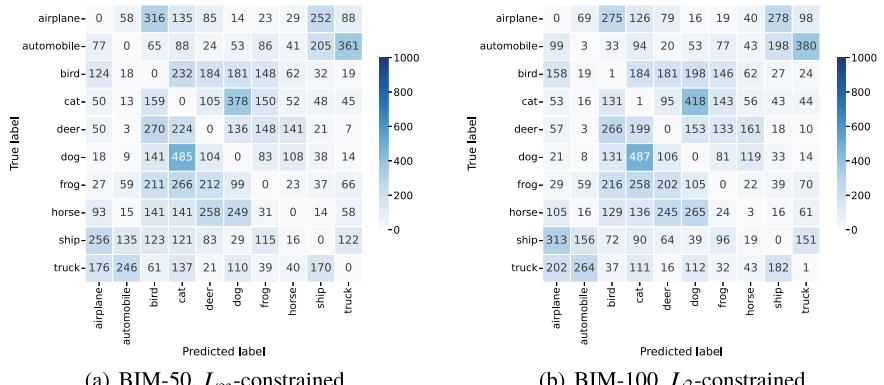
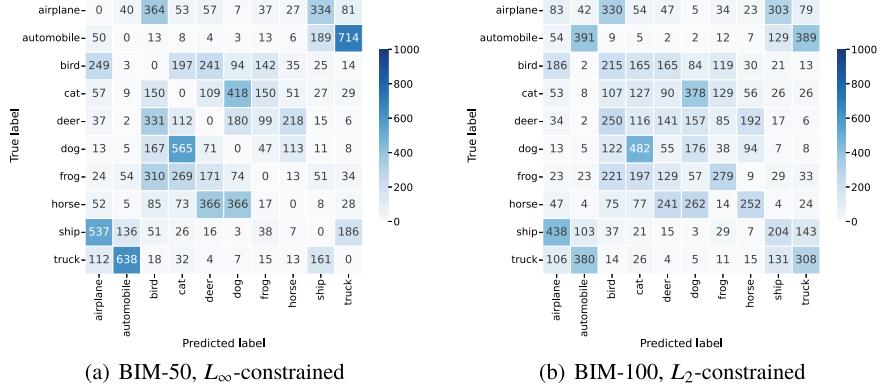
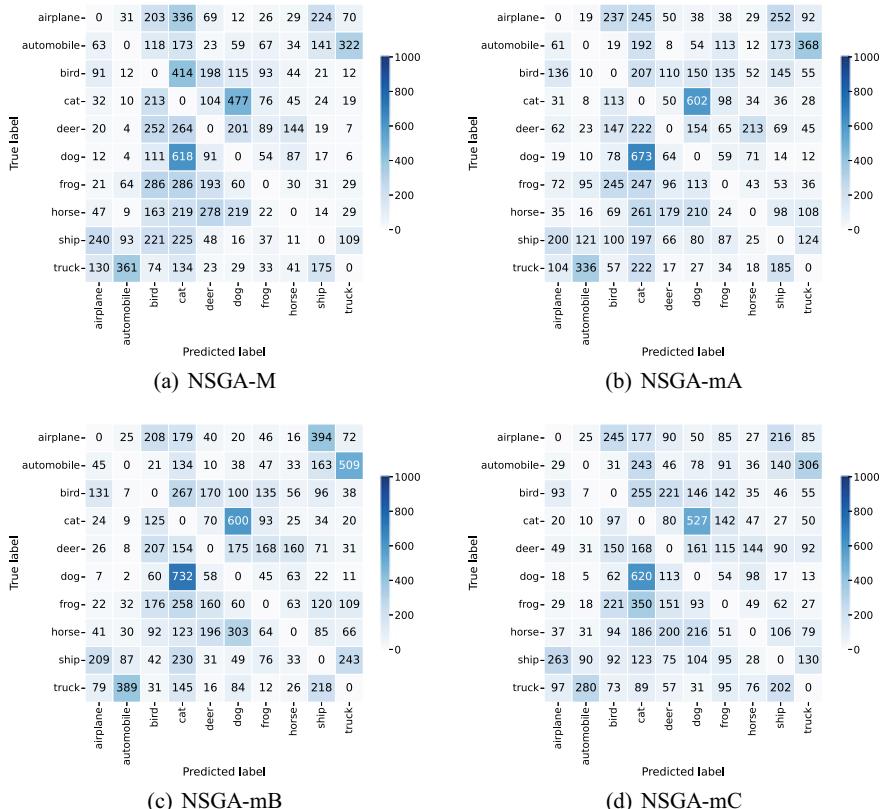


Fig. 15.5 Confusion matrices for the WRN-28-10 model under two attacks

**Fig. 15.6** Confusion matrices for the DENSER model under two attacks**Fig. 15.7** Confusion matrices for the NSGA-Net models under the BIM-50 attack with L_{∞} perturbations

for instance, the ship class), and NSGA-mA is in the middle of the spectrum. Similar to NSGA-M, most misclassifications of these three models also fall into the cat class (especially with NSGA-mA). However, the tendency of bird being the second most (mis)predicted class is not observed with these models. A prevailing mistake made by all NSGA-Net models is to classify dog instances as cats.

The confusion matrices for the BIM-100 attack with L_2 perturbations exhibit similar patterns, as shown in Fig. 15.8. In comparison with the BIM-50 attack with L_∞ constraints, less examples are misclassified by the three models from the micro search space as cat instances, especially in the case of NSGA-mB. With NSGA-M and NSGA-mB, some changes are also observed with the misclassification of examples that originally belong to the ship class.

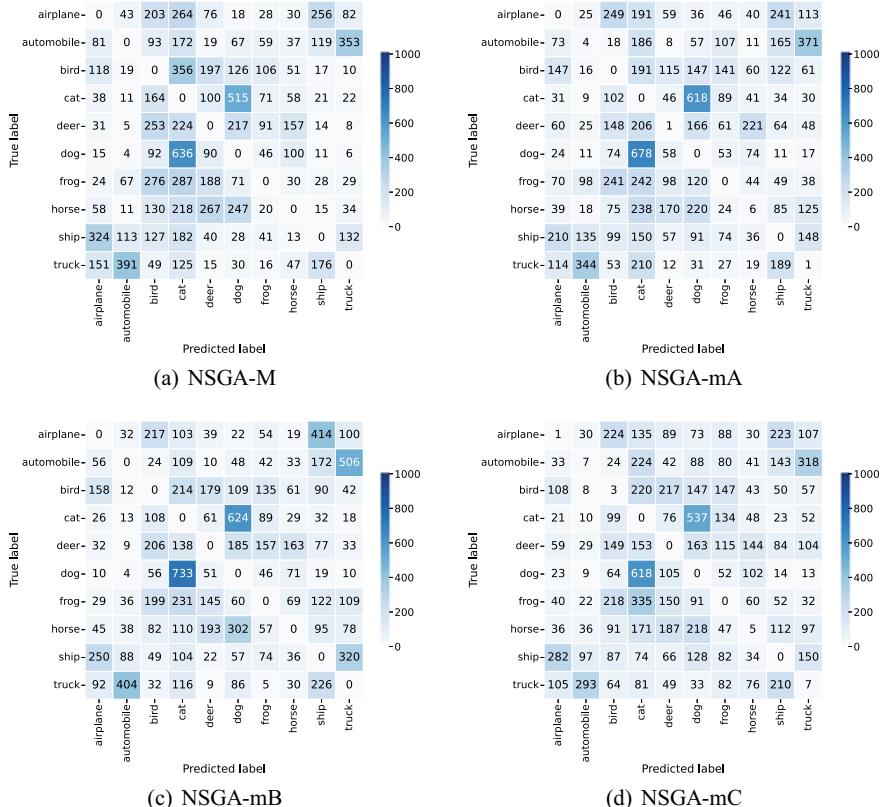


Fig. 15.8 Confusion matrices for the NSGA-Net models under the BIM-100 attack with L_2 perturbations

15.5 Open Challenges

Although there have been several proposals to generate adversarial examples using evolutionary methods, EML approaches that aim at improving the adversarial robustness of ML models, namely ANNs, are scarce. That being said, there is a growing body of work on the intersection of the two fields, which makes it necessary to keep track of the progress made. That naturally entails that we need means to compare approaches against one another, namely through well-established benchmarks. Existing tools only tackle these problems independently. For instance, RobustBench [10] benchmarks adversarial robustness but is not concerned with how the models are designed, as long as the prerequisites for being evaluated with the proposed method are met. Conversely, BenCHENAS [61] benchmarks ENAS approaches but does not support adversarial robustness assessments. Both benchmarking tasks already face several challenges when considered on their own, and so, such issues are only expected to be amplified with a benchmark that addresses the two tasks simultaneously, especially given the diversity within the approaches that may be a target for evaluation.

In the adversarial robustness research field, there have been discussions on the suitability and relevance of norm-constrained perturbations, especially in terms of L_p -norms acting as good (or bad) proxies of human perception [19]. Moreover, the underlying assumption that the perturbations should be small or imperceptible may be irrelevant under certain scenarios [19, 24].

Given the wide variety of threat models and the different facets of robustness, as described in Sect. 15.3, it is also an open issue to understand how, and if, a model being robust to a certain type of perturbations affects its robustness to some other kind of inputs. The work by Yin et al. [63] dives into the study of these trade-offs, but further analysis is needed, especially since new methods to improve robustness (either adversarial or some other form) are bound to appear in the literature.

Understanding the role played by certain architectural characteristics of the ANNs in their adversarial robustness, an aspect partially addressed in Benz et al. [5] and Huang et al. [27], still needs further investigation. We argue that gathering such insights is a crucial stepping stone to EML approaches that can create robust ANNs. On the other hand, incorporating adversarial robustness as an optimization objective of an ENAS approach is a challenge in itself. For instance, several components of the adversarial robustness fitness assessment strategy must be defined, such as the methods used to craft adversarial samples or the stages at which those samples should be generated. Going back to the need to then benchmark different proposals, all these design choices undoubtedly make creating such a benchmark a daunting task.

15.6 Discussion and Conclusions

The true value of ML models is their ability to generalize and perform well on data that has not been seen during training. Thus, when building these models, it is crucial to estimate how good they will do in production, i.e., estimate their generalization performance. In this chapter, we presented a general framework to validate models designed by evolutionary techniques. Establishing fair comparisons with similar approaches found in the literature is another fundamental task when designing new methods. Therefore, we also reviewed a recent initiative to benchmark ENAS approaches.

Following these good practices allows us to get a clearer picture of the current state-of-the-art and advances in the field. In fact, several models designed by evolutionary methods, in particular ANNs, have achieved impressive results in terms of predictive accuracy. However, new concerns arise with this success and the potential application of evolved models in real-world scenarios. Namely, the models should be robust, i.e., they should produce correct outputs even if presented with data that somewhat deviates from what was seen during training. From the many facets of robustness, we focused on robustness against adversarial examples, which are carefully crafted to fool a model by adding small, often imperceptible, perturbations to benign samples.

The vulnerability of manually designed ANNs to adversarial examples has been extensively studied, but the same cannot be said about evolved ANNs. As such, we conducted a study to evaluate the adversarial robustness of CNNs designed by two NE approaches, namely DENSER and NSGA-Net. Although the models were validated from a predictive accuracy perspective, the original works did not contemplate any adversarial robustness assessment, nor were the models explicitly designed to be robust in that sense.

The results show that the DENSER and the NSGA-Net models are susceptible to adversarial attacks, with the accuracy almost always dropping to zero (or close to zero). This constitutes clear evidence that more efforts should be made toward incorporating this kind of analysis in the design and validation of NE approaches. Therefore, the standard scenario where the main (and often single) focus is on predictive accuracy over clean data should be extended to accommodate other concerns.

At the same time, the DENSER model is, to a certain degree, robust to L_2 perturbations. This behavior was not explicitly enforced by the evolutionary process, but it shows that future approaches may succeed in discovering novel architectures that are inherently more robust. Designing such a method is a path we are currently investigating, starting from a more in-depth analysis of the DENSER model and its architectural characteristics. This analysis may reveal some of the ingredients responsible for the slightly higher robustness under that threat model, valuable knowledge that can guide key design choices of a robustness-aware NE method.

In this study, we tried to be as faithful as possible to the original works, but that came with some disadvantages. Namely, each model was trained under slightly different configurations, including distinct data pre-processing procedures. Following

the discussion on benchmarking EML methods, it would be interesting to re-train all the models under the exact same conditions so as to, for instance, make sure that the observed differences truly arise from architectural aspects. Furthermore, we analyzed pre-trained models for a single dataset and strictly considered white-box attacks. Those two factors constitute clear targets for improvement in future work. Nevertheless, current frameworks, like BenchENAS [61], are not prepared to support adversarial robustness assessments yet. Moreover, once the need to compare robustness-aware evolutionary approaches arises, even further challenges will have to be overcome to build a sound benchmark.

Acknowledgements This work is funded by the FCT—Foundation for Science and Technology, I.P./MCTES through national funds (PIDDAC), within the scope of CISUC R&D Unit—UIDB/00326/2020 or project code UIDP/00326/2020. It is also partially supported by the Portuguese Recovery and Resilience Plan (PRR) through project C645008882-00000055, Center for Responsible AI. The first author is also partially funded by FCT under the individual grant UI/BD/151047/2021.

References

1. Alzantot, M., Sharma, Y., Chakraborty, S., Zhang, H., Hsieh, C.-J., Srivastava, M.B.: GenAttack: practical black-box attacks with gradient-free optimization. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1111–1119 (2019)
2. Andriushchenko, M., Croce, F., Flammarion, N., Hein, M.: Square attack: a query-efficient black-box adversarial attack via random search. In: ECCV (2020)
3. Assunção, F., Lourenço, N., Machado, P., Ribeiro, B.: DENSER: deep evolutionary network structured representation (2018). [arXiv:1801.01563](https://arxiv.org/abs/1801.01563)
4. Athalye, A., Carlini, N., Wagner, D.: Obfuscated gradients give a false sense of security: circumventing defenses to adversarial examples. In: International Conference on Machine Learning (2018)
5. Benz, P., Zhang, C., Ham, S., Karjauv, A., Kweon, I.S.: Robustness comparison of vision transformer and MLP-mixer to CNNs. In: CVPR 2021 Workshop on Adversarial Machine Learning in Real-World Computer Vision Systems and Online Challenges (AML-CV) (2021)
6. Brendel, W., Rauber, J., Bethge, M.: Decision-based adversarial attacks: reliable attacks against black-box machine learning models. In: International Conference on Learning Representations (2018)
7. Brown, T.B., Mané, D., Roy, A., Abadi, M., Gilmer, J.: Adversarial patch (2017). [arXiv:1712.09665](https://arxiv.org/abs/1712.09665)
8. Carlini, N., Athalye, A., Papernot, N., Brendel, W., Rauber, J., Tsipras, D., Goodfellow, I., Madry, A., Kurakin, A.: On evaluating adversarial robustness (2019). [arXiv:1902.06705](https://arxiv.org/abs/1902.06705)
9. Carlini, N., Wagner, D.A.: Towards evaluating the robustness of neural networks. In: 2017 IEEE Symposium on Security and Privacy (SP), pp. 39–57. IEEE Computer Society (2017)
10. Croce, F., Andriushchenko, M., Sehwag, V., Debenedetti, E., Flammarion, N., Chiang, M., Mittal, P., Hein, M.: RobustBench: a standardized adversarial robustness benchmark (2020). [arXiv:2010.09670](https://arxiv.org/abs/2010.09670)
11. Croce, F., Hein, M.: Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In: Daumé III, H., Singh, A. (eds.) Proceedings of the 37th International Conference on Machine Learning, vol. 119, pp. 2206–2216. Proceedings of Machine Learning Research (PMLR), 13–18 Jul 2020 (2020)

12. Devaguptapu, C., Agarwal, D., Mittal, G., Gopalani, P., Balasubramanian, V.N.: On adversarial robustness: a neural architecture search perspective. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops, pp. 152–161 (2021)
13. DeVries, T., Taylor, G.W.: Improved regularization of convolutional neural networks with cutout (2017). [arXiv:1708.04552](https://arxiv.org/abs/1708.04552)
14. Dong, Y., Liao, F., Pang, T., Su, H., Zhu, J., Hu, X., Li, J.: Boosting adversarial attacks with momentum. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 9185–9193 (2018)
15. Dong, Y., Fu, Q.-A., Yang, X., Pang, T., Su, H., Xiao, Z., Zhu, J.: Benchmarking adversarial robustness on image classification. In: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 318–328 (2020)
16. Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., Houlsby, N.: An image is worth 16x16 words: transformers for image recognition at scale. In: International Conference on Learning Representations (2021)
17. Eykholt, K., Evtimov, I., Fernandes, E., Li, B., Rahmati, A., Xiao, C., Prakash, A., Kohno, T., Song, D.: Robust physical-world attacks on deep learning visual classification. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1625–1634 (2018)
18. Geraeinejad, V., Sinaei, S., Modarressi, M., Daneshtalab, M.: RoCo-NAS: robust and compact neural architecture search. In: 2021 International Joint Conference on Neural Networks (IJCNN), pp. 1–8. IEEE (2021)
19. Gilmer, J., Adams, R.P., Goodfellow, I.J., Andersen, D.G., Dahl, G.E.: Motivating the rules of the game for adversarial example research (2018). [arXiv:1807.06732](https://arxiv.org/abs/1807.06732)
20. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016)
21. Goodfellow, I.J., Shlens, J., Szegedy, C.: Explaining and harnessing adversarial examples. In: International Conference on Learning Representations (2015)
22. Guo, M., Yang, Y., Xu, R., Liu, Z., Lin, D.: When NAS meets robustness: in search of robust architectures against adversarial attacks. In: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 628–637 (2020)
23. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 770–778 (2016)
24. Hendrycks, D., Carlini, N., Schulman, J., Steinhardt, J.: Unsolved problems in ML safety (2021). [arXiv:2109.13916](https://arxiv.org/abs/2109.13916)
25. Hendrycks, D., Dietterich, T.: Benchmarking neural network robustness to common corruptions and perturbations. In: International Conference on Learning Representations (2019)
26. Hu, J., Shen, L., Sun, G.: Squeeze-and-excitation networks. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 7132–7141 (2018)
27. Huang, H., Wang, Y., Erfani, S., Quanquan, G., Bailey, J., Ma, X.: Exploring architectural ingredients of adversarially robust deep neural networks. *Adv. Neural Inf. Process. Syst.* **34**, 5545–5559 (2021)
28. Jacobsen, J.-H., Behrmann, J., Carlini, N., Tramèr, F., Papernot, N.: Exploiting excessive invariance caused by norm-bounded adversarial robustness (2019). [arXiv:1903.10484](https://arxiv.org/abs/1903.10484)
29. Kar, O.F., Yeo, T., Atanov, A., Zamir, A.: 3d common corruptions and data augmentation. In: 2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 18941–18952 (2022)
30. Katz, G., Barrett, C., Dill, D.L., Julian, K., Kochenderfer, M.J.: Reluplex: an efficient SMT solver for verifying deep neural networks. In: International Conference on Computer Aided Verification, pp. 97–117. Springer (2017)
31. Krizhevsky, A.: Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009
32. Kurakin, A., Goodfellow, I., Bengio, S.: Adversarial examples in the physical world (2016). [arXiv:1607.02533](https://arxiv.org/abs/1607.02533)

33. Kurakin, A., Goodfellow, I.J., Bengio, S.: Adversarial machine learning at scale (2016). [arXiv:1611.01236](https://arxiv.org/abs/1611.01236)
34. López-Ibáñez, M., Branke, J., Paquete, L.: Reproducibility in evolutionary computation. ACM Trans. Evol. Learn. Optim. **1**(4) (2021)
35. Lu, Z., Whalen, I., Boddeti, V., Dhebar, Y., Deb, K., Goodman, E., Banzhaf, W.: NSGA-Net: neural architecture search using multi-objective genetic algorithm. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 419–427 (2019)
36. Madry, A., Makelov, A., Schmidt, L., Tsipras, D., Vladu, A.: Towards deep learning models resistant to adversarial attacks. In: International Conference on Learning Representations (2018)
37. Metzen, J.H., Genewein, T., Fischer, V., Bischoff, B.: On detecting adversarial perturbations. In: International Conference on Learning Representations (2017)
38. Moosavi-Dezfooli, S., Fawzi, A., Fawzi, O., Frossard, P.: Universal adversarial perturbations. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 86–94 (2017)
39. Moreno-Torres, J.G., Raeder, T., Alaíz-Rodríguez, R., Chawla, N.V., Herrera, F.: A unifying view on dataset shift in classification. Pattern Recognit. **45**(1), 521–530 (2012). Jan
40. Nguyen, A., Yosinski, J., Clune, J.: Deep neural networks are easily fooled: high confidence predictions for unrecognizable images. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 427–436 (2015)
41. Nicolae, M.-I., Sinn, M., Tran, M.N., Buesser, B., Rawat, A., Wistuba, M., Zantedeschi, V., Baracaldo, N., Chen, B., Ludwig, H., Molloy, I., Edwards, B.: Adversarial robustness toolbox v1.2.0 (2018). [arXiv:1807.01069](https://arxiv.org/abs/1807.01069)
42. Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C.L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P.F., Leike, J., Lowe, R.: Training language models to follow instructions with human feedback (2022). [arXiv:2203.02155](https://arxiv.org/abs/2203.02155)
43. Papernot, N., McDaniel, P., Goodfellow, I., Jha, S., Berkay Celik, Z., Swami, A.: Practical black-box attacks against machine learning. In: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, pp. 506–519. ACM (2017)
44. Raschka, S.: Model evaluation, model selection, and algorithm selection in machine learning (2018). [arXiv:1811.12808](https://arxiv.org/abs/1811.12808)
45. Russakovsky, O., Deng, J., Hao, S., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L.: ImageNet large scale visual recognition challenge. Int. J. Comput. Vis. (IJCV) **115**(3), 211–252 (2015)
46. Sinn, M., Wistuba, M., Buesser, B., Nicolae, M.-I., Tran, M.: Evolutionary search for adversarially robust neural networks. In: Safe Machine Learning workshop at ICLR (2019)
47. Stanley, K.O.: Compositional pattern producing networks: a novel abstraction of development. Genet. Program Evolvable Mach. **8**, 131–162 (2007)
48. Storn, R., Price, K.: Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. J. Global Optim. **11**, 341–359 (1997)
49. Su, J., Vargas, D.V., Sakurai, K.: One pixel attack for fooling deep neural networks. IEEE Trans. Evol. Comput. **23**(5), 828–841 (2019)
50. Suganuma, M., Shirakawa, S., Nagao, T.: A genetic programming approach to designing convolutional neural network architectures. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 497–504 (2017)
51. Sutskever, I., Martens, J., Dahl, G., Hinton, G.: On the importance of initialization and momentum in deep learning. In: International Conference on Machine Learning (2013)
52. Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I.J., Fergus, R.: Intriguing properties of neural networks. In: International Conference on Learning Representations (2014)
53. Taori, R., Dave, A., Shankar, V., Carlini, N., Recht, B., Schmidt, L.: Measuring robustness to natural distribution shifts in image classification. In: Advances in Neural Information Processing Systems, vol. 33 (2020)

54. Telikani, A., Tahmassebi, A., Banzhaf, W., Gandomi, A.H.: Evolutionary machine learning: a survey. *ACM Comput. Surv.* **54**(8) (2021)
55. Tjeng, V., Xiao, K.Y., Tedrake, R.: Evaluating robustness of neural networks with mixed integer programming. In: International Conference on Learning Representations (2019)
56. Tolstikhin, I.O., Houlsby, N., Kolesnikov, A., Beyer, L., Zhai, X., Unterthiner, T., Yung, J., Steiner, A., Keysers, D., Uszkoreit, J., Lucic, M., Dosovitskiy, A.: MLP-Mixer: an all-MLP architecture for vision (2021). [arXiv: 2105.01601](https://arxiv.org/abs/2105.01601)
57. Tramèr, F., Carlini, N., Brendel, W., Madry, A.: On adaptive attacks to adversarial example defenses. In: Conference on Neural Information Processing Systems (NeurIPS) (2020)
58. Valentim, I., Lourenço, N., Antunes, N.: Adversarial robustness assessment of neuroevolution approaches. In: 2022 IEEE Congress on Evolutionary Computation (CEC) (2022)
59. Vargas, D.V., Kotyan, S.: Evolving robust neural architectures to defend from adversarial attacks (2019). [arXiv:1906.11667](https://arxiv.org/abs/1906.11667)
60. Webb, G.I., Hyde, R., Cao, H., Nguyen, H.-L., Petitjean, F.: Characterizing concept drift. *Data Min. Knowl. Disc.* **30**(4), 964–994 (2016)
61. Xie, X., Liu, Y., Sun, Y., Yen, G.G., Xue, B., Zhang, M.: BenchENAS: a benchmarking platform for evolutionary neural architecture search. *IEEE Trans. Evol. Comput.* **26**(6), 1473–1485 (2022)
62. Yang, J., Wang, P., Zou, D., Zhou, Z., Ding, K., Peng, W., Wang, H., Chen, G., Li, B., Sun, Y., Du, X., Zhou, K., Zhang, W., Hendrycks, D., Li, Y., Liu, Z.: OpenOOD: benchmarking generalized out-of-distribution detection. In: Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (2022)
63. Yin, D., Gontijo Lopes, R., Shlens, J., Cubuk, E.D., Gilmer, J.: A fourier perspective on model robustness in computer vision. In: Advances in Neural Information Processing Systems, vol. 32 (2019)
64. Yuan, X., He, P., Zhu, Q., Li, X.: Adversarial examples: attacks and defenses for deep learning. *IEEE Trans. Neural Netw. Learn. Syst.* **30**(9), 2805–2824 (2019)
65. Zagoruyko, S., Komodakis, N.: Wide residual networks. In: Proceedings of the British Machine Vision Conference (BMVC). BMVA Press (2016)
66. Zoph, B., Vasudevan, V., Shlens, J., Le, Q.V.: Learning transferable architectures for scalable image recognition. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 8697–8710 (2018)

Chapter 16

Evolutionary Approaches to Explainable Machine Learning



Ryan Zhou and Ting Hu

Abstract Machine learning models are increasingly being used in critical sectors, but their black-box nature has raised concerns about accountability and trust. The field of explainable artificial intelligence (XAI) or explainable machine learning (XML) has emerged in response to the need for human understanding of these models. Evolutionary computing, as a family of powerful optimization and learning tools, has significant potential to contribute to XAI/XML. In this chapter, we provide a brief introduction to XAI/XML and review various techniques in current use for explaining machine learning models. We then focus on how evolutionary computing can be used in XAI/XML, and review some approaches which incorporate EC techniques. We also discuss some open challenges in XAI/XML and opportunities for future research in this field using EC. Our aim is to demonstrate that evolutionary computing is well-suited for addressing current problems in explainability and to encourage further exploration of these methods to contribute to the development of more transparent, trustworthy, and accountable machine learning models.

16.1 Introduction

As the use of machine learning models becomes increasingly widespread in various domains, including critical sectors of society such as finance and healthcare, there is a growing need for human understanding of such models. Although machine learning can detect complex patterns and relationships in data, decisions based on the predictions made by these models can have real-world impacts on human lives. The use of machine learning models in high-stakes applications such as medicine, job hiring, and criminal justice has raised concerns about the fairness, transparency, and accountability of these models. Therefore, it is essential not only to develop accurate prediction models but also to understand and explain how these predictions are being made.

R. Zhou · T. Hu (✉)

School of Computing, Queen's University, Kingston, ON K7L 2N8, Canada

e-mail: ting.hu@queensu.ca

The field of explainable artificial intelligence (XAI) or explainable machine learning (XML) has emerged in response to this need [37]. XAI research aims to develop methods to explain the decisions, predictions, or recommendations made by machine learning models in a way that is understandable to humans. These explanations act as a foundation to build trust and improve the robustness of a model by highlighting biases and failures, allow researchers to better understand, validate and debug the model, ensure compliance with regulations, and improve human-machine interaction by giving users a better understanding of when they can rely on a model's decisions.

In this book chapter, we aim to provide an overview of XAI/XML and the role of evolutionary computing (EC) in this field. In Sect. 16.2, we introduce the concept of XML and review current techniques used in the field. In Sect. 16.3, we will focus on evolutionary techniques for explaining machine learning models. Specifically, we will cover data visualization, feature selection and engineering, local and global explanations, counterfactual and adversarial examples. We will also discuss evolutionary methods designed specifically for explaining deep learning models, as well as evolutionary methods for assessing the quality of explanations themselves. In Sect. 16.4, we identify some challenges remaining in the field of XAI/XML and discuss future opportunities for incorporating EC. Finally, Sect. 16.5 provides concluding remarks for the chapter.

16.2 Explainable Machine Learning

Explainability and the fields of XAI/XML are concerned with extracting insights from machine learning models in order to make them more transparent, interpretable and understandable to humans. The goal of XAI/XML is to develop methods that can provide clear explanations of the decision-making process and enable humans to understand what relationships a model has learned and how the model arrived at its conclusions.

More concretely, the types of questions we wish to answer with an explanation include [4]:

- Are the patterns the model is drawing on to make its prediction the ones we expect?
- Why did the model make this prediction instead of a different one, and what would it take to make it change its prediction?
- Is the model biased and are the decisions made by the model fair?

Explanations can be provided in many forms; examples include visualizations, numerical values, data instances, or text explanations [37].

16.2.1 Interpretability Versus Explainability

The terms interpretability and explainability are often used interchangeably by researchers. However, in this chapter we distinguish between them as referring to two different but related aspects of attempting to understand a model [30, 41].

For our purposes, interpretability refers to the level of transparency and ease of understanding of a model’s decision-making process. An *interpretable* model is one whose decisions can be traced and understood by a human, for example, by having a simple symbolic representation. In other words, a model is considered interpretable if its decision-making process can be understood simply by inspecting the model. For example, small decision trees are often considered interpretable because it is feasible for a human to follow the exact steps leading to the prediction.

On the other hand, even if we cannot trace the exact logic, a model can still be considered *explainable* if a human-understandable explanation can be provided for what the model is doing or why a decision is made. The more comprehensive and understandable the explanation is, the more explainable the overall system is. Explanations can be provided for the overall patterns and relationships learned by a model, or for the logic used to produce an individual prediction. Some practical methods for providing explanations include evaluating feature importance, visualizing intermediate representations, or comparing the prediction to be explained with samples from the training set. An explanation essentially acts as an interface between the human and the model, attempting to approximate the behavior or decisions made by the model in a way that is comprehensible to humans. As such, this “model of the model” must be both an accurate proxy for the model in the areas of interest but also be understandable itself.

These two terms, interpretability and explainability, also roughly correspond to two paradigms for improving the understandability of a model. The first approach, often known as “intrinsic interpretability”, “glass-box” or “interpretability by design”, focuses on the interpretability of the model and aims to improve it by simplifying or otherwise restricting its complexity [34]. A simple enough model allows the exact logic to be easily followed and understood by humans, but over-reducing the model complexity can also restrict the performance of the model. In some cases, it might not be possible to construct a simple enough model that can still effectively solve the problem at hand.

The second approach, known as “post-hoc” or “black-box” methods, focuses on explainability and aims to provide explanations by analyzing the model after it has been trained. This approach does not restrict the performance of the model in any way, but also makes it more challenging to provide informative explanations. These explanation methods are typically independent of the original model and are learned after the initial model has been trained [34]. In this chapter, we will primarily focus on this approach to understanding models. However, in some cases, an explanation for a complex model can take the form of simpler model; in this case, it is important to consider the interpretability of the explanation as well.

It is also worth noting that these two goals are not mutually exclusive, and both approaches can be used to better understand a model. The overall goal of explainability is to develop better tools to understand machine learning models, while the goal of interpretability is to construct models that can be understood using the tools we have.

16.2.2 *Explanation Methods*

There are several types of explanations, which focus on different aspects of the modeling process (Fig. 16.1). In this section, we will provide an overview of various categories, and describe some examples of each. This overview is not meant to be exhaustive, but rather to highlight the various areas where XAI techniques can be applied. For a more intensive survey of current methods in XAI, we direct the reader to recent reviews [11, 42] on the topic.

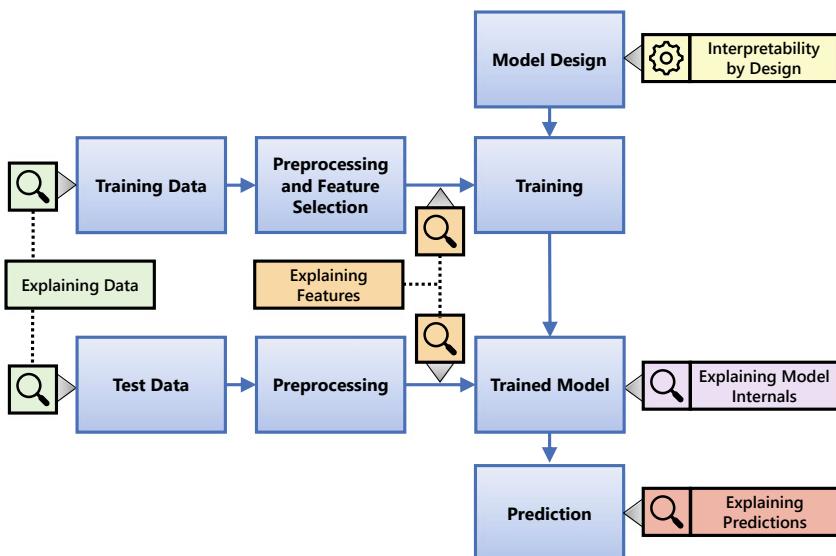


Fig. 16.1 Overview of the process of building a machine learning model, showing areas where explanations (magnifying glasses) are often applied. Examples of methods in each category are described in Sect. 16.2. Also shown is the intrinsic interpretability approach (cogwheel), where models are designed to be interpretable from the start. All these methods can be used together to form a more complete picture of a model's behavior

16.2.2.1 Explaining Data

Although this category is sometimes omitted in discussions of explainable machine learning, it is worth mentioning as part of the overall pipeline. Methods under this category do not necessarily explain the model itself but aim to explain the underlying data that a model is trained on, focusing on understanding the data distribution and its characteristics. Techniques such as exploratory analysis, data visualization and dimensionality reduction can be used to gain a better understanding of the patterns in the underlying data that the model might learn, as well as identify any potential biases. Examples of these techniques include Principal Component Analysis (PCA) [18, 39] and t-Distributed Stochastic Neighbor Embedding (t-SNE) [54], which reduce the dimensionality of data to allow for easy visualization. In addition, methods such as clustering and outlier detection can help identify patterns or anomalies in the data that may impact the model’s performance, and can aid in feature selection and engineering. These explanations can help identify data quality issues, biases, and preprocessing requirements, as well as build trust.

16.2.2.2 Explaining Features

This approach aims to explain the dependence of a model on each feature it uses. For example, feature importance returns a score that represents the significance of each feature to the model. This helps to identify which features have the greatest impact on the model’s predictions and provides insights into how the model is making decisions. This type of explanation can also be used to verify whether the model is behaving as expected—for example, by checking whether it is using the same features a human would solve the problem. In the case of a computer vision model, this type of explanation can be used to determine if the features being used to classify a particular image as a cat make sense, or if the model is using spurious patterns in the data, such as identifying the cat based on its surroundings. This type of explanation can also aid in optimizing models and performing feature selection by identifying less important features.

Some models, such as decision trees and tree ensembles like random forests, provide feature importance measures [6]. For other models, Shapley additive explanations (SHAP) [31] attempts to provide a universal method for assessing feature importance that can be applied to most machine learning models. This is based on the Shapley value, a concept from cooperative game theory that assigns a value to each player in a game based on their contribution to the overall outcome. In the context of machine learning, the “players” are the features in the data, and the “game” is the prediction task. The Shapley value scores each feature based on its contribution to each prediction. The exact calculation of Shapley values is usually computationally impractical, as it involves evaluating every possible combination of features. However, SHAP proposes approximating these values using sampling and regression, making the estimation of feature importance computationally feasible. This method is widely used in the field of XAI.

16.2.2.3 Explaining Model Internals

Methods in this category attempt to explain the internal function of the model, for example, by inspecting the structure of a tree or the weights in a neural network. These approaches can either attempt to explain the entire model, or to understand smaller components of the model.

Explaining the full model can be done by training a secondary model which both approximates the original model and is more interpretable. An example of this approach was proposed by Lakkaraju et al. [26]. Their approach approximates the behavior of the model using a small number of decision sets, providing an interpretable proxy for the entire model. While approximating the full model in this and similar ways is effective for smaller models, producing accurate but interpretable proxy models becomes increasingly difficult as the size of the model increases. For large models with sufficiently complex behavior, the proxy model will either be a poor approximation or will itself be so large that it becomes difficult to interpret.

This difficulty has led to the development of methods which attempt to explain smaller sub-components of a large model. Examining sub-components can allow us to break down the overall function in a modular way, or to identify the parts of a model which are responsible for certain decisions. For example, one way to explain the function of a neuron in a neural network is by finding or constructing an input that will maximize its activation [32]. This produces an idealized version of the input that the neuron activates on.

In recent years, many methods have been developed for explaining the internals of deep learning models [43]. This is due to the rise in popularity of such models and their inherent black-box nature and large size, making explanations for them challenging but particularly important. As an example of one such method, Interpretable Lens Variable Models [1] train an invertible mapping from a complex internal representation inside a neural network (the latent space in a generative or discriminative model) to a simpler, interpretable one. A user supplies some side information in the form of a few examples which illustrate the properties they are interested in being represented explicitly, such as rotation and scaling. Then, the mapping is learned while constraining certain dimensions in the interpretable representation to obey a linear relationship with the side information. Since the mapping is invertible, the output of the model can also be controlled by making changes to the interpretable representation. The authors' experiments showed that this method can enable the user to control these properties of the images generated by the model in an intuitive way.

16.2.2.4 Explaining Predictions

This type of approach aims to explain a specific prediction made by a model. As such, the explanation only needs to capture the behavior of the model with respect to the prediction in question, rather than the model as a whole.

One popular approach in this category is Local Interpretable Model-agnostic Explanations (LIME) [40]. LIME explains a prediction by sampling a set of instances similar to the input to be explained. It obtains predictions for each of these instances and then fits a linear model to the sampled set of inputs and predictions. This creates a linear approximation for the model in the local neighborhood surrounding the instance to be explained. Although this explanation does not necessarily reflect the global behavior of the model, it is locally faithful and allows us to understand the behavior of the model around that point.

Counterfactual explanations are another type of explanation which provide information through a hypothetical example in which the model would have made a different decision. For example, “the model would have approved a loan if your income were \$5000 higher” is a counterfactual illustrating how the input would need to be changed in order to get a different result from the model [56]. This type of explanation is intuitive and can be performed on a model in a black-box manner, without any access to a model’s internals. One advantage of counterfactual explanations is that they can provide users with *recourse*, a concrete set of changes that could be made to change the decision to a different one [23]. Additionally, because they directly operate on the model’s evaluation, they are always faithful to the model’s true behavior. However, because they consist of single instances or data points, they provide less insight into the model’s overall behavior compared to a more comprehensive explanation.

Diverse Counterfactual Explanations (DiCE) [35] is one common method of constructing counterfactual predictions. The aim of this method is to produce counterfactuals which are valid (produce a different result when fed into the model), proximal (are similar to the input), and diverse (different from each other). Diversity is desirable here as it increases the likelihood of finding a useful explanation and provides a more complete picture of the model’s behavior. DiCE generates a diverse set of counterfactual examples using a diversity metric based on determinantal point processes [24], a probabilistic model which can solve subset selection problems under diversity constraints. This diversity constraint forces the various examples apart, while an additional proximity constraint forces the examples to lie close to the original input. The method also attempts to make the counterfactual examples differ from the input in as few features as possible (feature sparsity).

16.2.2.5 Other Considerations

When choosing an explanation method, there are some other factors to consider, such as whether a method is *model-specific* or *model-agnostic*, and whether it provides *global* or *local* explanations.

Whether an explanation is considered model-specific or model-agnostic is based on whether the technique is restricted to certain types of models, or if it can be applied to multiple types of models. For instance, some methods of generating feature importance measures [6] are specific to random forests and cannot be directly used to generate feature importance for a neural network. Conversely, some methods

are designed for extracting information from the internal representation of neural networks, and cannot be applied to random forests. These methods are model-specific. On the other hand, Shapley additive explanations (SHAP) can be applied to assess feature importance for any of the commonly used machine learning models, making them model-agnostic.

In addition, explanations can be local or global. Local explanations aim to provide an explanation for a specific input or group of inputs, while global explanations aim to explain the behavior of the entire model. For instance, a global explanation technique such as partial dependence plots [16] or feature importance identifies influential features across the entire training dataset or model. On the other hand, LIME fits its linear model to the neighborhood of a specific prediction, making it a local explanation as it only deals with the behavior of the model in the specific region near the prediction. While global explanations may appear to be preferable to local explanations, in practice explanations are always constrained by the amount of information a human can grasp at one time. As such, while global explanations can provide information about overall trends in the data or model, local explanations are useful for providing more detail about specific aspects of interest.

16.3 Evolutionary Approaches to Explainable Machine Learning

16.3.1 Why Use EC?

Evolutionary computing (EC) is an approach to automatic adaptation inspired by the principles of biological evolution that has been successfully applied to various fields, including optimization, learning, engineering design, and artificial life. This approach mimics the process of natural selection, allowing the computer to evolve and optimize solutions to complex problems by combining and improving upon existing solutions. Common techniques within EC include evolutionary algorithms such as genetic algorithms, genetic programming, evolution strategies, and swarm intelligence algorithms such as particle swarm optimization. These techniques use iterative processes of selection, recombination, mutation, and adaptation to improve the performance of the models.

EC has unique strengths that make it well-suited for handling the challenges of XAI. First of all, EC can work with symbolic representations or interpretable models such as decision trees or rule systems. This can be used to guarantee interpretability of the evolved representation. If EC is used to evolve an explanation using interpretable components, the explanation will be interpretable. To leverage this, a common approach is to evolve an interpretable approximation of the model, in whole or in part.

Evolutionary methods are also highly flexible, being able to perform black-box optimization, and usually being derivative-free. As a result, they can be applied to

a variety of models without requiring knowledge of the model’s internal logic—for example, when access to a model is only available through an API which returns only the predictions. This flexibility also enables the optimization of unusual or customized metrics without requiring in-depth knowledge of how to optimize the metric. Additionally, the flexibility of evolutionary methods allows them to be used to create hybrid methods with other algorithms or to serve as meta-learning methods.

EC is advantageous for XAI as it can handle multiple objectives simultaneously. This is important as explanations by nature require both faithfulness to the model as well as interpretability to humans. The population-based nature of evolutionary algorithms also enables explicit use of diversity metrics. This allows us to explore a range of different explanations, increasing the chances of finding a useful one.

We will now delve into current methods which employ EC for producing explanations, and provide an overview of these approaches and their applications. There are a number of different ways to generate explanations, and as before, we will discuss them based on the stage in the machine learning modeling process in which they can be applied, highlighting works of interest. For an extensive survey on these methods, we invite the reader to explore recent reviews [4, 33] on the use of evolutionary methods in XAI.

16.3.2 *Explaining Data*

EC can be used to explain data by means of dimensionality reduction and visualization. One approach is GP-tSNE [27], which adapts the classic t-SNE [54] algorithm to use evolved trees to provide an interpretable mapping from the original data points to the embedded points. Similarly, Schofield and Lensen [46] use tree-GP to produce an interpretable mapping for Uniform Manifold Approximation and Projection (UMAP). By producing an explicit mapping function rather than simply the embedded points, we can not only make the process more transparent but also reuse the mapping on new data.

In some cases, we may want to use the lower dimensional representation for prediction as well as visualization. This is useful for interpretability as it allows us to visualize exactly the same data representation that the model sees. Therefore, another approach is to construct features which are both amenable to visualization and well-suited for downstream tasks. Icke and Rosenberg [22] proposed a multi-objective GP algorithm to optimize three objective measures desirable for constructed features—classifiability, visual interpretability and semantic interpretability. Similarly, Cano et al. [8] developed a method using multi-objective GP to construct features for visualization and downstream analysis, optimizing for six classification and visualization metrics. The classification metrics (accuracy, AUC and Cohen’s kappa rate) aim to improve the performance of the downstream classifier, while the visualization metrics (C-index, Davies-Bouldin index, Dunn’s index) aim to improve the clustering and separability of the features.

16.3.3 Feature Selection and Feature Engineering

Feature selection is a common preprocessing step where a relevant subset of features is selected from the original dataset. This is used to improve the performance of the model, but also has benefits for interpretability by narrowing down the features the model can draw on. As an explanation, feature selection shares some similarities with feature importance, which identifies the features a model is drawing on, but instead restricts the model explicitly so it can only draw on the chosen features.

Genetic algorithms are a straightforward and effective approach to feature selection, with a natural representation in the form of strings of 1s and 0s, making them a popular choice for feature selection [44, 49, 59]. Genetic programming can also be used for feature selection since the inclusion of features in a tree or linear genetic program is intrinsically evolved with the program [19, 20, 48]. For an in-depth review of genetic programming methods, we refer the reader to [58]. Swarm intelligence methods, such as particle swarm optimization, have been applied to feature selection as well [60]. For a more detailed review of these methods, we direct the reader to [38]. In addition to selecting features for a model, feature selection can also be used to improve data understanding by integrating it with techniques such as clustering [15].

Feature engineering, also referred to as feature construction, is a related approach that involves building higher-level condensed features out of basic features. Genetic programming can be used to evolve these higher-level features for downstream tasks such as classification and regression [25, 29, 36, 55]. This can also help improve the interpretability of a model, since this can reduce a large number of low-level features to a smaller number of higher-level features which may be easier to understand for humans. Moreover, it removes some of the modeling out of the black-box and into a transparent step beforehand, thereby reducing the amount of explanation needed.

These methods also share many similarities with dimensionality reduction techniques, and in some cases can fall under both categories. Uriot et al. [53] compared a variety of multi-tree-GP algorithms for dimensionality reduction, as well as a tree-based autoencoder architecture. In the multi-tree representation, each individual in the population is a collection of trees with each tree mapping the input to one feature in the latent dimension. In order to reconstruct the input for the autoencoder, a multi-tree decoder is simultaneously evolved with one tree per input dimension. Their results showed that GP-based dimensionality reduction was on par with the conventional methods they tested (PCA, LLE, and Isomap).

16.3.4 Model Extraction

This approach, also known as a global surrogate model, aims to approximate a black-box model with an evolved interpretable model. This idea is closely related to knowledge distillation [7, 17] in deep learning, but rather than simply making the model smaller we also want to make it more interpretable.

Evans et al. [12] propose a model extraction method using multi-objective genetic programming to construct decision trees that accurately represent any black-box classifier while being more interpretable. This method aims to simultaneously maximize the ability of the tree to reconstruct (replicate) the predictions of a black-box model and also maximize interpretability by minimizing the complexity of the decision tree. The reconstruction ability is measured by the weighted F1 score over cross-validation, and the complexity of the decision tree is measured by the number of splitting points in the tree. The overall evolutionary process uses a modified version of NSGA-II [10]. In their experiments on a range of classification problems, they found that the accuracy remained commensurate with other models extraction methods (Bayesian rule lists, logistic regression, and two types of decision trees) while significantly reducing the complexity of the models produced.

16.3.5 *Local Explanations*

Instead of creating an interpretable model to approximate the global performance of a black-box model, which may not be possible, these approaches only attempt to approximate the local behavior using an evolutionary algorithm. Much work in this area is inspired by LIME, but enhance the relatively simple sampling strategy and linear approximation with evolutionary components.

Ferreira et al. [13] proposed Genetic Programming Explainer (GPX), a GP-based method which fits a local explanation model for a given input example. Similar to LIME, when given a sample input to be explained, the method samples a set of neighboring data points around the input and fits a local explanation. However, rather than a linear model, GPX uses a GP to evolve symbolic expression trees that best capture the behavior of the pre-trained black-box model over the neighboring data points. The authors tested this on both classification and regression datasets and reported that the GP captured the model's behavior better than LIME, as the assumption of linear local behavior was not always valid, and also outperformed a decision tree used as an explainer for the same neighbor set.

On the other hand, Guidotti et al. [14] proposed Local Rule-based Explanations (LORE), which applies an evolutionary algorithm to neighborhood generation rather than evolving the explanation itself. A genetic algorithm generates a set of points near the prediction to be explained, which are either classified the same as or differently from the original prediction while being nearby. A decision tree is then used to fit the local behavior of the black-box model. The use of a genetic algorithm here ensures a dense sampling of points in the local neighborhood which lie on both sides of the decision boundary.

16.3.6 Counterfactuals

Another line of work is in creating counterfactuals using EC. An EA is well suited to this task as a black-box, possibly multi-objective optimizer, as it allows us to find counterfactuals without knowing the internal workings of the model while also optimizing for multiple desirable criteria in the counterfactuals.

CERTIFAI [50] generates a population of counterfactual explanations using a model-agnostic genetic algorithm. The initial population is generated by sampling instances which lie on the other side of the decision boundary of the model (i.e., are classified differently from the instance to be explained). Then, the genetic algorithm optimizes the population to minimize the distance (for some notion of distance, depending on the type of data) from each counterfactual instance to the input instance. The population is then analyzed for robustness, which increases if the best counterfactual examples found are farther away from the input, and fairness, which is measured by comparing robustness across different values of a particular feature.

GeCo [45] uses a genetic algorithm with feasibility and plausibility constraints on the features, specified using the constraint language PLAF. This allows certain counterfactuals which would be useless to the user (e.g., counterfactuals where the user changes their gender or decreases their age) to be ruled out. Similar to CERTIFAI, the genetic algorithm minimizes the distance from the input instance to the counterfactual examples, prioritizing examples on the other side of the decision boundary but also keeping examples which are close to the decision boundary if not enough counterfactuals are available. The fitness function does not consider how many features are changed relative to the input instance (with a smaller number being preferred for ease of understanding), but the algorithm is biased toward a smaller number of changes by initializing the population with only one feature changed.

Multi-objective counterfactuals (MOC) [9] explicitly use multi-objective optimization to consider multiple desirable properties of the explanations. MOC uses a modified version of NSGA-II to perform its search. Among the changes are the use of mixed integer evolution strategies (MIES) [28] to search a mixed discrete and continuous space, and a different crowding distance sorting algorithm which prioritizes diversity in feature space. A total of four objectives are used, optimizing for these four desirable properties: the model output for the example should be close to the desired output; the example should lie close in feature space to the input to be explained; the example should not differ from the input in too many features; and the example is plausible (likely to be drawn from the same distribution as the real data), measured by its distance to the closest k data points.

16.3.7 Explaining Deep Learning

Thus far, the methods we have covered are general and can be used with a variety of models. However, with the popularity of deep learning methods, we would be remiss

not to discuss some specific methods tailored for deep learning models. While deep learning models are large, they are also differentiable allowing for hybrid methods combining gradient information with evolution.

For image classification, the large number of input features (pixels) presents a significant problem for many explanation methods. As such, it is necessary to reduce the dimension first, for example by clustering similar pixels into “superpixels”. Wang et al. [57] propose using a multi-objective genetic algorithm to identify superpixels of importance for the final prediction, and using this set of superpixels as an explanation. The genetic algorithm uses NSGA-II to optimize for the least number of superpixels used, while maximizing the model’s confidence in its prediction.

Adversarial examples are closely related to counterfactuals. An adversarial example is a counterfactual example but with the intent of creating an incorrect prediction [34]. This is done by applying a small perturbation to an example to change its classification. Most approaches search for examples which are as close to the original input as possible, and perceptually similar to the input. These examples are a method to highlight failure modes of the model as well as a potential attack vector on deep learning models.

Su et al. [51] propose a method of finding adversarial examples which modify only one pixel in an image. This is in contrast to previous methods which modify multiple pixels in the image and are more obvious to humans. Their method uses differential evolution, where each individual is encoded by the coordinate of the pixel to be modified and the perturbation in RGB space. They find that in many cases one pixel is sufficient to deceive the model.

Adversarial examples are also present in models built for other domains, such as natural language processing. Alzantot et al. [2] generate adversarial examples on a sentiment analysis model and a textual entailment model. In addition, the examples they produce are designed to be semantically and syntactically similar to the original input, making the attack more difficult to spot. A genetic algorithm is used to optimize for a different target label than the original. Mutation is done by changing words in the input to similar words as measured by a word embedding model (GloVe) and filtering out words which do not fit the context.

16.3.8 Assessing Explanations

Finally, rather than using EC to generate the explanations themselves, we will discuss some ways that it can be used to assess or improve the quality of other explanation methods.

Huang et al. [21] propose two metrics to assess the robustness of an explanation: worst-case misinterpretation discrepancy and probabilistic interpretation robustness. Interpretation discrepancy measures the difference between two interpretations, one before and one after perturbation of the input. It is desirable for this value to be low for a interpretation to be robust to perturbation. They then measure the discrepancy in two worst cases: the largest interpretation discrepancy possible while still being classified

as the same class, and the smallest interpretation discrepancy possible while being classified differently (adversarial example). These values are optimized for using a GA. The other metric, probability of misinterpretation, calculates probabilistic versions of the above: the probability of an example having the same classification but significantly different interpretations, and the probability of an example having a different classification but similar interpretation. This is estimated using subset simulation.

It is also possible to perform an adversarial attack on the explanations themselves. Tamam et al. [52] do this with AttaXAI, a black-box approach based on evolution. AttaXAI tries to evolve an image similar in appearance to the original input and produce the same prediction from the model but with an arbitrary explanation map. In their experiments, pairs of images were selected and they showed that they were able to generate a new image with the appearance and prediction of the first image, but with a similar explanation map to the second.

16.4 Research Outlook

16.4.1 Challenges

One major challenge for evolutionary approaches to XAI is scalability. As data continues to grow and machine learning models become increasingly complex, the number of parameters and features to be optimized grows as well. As such, methods which work well on small models and datasets may become too expensive on larger ones. However, large models are the most opaque and most in need of explanation, so improving the scalability of XAI methods is necessary to ensure they can be applied to even the largest models. In particular, producing fully interpretable global explanations which accurately capture behavior yet are still simple enough to understand may become too challenging as models become larger—necessitating more local explanations or a more focused approach concentrating on explaining particular properties or components of the model. We also see here the potential for more use of automated approaches to explainability—for example, by using evolutionary search to find local explanations of interest and optimize for particular properties. This idea has been explored with counterfactual examples, but it could be extended to other types of explanations.

Another challenge for explainability is the incorporation of domain knowledge. This can include knowledge from subject matter experts, as well as prior knowledge about the dataset or problem. Current approaches to XAI are broad and aim to provide explanations which are independent of the problem setting, or at most are model-specific rather than problem-specific. However, it can be useful to see how well a solution found by a machine learning model aligns with current knowledge in the field to evaluate the quality of the solution, or conversely, to identify areas where the model deviates from current understanding. For example, a practitioner may want

to see how well the gene associations found by a genomics model aligns with the literature, as well as which associations are novel. This domain knowledge can be provided in the form of expert rules, constraints, or structured data such as a graph structure or tree. Domain knowledge can also be incorporated into the model-building process to improve interpretability, for instance by constraining the models to focus on associations known to be plausible (e.g., by incorporating causality) or excluding irrelevant features.

16.4.2 Opportunities

We see some additional opportunities for future work employing EC for XAI. One promising direction in current research is the use of multiple objectives to optimize explanations. Explainability is inherently a multi-objective problem, requiring the explanation to both be faithful to the model as well as being simple enough to be interpretable. EC is well-suited to explicitly optimizing for this, and we believe introducing these ideas into current and future explanation methods is a straightforward but effective way of improving the quality of explanations.

Along similar lines, the use of diversity metrics and novelty search is a unique strength available to evolutionary algorithms, which can help improve the explanations provided. The use of quality-diversity (illumination) algorithms can produce a range of explanations which are both accurate and present different perspectives on the behavior of the model. For example, a quality-diversity approach to counterfactual explanations could ensure that a range of behaviors are showcased in the examples.

Another opportunity for EC is the incorporation of user feedback, considering the evolution of explanations as an open-ended evolution process. Explainability is intended for the human user, and as such explanation quality is ultimately subjective and can only be approximated by metrics. Users may also have their own unique preferences for what constitutes a useful explanation. Incorporating user feedback into the evolution process can allow better tailored explanations that continue to improve. At the same time, better metrics for measuring the quality of an explanation are also necessary in order to not overwhelm the user.

16.4.3 Real-World Impacts

As AI becomes increasingly integrated into real-world applications, developing better methods for providing explanations is essential for ensuring safety and trust across various domains. With this in mind, it is also crucial to consider the practical effects and benefits that XAI research can have. We would like to highlight here a few application areas where work on evolutionary approaches to XAI can have a substantial impact.

Healthcare is a domain where the consequences of errors can be especially high. Untrusted models may be ignored by clinicians, wasting resources and providing no benefit. Worse, seemingly trustworthy but flawed models may cause harm to patients. Even models with few errors may exhibit systematic biases, such as diagnostic models underdiagnosing certain patient groups while appearing accurate [47]. Explainability can help identify these systematic errors and biases [3]. In the financial sector AI models are employed for fraud detection and risk assessment. Similar systematic biases in these models can also produce harm, for example, by disproportionately denying loans to certain groups. In addition, regulatory bodies often require explanations for these models to ensure compliance and maintain transparency.

Explainability also holds significant potential to advance engineering and scientific discovery. AI models are used in various engineering applications, for example, in AI-driven materials design and drug discovery, and to produce scientific insights in fields such as genomics and astrophysics. Explanations can offer insight into the underlying mechanisms and relationships, improving hypothesis generation, and validating domain knowledge.

Natural language processing has experienced many recent breakthroughs, with the development and deployment of models of unprecedented size. In particular, there is an emerging paradigm of building “foundation models”, generalist deep learning models which are trained on a wide range of data for general capabilities and which can be further fine-tuned for downstream tasks [5]. These models are capable of tasks which they are not specifically trained for, but it is still unclear how they make decisions or generate outputs. Any flaws in these foundation models may be carried over to application-specific models built on top of them. As these models become more pervasive and their applications expand, understanding them and identifying their failure modes becomes increasingly important.

16.5 Conclusion

Explainable AI/ML is an emerging field with important implications for machine learning as a whole. With the increasing use of machine learning models in real-world applications, it's more important than ever that we understand such models and what they learn. Evolutionary computing is well-poised to contribute to the field, bringing a rich toolbox of tools for performing black-box optimization. In this chapter, we introduced various paradigms for explaining a machine learning model and current methods of doing so. We then discussed how evolutionary computing can fit into these paradigms and the advantages of employing them. In particular, evolutionary computing as an optimizer is well-suited for tricky interpretability metrics which are difficult to handle due to reasons such as non-differentiability, as well as for population-based metrics such as diversity and for optimizing multiple of these metrics at the same time. We highlighted a few methods in each category which leveraged some of these strengths, but there is still significant room for more exploration and more advanced evolutionary algorithms.

The field of explainable machine learning is still new, and much knowledge remains locked away within trained models that we still do not have the means to decipher. The use of evolutionary computing for XAI is still uncommon, but there are many opportunities ripe for the picking and we believe that it has the potential to play a key part in the future of XAI.

References

1. Adel, T., Ghahramani, Z., Weller, A.: Discovering interpretable representations for both deep generative and discriminative models. In: Proceedings of the 35th International Conference on Machine Learning, pp. 50–59. PMLR (July 2018)
2. Alzantot, M., Sharma, Y., Elgohary, A., Ho, B.-J., Srivastava, M.B., Chang, K.-W.: Generating natural language adversarial examples. In: Riloff, E., Chiang, D., Hockenmaier, J., Tsujii, J. (eds.) Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, 3 Oct–4 Nov 2018, pp. 2890–2896. Association for Computational Linguistics (2018)
3. Arias-Duart, A., Parés, F., Garcia-Gasulla, D., Gimenez-Abalos, V.: Focus! rating XAI methods and finding biases. In: IEEE International Conference on Fuzzy Systems, FUZZ-IEEE 2022, Padua, Italy, 18–23 July 2022, pp. 1–8. IEEE (2022)
4. Bacardit, J., Brownlee, A.E.I., Cagnoni, S., Iacca, G., McCall, J., Walker, D.: The intersection of evolutionary computation and explainable AI. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO’22, pp. 1757–1762, New York, NY, USA, July 2022. Association for Computing Machinery (2022)
5. Bommassani, R., Hudson, D.A., Adeli, E., Altman, R.B., Arora, S., von Arx, S., Bernstein, M.S., Bohg, J., Bosselut, A., Brunskill, E., Brynjolfsson, E., Buch, S., Card, D., Castellon, R., Chatterji, N.S., Chen, A.S., Creel, K., Quincy Davis, J., Demszky, D., Donahue, C., Doumbouya, M., Durmus, E., Ermon, S., Etchemendy, J., Ethayarajh, K., Fei-Fei, L., Finn, C., Gale, T., Gillespie, L., Goel, K., Goodman, N.D., Grossman, S., Guha, N., Hashimoto, T., Henderson, P., Hewitt, J., Ho, D.E., Hong, J., Hsu, K., Huang, J., Icard, T., Jain, S., Jurafsky, D., Kalluri, P., Karamcheti, S., Keeling, G., Khani, F., Khattab, O., Koh, P.W., Krass, M.S., Krishna, R., Kuditipudi, R., et al.: On the opportunities and risks of foundation models (2021). [arXiv:2108.07258](https://arxiv.org/abs/2108.07258)
6. Breiman, L.: Random forest. *Mach. Learn.* **45**, 5–32 (2001)
7. Buciluă, C., Caruana, R., Niculescu-Mizil, A.: Model compression. In: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD’06, pp. 535–541, New York, NY, USA, Aug 2006. Association for Computing Machinery (2006)
8. Cano, A., Ventura, S., Cios, K.J.: Multi-objective genetic programming for feature extraction and data visualization. *Soft. Comput.* **21**(8), 2069–2089 (2017). April
9. Dandl, S., Molnar, C., Binder, M., Bischl, B.: Multi-objective counterfactual explanations. In: Bäck, T., Preuss, M., Deutz, A., Wang, H., Doerr, C., Emmerich, M., Trautmann, H. (eds.) Parallel Problem Solving from Nature—PPSN XVI. Lecture Notes in Computer Science, pp. 448–469, Cham, 2020. Springer International Publishing (2020)
10. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* **6**(2), 182–197 (2002)
11. Dwivedi, R., Dave, D., Naik, H., Singhal, S., Omer, R., Patel, P., Qian, B., Wen, Z., Shah, T., Morgan, G., Ranjan, R.: Explainable AI (XAI): core ideas, techniques, and solutions. *ACM Comput. Surv.* **55**(9), 194:1–194:33 (2023)
12. Evans, B.P., Xue, B., Zhang, M.: What’s inside the black box? A genetic programming method for interpreting complex machine learning models. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), pp. 1012–1020 (2019)

13. Ferreira, L.A., Guimarães, F.G., Silva., R.: Applying genetic programming to improve interpretability in machine learning models. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC), pp. 1–8 (2020)
14. Guidotti, R., Monreale, A., Ruggieri, S., Pedreschi, D., Turini, F., Giannotti, F.: Local Rule-Based Explanations of Black Box Decision Systems (2018). CoRR, [arXiv:1805.10820](https://arxiv.org/abs/1805.10820)
15. Hancer, E., Xue, B., Zhang, M.: A survey on feature selection approaches for clustering. *Artif. Intell. Rev.* **53**(6), 4519–4545 (2020)
16. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning. Springer (2001)
17. Hinton, G.E., Vinyals, O., Dean, J.: Distilling the knowledge in a neural network (2015). CoRR, [arXiv:1503.02531](https://arxiv.org/abs/1503.02531)
18. Hotelling, H.: Analysis of a complex of statistical variables into principal components. *J. Educ. Psychol.* **24**, 417–441 (1933)
19. Ting, H.: Can genetic programming perform explainable machine learning for bioinformatics? In: Genetic Programming Theory and Practice XVII. Springer (2020)
20. Hu, T., Oksanen, K., Zhang, W., Randell, E., Furey, A., Sun, G., Zhai, G.: An evolutionary learning and network approach to identifying key metabolites for osteoarthritis. *PLoS Comput. Biol.* **14**(3), e1005986 (2018)
21. Huang, W., Zhao, X., Jin, G., Huang, X.: SAFARI: versatile and efficient evaluations for robustness of interpretability (2022). CoRR, [arXiv:2208.09418](https://arxiv.org/abs/2208.09418)
22. Icke, I., Rosenberg, A.: Multi-objective genetic programming for visual analytics. In: Silva, S., Foster, J.A., Nicolau, M., Machado, P., Giacobini, M. (eds.) Genetic Programming. Lecture Notes in Computer Science, pp. 322–334. Springer, Berlin, Heidelberg (2011)
23. Karimi, A.-H., Barthe, G., Schölkopf, B., Valera, I.: A survey of algorithmic recourse: Definitions, formulations, solutions, and prospects (2020). CoRR, [arXiv:2010.04050](https://arxiv.org/abs/2010.04050)
24. Kulesza, A., Taskar, B.: Determinantal point processes for machine learning. *Found. Trends Mach. Learn.* **5**(2–3), 123–286 (2012)
25. La Cava, W., Moore, J.H.: Learning feature spaces for regression with genetic programming. *Genet. Program Evolvable Mach.* **21**(3), 433–467 (2020). September
26. Lakkaraju, H., Kamar, E., Caruana, R., Leskovec, J.: Interpretable & exploratory approximations of black box models (2017). CoRR, [arXiv:1707.01154](https://arxiv.org/abs/1707.01154)
27. Lensen, A., Xue, B., Zhang, M.: Genetic programming for evolving a front of interpretable models for data visualization. *IEEE Trans. Cybern.* **51**(11), 5468–5482 (2021)
28. Li, R., Emmerich, M.T.M., Eggermont, J., Bäck, T., Schütz, M., Dijkstra, J., Reiber, J.H.C.: Mixed integer evolution strategies for parameter optimization. *Evol. Comput.* **21**(1), 29–64 (2013)
29. Li, Z., He, J., Zhang, X., Fu, H., Qin, J.: Toward high accuracy and visualization: an interpretable feature extraction method based on genetic programming and non-overlap degree. In: Proceedings of the IEEE International Conference on Bioinformatics and Biomedicine (BIBM), pp. 299–304 (2020)
30. Lipton, Z.C.: The mythos of model interpretability. *Commun. ACM* **61**(10), 36–43 (2018)
31. Lundberg, S.M., Lee, S.-I.: A unified approach to interpreting model predictions. In: Proceedings of the 31st International Conference on Neural Information Processing Systems (NeurIPS), pp. 4768–4777 (2017)
32. Mahendran, A., Vedaldi, A.: Visualizing deep convolutional neural networks using natural pre-images. *Int. J. Comput. Vision* **120**(3), 233–255 (2016). December
33. Mei, Y., Chen, Q., Lensen, A., Xue, B., Zhang, M.: Explainable artificial intelligence by genetic programming: a survey. *IEEE Trans. Evol. Comput.* 1–1 (2022)
34. Molnar, C.: Interpretable Machine Learning: A Guide for Making Black Box Models Explainable (2022). leanpub.com
35. Mothilal, R.K., Sharma, A., Tan, C.: Explaining machine learning classifiers through diverse counterfactual examples. In: ACM Conference on Fairness, Accountability, and Transparency (Jan 2020)
36. Muhammed, M., Smith, G.D.: Evolutionary constructive induction. *IEEE Trans. Knowl. Data Eng.* **17**(11), 1518–1528 (2005). November

37. Murdoch, W.J., Singh, C., Kumbier, K., Abbasi-Asl, R., Yu, B.: Interpretable machine learning: definitions, methods, and applications. *Proc. Natl. Acad. Sci.* **116**(44), 22071–22080 (2019)
38. Nguyen, B.H., Xue, B., Zhang, M.: A survey on swarm intelligence approaches to feature selection in data mining. *Swarm Evol. Comput.* **54**, 100663 (2020)
39. Pearson, K.: LIII. On lines and planes of closest fit to systems of points in space. *Lond. Edinb. Dublin Philos. Mag. J. Sci.* **2**(11), 559–572 (1901)
40. Ribeiro, M.T., Singh, S., Guestrin, C.: “Why should I trust you?”: Explaining the predictions of any classifier. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1135–1144 (2016)
41. Rudin, C.: Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nat. Mach. Intell.* **1**, 206–215 (2019)
42. Saeed, W., Omlin, C.: Explainable AI (XAI): a systematic meta-survey of current challenges and future opportunities. *Knowl. Based Syst.* **263**, 110273 (2023). March
43. Samek, W., Montavon, G., Lapuschkin, S., Anders, C.J., Müller, K.-R.: Explaining deep neural networks and beyond: a review of methods and applications. *Proc. IEEE* **109**(3), 247–278 (2021)
44. Sayed, S., Nassef, M., Badr, A., Farag, I.: A Nested Genetic Algorithm for feature selection in high-dimensional cancer Microarray datasets. *Expert Syst. Appl.* **121**, 233–243 (2019). May
45. Schleich, M., Geng, Z., Zhang, Y., Suciu, D.: GeCo: quality counterfactual explanations in real time. *Proc. VLDB Endow.* **14**(9), 1681–1693 (2021). May
46. Schofield, F., Lenssen, A.: Using genetic programming to find functional mappings for UMAP embeddings. In: 2021 IEEE Congress on Evolutionary Computation (CEC), pp. 704–711 (June 2021)
47. Seyyed-Kalantari, L., Zhang, H., McDermott, M.B.A., Chen, I.Y., Ghassemi, M.: Underdiagnosis bias of artificial intelligence algorithms applied to chest radiographs in under-served patient populations. *Nat. Med.* **27**(12), 2176–2182 (2021)
48. Sha, C., Cuperlovic-Culf, M., Ting, H.: SMILE: systems metabolomics using interpretable learning and evolution. *BMC Bioinform.* **22**, 284 (2021)
49. Sha, Z., Hu, T., Chen, Y.: Feature selection for polygenic risk scores using genetic algorithm and network science. In: 2021 IEEE Congress on Evolutionary Computation (CEC), pp. 802–808 (June 2021)
50. Sharma, S., Henderson, J., Ghosh, J.: CERTIFAI: a common framework to provide explanations and analyse the fairness and robustness of black-box models. In: Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society, pp. 166–172, New York NY USA, Feb 2020. ACM (2020)
51. Su, J., Vargas, D.V., Sakurai, K.: One pixel attack for fooling deep neural networks. *IEEE Trans. Evol. Comput.* **23**(5), 828–841 (2019)
52. Tamam, S.V., Lapid, R., Sipper, M.: Foiling explanations in deep neural networks (2022). CoRR, [arXiv:2211.14860](https://arxiv.org/abs/2211.14860)
53. Uriot, T., Virgolin, M., Alderliesten, T., Bosman, P.A.N.: On genetic programming representations and fitness functions for interpretable dimensionality reduction. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO’22, pp. 458–466, New York, NY, USA, July 2022. Association for Computing Machinery (2022)
54. van der Maaten, L., Hinton, G.: Visualizing Data using t-SNE. *J. Mach. Learn. Res.* **9**(86), 2579–2605 (2008)
55. Virgolin, M., Alderliesten, T., Bosman, P.A.N.: On explaining machine learning models by evolving crucial and compact features. *Swarm Evol. Comput.* **53**, 100640 (2020)
56. Wachter, S., Mittelstadt, B., Russell, C.: Counterfactual explanations without opening the black box: automated decisions and the GDPR. *Harv. JL & Tech.* **31**, 841 (2017)
57. Wang, B., Pei, W., Xue, B., Zhang, M.: A multi-objective genetic algorithm to evolving local interpretable model-agnostic explanations for deep neural networks in image classification. *IEEE Trans. Evol. Comput.* 1–1 (2022)
58. Xue, B., Zhang, M., Browne, W.N., Yao, X.: A survey on evolutionary computation approaches to feature selection. *IEEE Trans. Evol. Comput.* **20**(4), 606–626 (2016). August

59. Xue, Yu., Tang, Y., Xin, X., Liang, J., Neri, F.: Multi-objective feature selection with missing data in classification. *IEEE Trans. Emerg. Top. Comput. Intell.* **6**(2), 355–364 (2022). April
60. Xue, Y., Xue, B., Zhang, M.: Self-adaptive particle swarm optimization for large-scale feature selection in classification. *ACM Trans. Knowl. Discov. Data* **13**(5), 50:1–50:27 (2019)

Chapter 17

Evolutionary Algorithms for Fair Machine Learning



Alex Freitas and James Brookhouse

Abstract At present, supervised machine learning algorithms are ubiquitously used to learn predictive models that have a major impact on people's lives. However, the vast majority of such algorithms were developed to optimise predictive accuracy only, ignoring the issue of fairness in the predictions of the learned models. This often leads to unfair predictive models, since real-world data usually contains bias or prejudices against certain groups of individuals (e.g. some gender or race). Hence, an increasingly important research area involves fairness-aware machine learning algorithms, i.e. algorithms that optimise both the predictive accuracy and the fairness of their learned predictive models, from a multi-objective optimisation perspective. In this chapter, we review fairness-aware Evolutionary Algorithms (EAs) for supervised machine learning. We first briefly provide some background concepts on fairness measures and multi-objective optimisation approaches. Then, we review six EAs for fairness-aware machine learning, which are in general based on multi-objective optimisation principles. The reviewed EAs address a variety of supervised machine learning tasks, namely: three EAs address a data pre-processing task for classification (one addressing feature construction and two addressing feature selection); one EA optimises the hyper-parameters of a base classification algorithm; one EA evolves an ensemble of artificial neural network models; and one EA finds fair counterfactuals. We conclude with a summary of the main findings of this review and some suggested future research directions.

17.1 Introduction

At present, many important decisions affecting people's lives are made by automated processes. Examples are the decisions about whether or not someone is hired for a job, or whether or not their application for a loan or mortgage is approved. The automated processes that make such decisions, or at least recommend such decisions

A. Freitas (✉) · J. Brookhouse
School of Computing, University of Kent, Canterbury, UK
e-mail: A.A.Freitas@kent.ac.uk

(which could be later approved by humans) are often based on supervised machine learning methods [41]. We focus here on classification, a major type of supervised learning task where the goal is to learn, from data, a classification model that predicts the class of an object (e.g. an individual) based on the values of a set of features (attributes or variables) describing the characteristics of each object.

Many types of supervised learning methods have been developed over decades of research [14, 41, 44], and there are now many methods that can achieve high predictive performance in general. However, the vast majority of such methods was developed to maximise predictive performance only, ignoring the issue of fairness in their predictions. Hence, when learning from data about people, they often learn unfair classification models, in the sense that the learned models often reflect biases or prejudices in the data [1, 31, 35]. In addition, algorithms can also introduce biases or prejudices into the learned model, e.g. when choosing the wrong loss function. Some precise definitions of fairness will be given in the next Section, discussing fairness measures, but for now, by unfair predictions, we informally mean the case where a desirable outcome or class (e.g. a loan approval) is predicted more often for one group of people than another, due to an unfair discrimination against the unfavoured group—e.g. discrimination against females or non-white individuals.

Hence, in recent years, there has been an increasing interest in developing classification methods that learn fair predictive models from data. In this chapter, we focus on Evolutionary Algorithms (EAs) for fair classification, which is a new and promising research area. The motivation for developing such EAs for fair classification is twofold. First, EAs are robust search and optimisation methods that are less likely to get trapped into local optima than (often greedy) local search methods [15, 37]. Second, the population-based nature of EAs facilitates multi-objective optimisation [11, 38]. Multi-objective optimisation problems arise naturally in the area of fair classification, since in general we would like to optimise at least two objectives: the predictive accuracy and the fairness of the learned classification models. Indeed, the EAs for fair classification discussed in this chapter perform, in general, multi-objective optimisation, as will be discussed later.

The remainder of this chapter is divided into four sections. The next two sections discuss some background on fairness measures and multi-objective optimisation approaches. These sections are followed by a section that briefly reviews six EAs for fair classification, in six separate subsections. One of these subsections will be substantially longer than the others because it will also report computational results comparing two versions of a GA for fair feature selection. Finally, the last section presents a general discussion and conclusions.

17.2 Background on Fairness Measures

In the context of the well-known machine learning classification task, what do fairness and bias mean? The classification task involves assigning an individual a class label using its predictive features. However, some of the predictive features could be

considered sensitive (or protected) features—e.g. gender, race and age. These sensitive features can be used to partition individuals (instances in a dataset) into two groups: the protected group contains the individuals who are considered to be subject to unfair bias and are more likely to obtain a negative outcome (class label)—e.g. being denied a loan, whilst the unprotected group contains the individuals who are seen to be in a privileged position and are more likely to obtain a positive class label.

Fairness is not a single simple concept, and the literature contains a large number of measures that capture some notion of fairness [31, 40]. We can categorise these fairness metrics into two main types: group-level and individual-level measures of fairness.

An example of a group-level fairness metrics is the discrimination score [6], which measures the difference between the prediction rates of the positive class across the protected and unprotected groups. Group-level measures can also capture differences in the different types of errors in a classification model, such as the differences in the false positive or false negative error rates across the protected and unprotected groups [7].

One limitation of group-level fairness measures is that they do not consider fairness at the individual level. That is, two very similar individuals within the same group might unfairly receive different outcomes (class labels), but a group-level fairness metric would not pick on this behaviour.

Individual-level fairness metrics avoid this limitation, as they concentrate on the similarities between individuals and their corresponding class labels. Individual-level metrics often consider all non-sensitive features which are often ignored by the group-level metrics.

One well-known individual-level fairness metric is consistency. Consistency uses a k-nearest neighbours approach to fairness, where an individual is compared to the neighbours, and if they are all assigned the same class the test is considered maximally satisfied. This test can then be performed for all individuals and an average is taken [43]. However, one disadvantage of considering all the non-sensitive features is that the neighbourhood becomes increasingly meaningless as the distances between individuals will tend to increase as the number of features increases. This leads to the comparisons of increasingly different individuals.

In reality, it is not possible to find a single measure of fairness that can be declared always the best. It has also been shown that some fairness measures cannot be simultaneously optimised and some sort of trade-off is required [7, 23, 30, 32].

We have mentioned a number of fairness measures in the last few paragraphs and will define them more precisely below, but first, we need to define some nomenclature:

- S : Protected/sensitive feature (attribute): 0 → unprotected group, 1 → protected group.
- \hat{Y} : the predicted class; Y : the actual class; both \hat{Y} and Y can take class labels 1 (positive outcome) or 0 (negative outcome).
- TP, FP, TN, FN : Number of true positives, false positives, true negatives and false negatives, respectively.

The first fairness measure we mentioned was the discrimination score (DS) [6], sometimes referred to as statistical parity, which is defined as

$$DS = |P(\hat{Y} = 1|S = 0) - P(\hat{Y} = 1|S = 1)| \quad (17.1)$$

The discrimination score is a group-level fairness measure that takes the optimal value of 0 if both protected and unprotected groups have an equal probability of being assigned to the positive class by the classifier.

The next two measures are related in that they are group-level measures that concentrate on opposite types of misclassification errors. The first is the False Positive Error Rate Balance Score (FPERBS) [7, 8]:

$$FPERBS = \left| \frac{FP_{S=0}}{FP_{S=0} + TN_{S=0}} - \frac{FP_{S=1}}{FP_{S=1} + TN_{S=1}} \right| \quad (17.2)$$

FPERBS measures the absolute value (ignoring the sign) of the difference in the probability that a truly negative instance is incorrectly assigned to the positive class between the protected and unprotected groups.

Analogously, the False Negative Error Rate Balance Score (FNERBS) [7, 18, 25] measures the absolute value of the difference in the probability that a truly positive instance is incorrectly assigned to the negative class between protected and unprotected groups. This score is calculated as

$$FNERBS = \left| \frac{FN_{S=0}}{FN_{S=0} + TP_{S=0}} - \frac{FN_{S=1}}{FN_{S=1} + TP_{S=1}} \right| \quad (17.3)$$

Note that a score of 0 indicates an optimally fair result for both FPERBS and FNERBS.

Finally, we will look at an individual-level measure, consistency [43], which is defined as

$$C = 1 - \frac{1}{Nk} \sum_i \sum_{j \in kNN(x_n)} |\hat{y}_i - \hat{y}_j| \quad (17.4)$$

Consistency is an individual-level similarity metric that compares the predicted class of each instance in the dataset to the class prediction of that individual's k -nearest neighbours in the dataset. If all of an individual's k neighbours have the same predicted class, then that individual is considered maximally consistent. We then repeat this for all N individuals in the dataset. A completely consistent model would have a consistency of 1 and a maximally inconsistent model would have a value of 0. As mentioned earlier, one downside of this measure is the validity of an individual's neighbours being similar to it—i.e. as the number of features increases the dataset becomes sparser in terms of the search area that contains individuals, which breaks down the concept of neighbourhood. In addition, the consistency measure is sensitive to variations in feature scaling, value ranges and distance metrics.

Note that, for group-based fairness measures in general, when the dataset has multiple sensitive features, trying to optimise a fairness measure with respect to the values of each sensitive feature separately can lead to the problem of ‘fairness gerrymandering’ [20]. In essence, this problem occurs when, although the model is fair with respect to the values of one sensitive feature (specifying coarse-grained groups of individuals), the model is unfair with respect to a logical conjunction of the values of multiple sensitive features (specifying finer-grained subgroups of individuals). For example, in a study evaluating commercial face-recognition systems, it was observed that the logical conjunction of gender and skin-type features led to a much more unfair distribution of error rates of facial recognition than the distribution for each of those two features separately [4].

It should also be noted that there is no guarantee that maximising any of the above-described fairness measures would lead to really fair predictions from a user’s perspective, since machine learning algorithms usually ignore political philosophy aspects of fairness [2]—which are out of the scope of this book chapter.

17.3 Background on Multi-objective Optimisation Approaches

In general, when analysing data about people or otherwise sensitive data, ideally a classification (supervised learning) algorithm should maximise both predictive accuracy and fairness, so this naturally leads to a multi-objective optimisation problem. Hence, not surprisingly, EAs for fair classification usually cope with multi-objective optimisation in some way. Therefore, before discussing such EAs, let us briefly review here the topic of multi-objective optimisation.

In general, methods for coping with multi-objective optimisation problems can be divided into three broad approaches [16]. The first and simplest approach is to convert a multi-objective problem into a single-objective one by using some kind of weighted-sum formula, where each objective is assigned a (user-specified) weight. In this case any standard, single-objective optimisation method can be used to optimise the weighted sum. This approach has the drawback that the weights are usually arbitrary and ad hoc. In practice, we can run the optimisation method many times with different assignments of weights to the objectives and compare the results, but this is time-consuming, tedious and inefficient, since each run using a specific weight assignment will ignore the results of other runs with different weight assignments.

The other two approaches, Pareto optimisation and lexicographic optimisation, avoid the drawback of having to specify ad hoc numerical weights for the objectives, as follows.

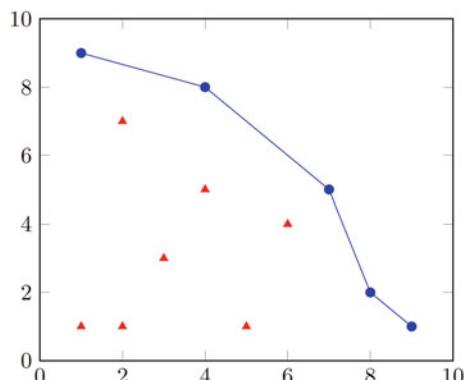
17.3.1 The Pareto Optimisation Approach

This approach aims at finding a set of non-dominated individuals (candidate solutions) based on the concept of Pareto dominance, as follows. An individual i dominates another individual j if i is better than j in at least one objective and is not worse in all other objectives. An individual i is deemed non-dominated if no other individual dominates i . At each generation of the EA, every generated individual is compared against all others in the population to check if it is a non-dominated solution.

Figure 17.1 shows an example set of candidate solutions for a two-objective problem, where each objective is represented as an axis on the graph. Assuming both objectives are to be maximised, the Pareto front is formed by the non-dominated solutions represented by blue circles, whilst the solutions represented by red triangles are dominated solutions. A very popular type of EA for multi-objective optimisation based on Pareto dominance is the NSGA (Non-dominated Sorting Genetic Algorithm) series of algorithms [12, 36].

In general, in EAs based on the Pareto approach, the solution returned by the EA is the best Pareto front found by the algorithm. This front will usually contain a large number of non-dominated solutions, representing different trade-offs between the different objectives. In real-world applications, in general, there is a need to choose just one or perhaps a few non-dominated solutions to be implemented in practice (e.g. choose a single classification model to predict the credit status of new customers). Usually, this choice is left for users, based on their subjective preference for different trade-offs of objectives.

Fig. 17.1 Example Pareto front for two objectives to be maximised. The non-dominated solutions are shown in blue circles and the dominated solutions in red triangles



17.3.2 *The Lexicographic Optimisation Approach*

Unlike the Pareto approach, the lexicographic approach to multi-objective optimisation requires that an order of priority (importance) for the objectives be defined by the user. Importantly, it is usually easier and more natural for users to specify an ordering of objectives than an ad hoc numeric value for the weight of each objective in the weighted sum approach.

The lexicographic approach works by considering each objective in turn, in their decreasing order of priority, i.e. starting with the most important objective to be optimised. For each objective, the system compares two individuals in terms of their values for that objective. If there is a ‘substantial’ difference in their objective values—i.e. if the difference is greater than a pre-defined ϵ parameter, then the best individual based on that objective is selected. However, if the difference between the objective values is within ϵ , the difference is considered ‘negligible’ and the two individuals are considered ‘equivalent’ based on that objective. In this case, the next objective in the priority order is considered in the same way, and so on. This is repeated until a substantial (greater than ϵ) difference is observed and the best individual is selected. If there is no substantial difference between two individuals for all objectives, then the individual with the best value of the first, most important, objective is selected.

Note that the use of the ϵ parameter in the lexicographic approach is in theory somewhat analogous to the use of weights assigned to objectives in the weighted-sum approach. However, in practice the ϵ parameters can be considered ‘less critical’ than the weights of objectives in the weighted-sum approach, as follows. In the weighted-sum approach, all the weights of the objectives will always affect the evaluation of the quality of all candidate solutions, whilst, in the lexicographic approach, the ϵ values of lower priority objectives will be less important, since their value is irrelevant when evaluating many candidate solutions. In particular, when a solution s_1 has a significantly better value of the highest priority objective than solution s_2 (i.e. their difference is greater than ϵ), solution s_1 is selected over s_2 regardless of the ϵ values for the lower priority objectives. In addition, in practice, since ϵ is just a ‘difference threshold’, often the same value of ϵ is used for all objectives, reducing the number of parameters.

It should also be noted that, unlike the Pareto approach (which returns many non-dominated solutions), the lexicographic approach returns a single best individual to the user.

17.3.3 *Pros and Cons of the Pareto and Lexicographic Multi-objective Optimisation Approaches*

Overall, the Pareto and lexicographic approaches have different pros and cons depending on the application, as follows.

First, the fact that the Pareto approach returns many non-dominated solutions gives users the flexibility to choose their favourite non-dominated solution, which could be an advantage in many applications. However, in applications where users are extremely busy, they might prefer to have a single solution returned by the system, as provided by the lexicographic approach.

In addition, among the three multi-objective optimisation approaches discussed here, the Pareto approach is the only one which has the advantage of not requiring any parameter to cope with the trade-offs between the different objectives. Recall that the lexicographic approach requires the specification of a parameter (threshold) ϵ to determine whether or not the difference between two values of an objective is substantial, as mentioned earlier. Although it is common to use the same value of ϵ for all objectives (particularly when all objectives take values in the same range), in some cases, a different ϵ value can be used for different objectives (e.g. if the objectives take values in different ranges), which would increase the number of parameters.

Moreover, in some applications, users clearly consider some objective(s) as more important than other(s), but the Pareto approach has the limitation that it does not allow the system to take the user's priorities about objectives into account during the search—i.e. those priorities would be considered only after the search, when the user chooses the best non-dominated solution.

For example, in the classification task, when maximising model accuracy and minimising model size, nearly all users would clearly consider accuracy more important than size. This is important background knowledge that is not considered during the search in the Pareto approach. Consider, for example, a solution which is the smallest model but has low accuracy. In the Pareto approach, this solution could be a non-dominated solution in the Pareto front for many generations, which would be an inefficient use of computational resources, since that solution would never be selected by the user as the preferred non-dominated solution after the search.

In the lexicographic approach, the EA would directly take into account the background knowledge (user preference) that model accuracy is more important than model size, so the smallest model with low accuracy would never be selected over a substantially more accurate solution (as the highest priority objective).

17.4 Evolutionary Algorithms for Fair Machine Learning

This section reviews six EAs for fair classification. The first one is a Genetic Programming (GP) method, whilst the others are in general Genetic Algorithms (GAs). In general, these EAs optimise predictive accuracy and some measure(s) of fairness. In addition, two other EAs for fair classification are mentioned in [17, 33], although these articles focus mainly on multi-objective optimisation issues, rather than focusing specifically on EAs.

17.4.1 A Genetic Programming Algorithm for Fair Feature Construction

La Cava and Moore [26] proposed a GP algorithm for feature construction that optimises predictive accuracy and fairness. This GP essentially consists of extending another GP for feature construction, called FEAT [27], to the context of fairness with multi-objective optimisation.

The individual representation is basically the same used by FEAT, where each individual is a set of trees consisting of continuous and/or boolean features in the terminal set (leaf nodes) and a number of mathematical operations in the function set (internal nodes). Hence, each tree combines original features into a new constructed feature so that a single individual represents multiple constructed features. The constructed features are then fed to a classification algorithm. For example, logistic regression was the algorithm used in [26].

The GP minimises the classification error and the unfairness of logistic regression models learned with the features constructed by the individuals. The overall goal is to minimise these two objectives across groups of instances, where each group contains instances with a given value of a sensitive (protected) feature. For example, consider a dataset with two sensitive features, Gender (with value male or female) and Age (with value old or not-old). In this case, there are four groups based on these features' values, namely, the groups containing instances where Gender = male, Gender = female, Age = old and Age = not-old.

The fitness of an individual is computed by considering such groups of instances. The goal is essentially to minimise two objectives in each group: (a) the error (loss) of a classifier (logistic regression) in the group; and (b) a measure of unfairness given by the difference of false positive (FP) rate or false negative (FN) rate between that group and the full training set of instances.

For minimising these two objectives, La Cava and Moore [26] proposed two main approaches. The first one is to use the Pareto approach, where they extend the GP (FEAT) to use the procedure for selecting non-dominated solutions in the well-known NSGA-II algorithm [12].

The second approach proposed [26] modifies the GP (again, FEAT) to use lexicase selection, which is a selection method broadly analogous to the lexicographic approach. However, instead of ordering the objectives based on user-defined priorities (which is typically the case when using the lexicographic approach), lexicase selection uses randomised lexicographic orderings of different groups of instances (with different sensitive features' values).

The basic idea of lexicase selection is that the selection procedure considers one ‘training case’ at a time, but considering different random orders of cases for each parent-selection event [19]. For each parent-selection event, it starts with a full pool of individuals (typically the entire population), and then for each training case, the pool of individuals is filtered to retain only the individuals with the smallest error in that case. The pool is often reduced to a single individual, which then becomes the selected individual. If the procedure runs out of cases and multiple individuals sur-

vived this filtering process, then the parent is selected at random among the surviving individuals.

Usually, each training instance (or example) is a ‘case’, but in this GP for constructing fair features, each group of instances, as defined above, is considered a case. Hence, for each parent-selection event, the algorithm tries to pass an individual through all groups, but an individual is allowed to pass through one of these groups only if its fitness is within a threshold ϵ (epsilon) of the best (using the ϵ -lexicase variant of lexicase selection). Note that a random sequence of groups effectively represents a logical conjunction of sensitive features’ values (e.g. Gender = female and Age = old). Hence, the GP has to evolve features that lead to accurate and fair classifiers for subgroups defined by intersections of the original groups.

Lexicase selection promotes diversity in the selected parents, because it can select individuals that perform very well on a small number of ‘hard training cases’ (in this context, hard-to-classify (sub)groups of instances) even if those individuals do not perform well on most of the training cases. This means this method can select ‘specialist’ individuals, contributing to the diversity of the selected parents [19, 26].

In addition, the fact that the GP considers a logical conjunction of sensitive features’ values helps to mitigate the problem of ‘fairness gerrymandering’ [20], mentioned around the end of Sect. 17.2.

The experiments reported in [26] used four datasets often used as benchmarks in fair-classification research, and compared the results of six different selection or survival methods for the GP, including random search. The results were evaluated using the well-known hyper-volume measure, which estimates the area of the objective space covered by the solutions in a Pareto front. Overall, the GP variants performed better than a baseline approach for improving fairness [21]. However, surprisingly, the overall best rankings were obtained by random search, followed by the two versions of lexicase selection—with and without fairness as an objective (with no significant difference between the results of these two versions). The fact that the GP variants did not outperform random search seems to be due to the GPs overfitting the training data.

17.4.2 Optimising the Hyper-parameters of a Fair Classifier with a Genetic Algorithm

Valdivia et al. [39] proposed a fairness-aware multi-objective GA for optimising the hyper-parameters of a classification algorithm, in order to produce a set of non-dominated classification models that trade-off accuracy and fairness performance. The GA is again based on the popular NSGA-II algorithm [12] for multi-objective optimisation.

The authors referred to the hyper-parameter optimisation process as a meta-learning approach. However, since that term is very broad, in this review, we prefer to describe this process as a type of Automated Machine Learning (Auto-ML).

Auto-ML systems aim at automatically selecting the best algorithm and/or the best hyper-parameter settings of an algorithm to a given input dataset [46]. Hence, the task of automatically optimising the hyper-parameter settings of a given classification algorithm, performed by this GA, matches well the goal of Auto-ML.

One advantage of this Auto-ML approach is that any generic classification algorithm, even those that are fairness unaware, can be used as the base classifier. Each classifier only requires its own hyper-parameter coding scheme and initialisation procedure to be defined in the Auto-ML framework before it can be used.

The GA uses two genetic operators, as follows. The crossover operator is a variant of uniform crossover operating on each gene. For each gene $g_{k,i}$ —where k is the iteration and i refers to a specific individual—one of two outcomes is realised, based on the probability p_c : either the values of the parents, $g_{k-1,a}$ and $g_{k-1,b}$, are simply copied from the parent to the children or the values from the parents are combined using the following equation:

$$\begin{aligned} g_{k,j} &= \frac{g_{k-1,a} + g_{k-1,b}}{2} + \beta \frac{|g_{k-1,a} - g_{k-1,b}|}{2} \\ g_{k,j+1} &= \frac{g_{k-1,a} + g_{k-1,b}}{2} - \beta \frac{|g_{k-1,a} - g_{k-1,b}|}{2} \end{aligned} \quad (17.5)$$

This pair of equations creates two individuals that have their parents' average gene values perturbed by some proportional value of the difference between the parents' gene values, where β is a uniformly distributed random value between 0 and 1.

The second genetic operator is a mutation procedure that modifies the hyper-parameter values encoded in an individual's genes; and an individual undergoes mutation based on the user-defined probability p_m . When mutation is performed, a random gene is selected and mutated based on the following equations:

$$g_{k,j} = \begin{cases} g_{k,j} + \delta(g_{k,j} - \min(h_i)), & u' < 0.5 \\ g_{k,j} + \delta(\max(h_i) - g_{k,j}), & u' \geq 0.5 \end{cases} \quad (17.6)$$

and

$$\delta = \begin{cases} -1 + 2u''^{\frac{1}{\mu+1}}, & u'' \leq 0.5 \\ 1 - 2(1 - u'')^{\frac{1}{\mu+1}}, & u'' > 0.5 \end{cases} \quad (17.7)$$

where $\min(h_i)$ and $\max(h_i)$ are the allowed ranges for the i -th hyper-parameter, u' and u'' are uniformly distributed random numbers and μ is a user-defined mutation parameter.

The GA optimises a single measure of fairness alongside the geometric mean of sensitivity (the true positive rate) and specificity (the true negative rate). The fairness measure used is the FPERBS shown in Eq. 17.2.

Valdivia et al. [39] have tested the hyper-parameter optimisation framework using two base classifiers: a logistic regression classifier and a decision tree classifier, on five datasets. The analysis of the Pareto front showed that with the base decision

tree classifier modest reductions in predictive performance allowed large gains in the fairness metric. However, the logistic regression base learner required significantly higher losses in predictive performance as a trade-off for improvements to the fairness measure.

When investigating the complexity of the decision trees, it was found that increased levels of fairness also led to an increase in the number of leaf nodes in the models. This allows the model to split the data into increasingly finer divisions to equalise the false positive rates between the protected and unprotected groups. Note that more complex models are less interpretable, which may conflict with the goal of making machine learning algorithms fairer.

The experiments reported in [39] have not compared the GA against any other Auto-MIL method for hyper-parameter optimisation of classification algorithms, which would be an interesting comparison for further validating the GA's performance.

17.4.3 A Lexicographic Optimisation-Based Genetic Algorithm for Fair Feature Selection (LGAFFS)

LGAFFS [3] is a lexicographic optimisation-based GA that performs feature selection in a pre-processing phase, i.e. before learning the final classification model. It aims to select a subset of features that lead to the construction of a fairer model by the base classification algorithm whilst retaining predictive performance.

Unlike most EAs in this chapter, which perform multi-objective optimisation using the Pareto dominance approach, LGAFFS selects individuals for reproduction based on the principle of lexicographic optimisation to combine predictive accuracy and fairness measures.

Each individual in the population represents a candidate feature subset. More precisely, each individual consists of a string of N bits (genes), where N is the number of features in the dataset, and the i -th gene takes the value 1 or 0 to indicate whether or not (respectively) the i -th feature is selected.

LGAFFS follows a wrapper approach to feature selection [29], where a base classification algorithm is used to learn a classification model based on the feature subset selected by an individual, and that model's quality (in terms of accuracy and fairness) is used to compute that individual's fitness. Hence, the GA aims at finding the best subset of features for the base classification algorithm. Fitness computation is performed using an internal cross-validation routine that uses only the training set (i.e. it does not use the test set).

LGAFFS uses two standard genetic operators, uniform crossover and bit-flip mutation to create new individuals in each generation. However, it uses a non-standard, ramped population initialisation procedure. In this procedure, each i -th member of the initial population has a different probability p_i of having a feature turned on (activated). These probability values are bounded by two user-specified

parameters, `MIN_P` and `MAX_P`. This procedure creates a more diverse initial population, where different individuals can have very different numbers of activated (selected) features. By contrast, if a standard population initialisation was used, all individuals would have the same expected number of activated features, which would be normally distributed around pN , where N is the number of features and p would be the probability of activating a feature if that was kept fixed for all individuals.

LGAFFS uses a lexicographic tournament selection procedure, with a tournament size of two, in order to select individuals for undergoing crossover and mutation. This lexicographic tournament selection procedure is a direct application of the lexicographic optimisation approach (described in Sect. 17.3.2) to the two individuals competing in the tournament.

As mentioned earlier, the lexicographic approach requires the objectives to be ordered in decreasing order of priority. LGAFFS optimises two objectives. The first, highest priority objective is a measure of predictive accuracy, namely, the geometric mean of sensitivity and specificity ($GM_{Sen \times Spec}$). The second, lower priority objective is an aggregated value of the four fairness measures defined in Sect. 17.2.

It should be noted that there is no consensus in the literature about what is the best fairness measure, and so it would be ‘unfair’ to prioritise one fairness measure over the others. Hence, the need to aggregate the fairness measures into a single objective without preference to a particular measure. This aggregation is achieved by computing all possible 24 (4!) permutations of the four fairness measures and comparing individuals across all permutations. Each of the 24 possible permutation defines a lexicographic order of measures that can be evaluated to find the first substantial difference of fairness (i.e. a difference greater than ϵ) between the two individuals in the tournament, at which point the best individual is given a win. Once all permutations of fairness measures have been evaluated, the individual with the higher number of wins is declared the best individual overall (i.e. the tournament winner) as long as this difference is greater than some ϵ for the number of wins—a user-defined parameter.

Experiments reported in [3] compared LGAFFS against two other approaches on seven datasets, using random forest as the classifier. The results can be summarised as follows. First, in the comparison against the baseline approach of not performing feature selection in a pre-processing step, overall LGAFFS obtained similar predictive accuracies but better values for three of the four measures of fairness. Second, in the comparison against a local search method that optimises accuracy only, overall LGAFFS obtained slight better accuracies and substantially better values of all the four measures of fairness.

17.4.3.1 Experimental Results Comparing LGAFFS to a Weighted-Sum GA

In these experiments, we compare LGAFFS to a simpler GA using the weighted-sum approach for multi-objective optimisation, hereafter called Weighted GA (WGA) for short. WGA uses a simple weighted formula that allows the user to alter the search

bias between accuracy and the average value of four fairness measures (the same four measures used by LGAFFS).

The comparison of LGAFFS to WGA is a controlled experiment, as both GAs use the same population-initialisation procedure, genetic operators, operator probabilities, population size and number of generations. They also use the same predictive accuracy and fairness measures for fitness computation including the same aggregation of four fairness measures into a single objective. In addition, both GAs follow a wrapper approach for feature selection using the random forest algorithm as the base classifier.

On the other hand, LGAFFS and the WGA vary in their multi-objective optimisation approach. Hence, the observed differences in performance between the two algorithms (reported below) reflect mainly the differences in the effectiveness of the lexicographic and weighted-sum approaches.

The parameters required by these two algorithms can be divided into three groups: (a) parameters specific to LGAFFS; (b) parameters specific to WGA; and (c) parameters used by both algorithms—which were set to the same values for both algorithms, to make their comparison as fair as possible. The parameter values in each group are as follows:

- LGAFFS-specific lexicographic parameters: `accuracy_ε` and `fairness_ε`: 0.01, `fair_rank_ε` and `fair_test_ε`: 1 (for details of these parameters, see [3])
- WGA-specific parameters: weights for the accuracy and fairness objectives in the weighted-sum fitness function: 0.5
- GA-related parameters, shared by LGAFFS and WGA: `population_size`: 100, `MAX_P`: 0.5, `MIN_P`: 0.1, number of folds for internal cross-validation: 3, `max_iterations`: 50, `tournament_size`: 2, `crossover_probability`: 0.9, `mutation_probability`: 0.05 (again, see [3] for details)

Table 17.1 mentions, for each of the seven datasets used in all experiments, its number of instances and features, as well as the features used as sensitive features in the experiments. Recall that a sensitive feature represents a protected characteristic or a group that is unfairly treated. When a dataset has multiple sensitive features, the algorithm is run multiple times using a different sensitive feature each time. All these are binary classification datasets. The first six datasets are from the UCI Machine Learning repository [13]; and the ProPublica dataset involves bias in predictions of which criminals would re-offend [1].

Recall that the highest priority objective for LGAFFS is the predictive accuracy measure ($GM_{Sen \times Spec}$). The experimental results of the comparison between LGAFFS and the WGA with respect to this measure are found in Table 17.2. In each row of this table (i.e. for each pair of a dataset and a sensitive feature), the best result is shown in boldface. The last but one row of the table shows the average rank obtained by each approach (the lower the rank, the better the result), whilst the last row shows the p-value obtained by the Wilcoxon signed-rank statistical significance test. We can see that the average rank of both algorithms is similar and no statistical significance is found between the two algorithms (at the usual significance level of

Table 17.1 Datasets used in all experiments, detailing the number of instances, features and the sensitive features for each dataset

Dataset	Instances	Features	Sensitive features
Adult income (US Census)	48842	14	Race, Gender, Age
German credit	1000	20	Age, Gender
Credit card default	30000	24	Gender
Communities and crime	1994	128	Race
Student (Portuguese)	650	30	Age, Gender, Relationship
Student (Maths)	396	30	Age, Gender, Relationship
ProPublica recidivism	6167	52	Race, Gender

$\alpha = 0.05$), with LGAFFS performing better in 11 classification problems (combinations of dataset and sensitive feature) and WGA performing better in 10 classification problems.

However, regarding the secondary objective of fairness (aggregating four fairness measures), the two algorithms have substantially different performances, as can be observed in Table 17.3. It should be noted that three fairness measures (DS, FPERBS and FNERBS) have been slightly modified from the equations shown in Sect. 17.2, which are to be minimised, by computing instead 1 minus the value in each of those equations (e.g. DS was re-defined as 1 minus the DS formula shown in Eq. 17.1), so that all objectives are now to be maximised. Statistically significant results, at the usual significance level of $\alpha = 0.05$, are marked with a red triangle in the table's last row. These results show that LGAFFS significantly outperformed WGA in three of the four fairness measures: discrimination score, FPERBS and FNERBS, which are all measures of fairness at the group level (i.e. for a group of individuals in the datasets). By contrast, there is no significance and very similar average rank for the consistency measure, which is the only measure of fairness at the individual level (i.e. at the level of a data instance) in these experiments.

17.4.4 A Pareto Dominance-Based Genetic Algorithm for Fair Feature Selection

Rehman et al. have proposed another GA for fair feature selection [34], again in the context of the classification task of machine learning. This GA has both similarities and differences with respect to LGAFFS [3] (described in the previous Subsection), as follows.

First of all, in both this GA and LGAFFS, each individual represents a candidate feature subset and both GAs use the same individual representation: each individual

Table 17.2 Predictive accuracy-based results comparing the $GM_{Sen \times Spec}$ (Geometric Mean of Sensitivity and Specificity) of LGAFFS to a weighted GA (WGA) using a Random Forest base classifier across all dataset/sensitive feature combinations. The average rank of each algorithm is shown in the penultimate row of the table, followed by the Wilcoxon signed-rank test P value

Dataset	Sensitive attribute	$GM_{Sen \times Spec}$	
		LGAFFS	WGA
Adult	Age	0.6475	0.7511
Adult	Race	0.7409	0.7511
Adult	Sex	0.7420	0.7530
German credit	Age	0.5901	0.5612
German credit	Gender	0.6036	0.5776
Student maths	Age	0.9208	0.8825
Student maths	Dalc	0.8951	0.9006
Student maths	Famrel	0.9052	0.8971
Student maths	Romantic	0.8984	0.8955
Student maths	Sex	0.9027	0.8970
Student maths	Walc	0.9000	0.9037
Student Portuguese	Age	0.8196	0.7898
Student Portuguese	Dalc	0.7867	0.7785
Student Portuguese	Famrel	0.8031	0.8101
Student Portuguese	Romantic	0.7846	0.7944
Student Portuguese	Sex	0.8110	0.8206
Student Portuguese	Walc	0.8035	0.8158
Communities and crime	Race	0.8303	0.8422
Default of credit	Sex	0.5896	0.4866
Propublica recidivism	Race	0.7306	0.6883
Propublica recidivism	Sex	0.7113	0.5929
Average rank		1.4762	1.5238
Wilcoxon signed-rank test		0.4354	

consists of m bits, where m is the number of features (attributes) in the dataset, and each bit takes the 1 or 0 value to indicate whether or not its corresponding feature is selected, respectively.

Another similarity between Rehman et al.'s GA and LGAFFS is that, at a high level of abstraction, both these GAs follow the wrapper approach for feature selection in a pre-processing phase, so that the selected features will be used later as input by a given classification algorithm (called here the 'target algorithm'). Hence, following the principle of their wrapper approach, both GAs compute an individual's fitness by measuring the quality of a model learned by the target algorithm when it is trained with the candidate feature subset encoded into that individual. As a result, the selected features are customised to the target algorithm. To be precise, the target classification

Table 17.3 Experimental results comparing LGAFFS to a weighted GA (WGA) for the four fairness measures being used by both algorithms. Random forest is used as the base classifier. Statistically significant results are signified by a red triangle

Dataset	Sensitive Attribute	Discrimination Score		Consistency		FPERBS		FNERBS	
		LGAFFS	WGA	LGAFFS	WGA	LGAFFS	WGA	LGAFFS	WGA
Adult	Age	0.8485	0.7554	0.8656	0.7937	0.9522	0.9008	0.9189	0.6204
	Race	0.9356	0.8923	0.8201	0.8005	0.9862	0.9541	0.9902	0.8946
Adult	Sex	0.8498	0.8232	0.8163	0.7952	0.9490	0.9217	0.9416	0.9274
	Age	0.9361	0.8545	0.7642	0.7872	0.8633	0.7874	0.8911	0.9066
German credit	Gender	0.9399	0.9291	0.7510	0.7302	0.8993	0.8584	0.9230	0.9457
	Age	0.7964	0.8314	0.8444	0.8326	0.9022	0.8753	0.8951	0.8652
Student Maths	Dalc	0.7563	0.7825	0.8377	0.8401	0.8051	0.8182	0.8157	0.8141
	Famrel	0.7039	0.7041	0.8361	0.8391	0.8760	0.8600	0.9222	0.9022
Student Maths	Romantic	0.8840	0.8468	0.8271	0.9151	0.8998	0.9106	0.9101	
	Sex	0.8397	0.8426	0.8387	0.8341	0.7646	0.8064	0.9012	0.9111
Student Maths	Walc	0.8364	0.7780	0.8376	0.8451	0.8206	0.7865	0.9196	0.8892
	Student Portuguese	0.8638	0.8604	0.9106	0.9205	0.7038	0.6481	0.9578	0.9511
Student Portuguese	Dalc	0.8470	0.8408	0.9128	0.9211	0.6230	0.5719	0.9088	0.9293
	Student Portuguese	Walc	0.8019	0.8206	0.9097	0.9146	0.6450	0.6508	0.9205
Student Portuguese	Romantic	0.9370	0.9364	0.9211	0.9199	0.7608	0.7608	0.9686	0.9710
	Student Portuguese	Sex	0.9200	0.9093	0.9134	0.9128	0.7646	0.6955	0.9708
Student Portuguese	Walc	0.9271	0.9215	0.9186	0.9196	0.7214	0.7343	0.9679	0.9587
	Communities and crime	0.6623	0.5902	0.6056	0.6038	0.9182	0.8428	0.8313	0.7813
Default of credit	Sex	0.9755	0.9729	0.8354	0.8695	0.9739	0.9688	0.9852	0.9826
	Race	0.8324	0.7714	0.6849	0.6788	0.9022	0.8078	0.9105	0.8469
Propublica recidivism	Sex	0.9408	0.9659	0.6756	0.6623	0.9254	0.9326	0.9451	0.9411
	Average rank	1.2857	1.7143	1.4762	1.5238	1.2381	1.7143	1.2381	1.7619
								0.0036	
								0.0030	
								0.0036	
								0.0030	
								0.0036	
								0.0030	
								0.0036	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	
								0.0030	

algorithm was random forests in the experiments with LGAFFS in [3], whilst the experiments with Rehman et al.'s GA in [34] used three target algorithms: logistic regression, XGBoost and a Support Vector Machine (SVM).

However, Rehman et al.'s GA and LGAFFS also have important differences in their fitness computation, particularly with respect to which objective functions they optimise and the approach used to perform multi-objective optimisation, as follows.

First, Rehman et al.'s GA optimises two objectives: the F1-score, as a measure of predictive accuracy, and statistical parity (or discrimination score) as a measure of fairness. To be precise, we are referring here to the more sophisticated version of that GA, which is referred to as 'Algorithm 2' in [34], rather than the simpler version of that GA, referred to as 'Algorithm 1', which optimises just the fairness objective, ignoring predictive accuracy. By contrast, as mentioned earlier, LGAFFS optimises the geometric mean of sensitivity and specificity as the predictive accuracy measure, and optimises four fairness measures (including the discrimination score).

Second, the (Algorithm 2) GA in [34] is basically the well-known NSGA-II for multi-objective optimisation based on the concept of Pareto dominance. By contrast, as mentioned earlier, LGAFFS follows a lexicographic approach [3], rather than the Pareto dominance approach.

Two experiments were reported in [34], each using only two datasets. Each run of the GA selected features for one of the three aforementioned classification algorithms. In the first experiment, the GA was used to optimise fairness only (i.e. not accuracy). As expected, in general, the features selected by the GA led to improved fairness but reduced predictive accuracy, by comparison with not performing feature selection in a pre-processing step (i.e. giving all features to the classification algorithm). The second experiment used NSGA-II to optimise both accuracy and fairness, but there was no comparison with other feature selection approaches.

In the future, it would be interesting to compare the results of the GAs for fair feature selection proposed in [3, 34] on controlled experiments with the same target classification algorithm (for the wrapper approach) and the same objective functions on the same datasets, since they use basically the same individual representation but quite different multi-objective optimisation approaches.

17.4.5 An Evolutionary Algorithm for Fair Ensembles of Artificial Neural Network Models

Zhang et al. [45] proposed a fairness-aware multi-objective GA for evolving an ensemble of Artificial Neural Network (ANN) models. The individual representation consists of a vector of real-valued weights and biases for an ANN, using the same fixed ANN architecture (a fully connected network with one hidden layer) for all individuals.

In addition, the models produced by the individuals in the last generation are combined into an ensemble of non-dominated models. This work considered four

different approaches to produce the ensemble, including the selection of all non-dominated solutions and different approaches to select a subset of non-dominated solutions, based on their quality or their diversity. The use of an ensemble of ANN models leads to good predictive accuracy in general, but it sacrifices the interpretability, due to both the black-box nature of ANNs and the aggregation of multiple models in the ensemble. Hence, in applications where the interpretability of the learned models is also important, it would be interesting to use some post hoc interpretability approach [5] to try to explain the predictions of the ANN ensemble.

Regarding selection, the EA proposed in [45] does not use Pareto dominance-based selection (like in the well-known NSGA-II algorithm) because this approach tends to be inefficient when the problem has many (more than 3) objectives, since the proportion of non-domination solutions tends to be quite large in this case. In some of the experiments in this work, there are nine objectives being optimised (see below), so Pareto dominance-based selection would not be practical. Hence, the authors used instead a selection method more suitable for many-objective optimisation problems, based on the stochastic ranking algorithm described in [28].

The empirical results reported in [45] involve two sets of experiments. In the first set, the EA optimised three objectives, namely, one measure of predictive accuracy (the cross-entropy) and two measures of fairness—where one is a group-fairness measure and the other is an individual-fairness measure. The results of the first set of experiments were evaluated in terms of the well-known hyper-volume measure and the coverage over Pareto front (which emphasises more the diversity of solutions in the Pareto front). The results of the multi-objective EA were compared with the results of four variants of Multi-FR [42], a non-evolutionary method for multi-objective optimisation based on gradient descent, on 8 datasets. Broadly speaking, the solutions generated by the multi-objective EA dominated (in the Pareto sense) the solutions generated by the multi-FR method more often than vice versa, showing the greater effectiveness of the multi-objective EA.

In the second set of experiments, the EA optimised nine objectives, the cross-entropy and eight fairness measures. The evaluation of the results actually considered 16 fairness measures, but they were divided into 2 groups: the fairness measure set 1, containing 8 measures optimised by the EA (chosen as representative measures of the full set of 16 measures), and the fairness measure set 2, containing the other 8 measures, which were not optimised by the EA and were used only to evaluate if the EA results also generalise well to improve the values of fairness measures not optimised during training. This second set of experiments used 15 datasets. The four variants of the proposed EA (with four types of ensembles) were compared against five variants of non-evolutionary ensembles proposed in [22], which used various types of base classifiers as members of the ensembles. Hence, due to the use of a relatively large number of datasets and fairness measures, as well as several variants of evolutionary and non-evolutionary multi-objective ensemble methods being compared, this seems the most extensive evaluation of a fairness-aware EA in the current literature. The results of these experiments have shown that, for most datasets, the values of the fairness measures in set 2 (not optimised during training)

are also improved in general by the EA, as evaluated by the hyper-volume measure, showing good generalisation ability regarding the fairness measures.

17.4.6 A Genetic Algorithm for Finding Fair Counterfactuals

Dandl et al. [10] proposed a GA for generating fair counterfactuals, which they called ‘counterfactual fairness’. To explain this concept, first, let us briefly introduce the concept of counterfactual, which is a statement of how the world would have to be different in order for a desirable outcome to occur [9].

To make this clearer, let us use as an example the typical application domain of credit approval. Suppose a customer’s credit card application is rejected by a bank. In this scenario, a counterfactual should identify how the values of some features in the customer’s application would have to change in order for the customer to receive the desirable outcome of credit approval. An example counterfactual would be: ‘Your current annual salary is \$40,000; if your annual salary were \$60,000 your credit card would have been approved’.

Turning to the concept of counterfactual fairness in [10], in the context of supervised learning (with a target variable), the key idea is that the value of the target variable predicted by a model learned from the data for a given individual should be the same as the value that would be predicted if that individual had a different value of a sensitive (or protected) feature, which typically would mean the individual would belong to a different demographic group. For example, consider again the credit approval domain, and suppose the sensitive feature is gender. Consider a male customer whose credit-card application was approved based on a feature vector (set of variables) describing that customer. Then, counterfactual fairness requires that if a customer had the same feature vector with the exception of having gender female, the credit card application would also be approved. That is, the approval outcome should not be affected by the change only in the value of a sensitive feature.

The GA for counterfactual fairness in [10] is based on the GA for counterfactual explanations in [9], and its main characteristics are as follows.

The individual representation essentially consists of a feature vector representing a candidate counterfactual for a given input data instance. Note that a candidate counterfactual refers to the same features as the data instances; the difference is that some features in the counterfactual will take values that differ from their values in the feature vector in the input data instance. Hence, an individual (candidate counterfactual) will contain both numerical and categorical features, if the dataset consists of both these data types of features.

In the search for fair counterfactuals for a given input data instance x , the GA optimises three objectives: (a) validity, in the sense that the counterfactual should have a high probability with respect to the distribution of the training instances in the protected (or sensitive) group; (b) similarity to the current instance x , i.e. the counterfactual should have feature values that are similar to the feature values of x ;

and (c) plausibility, i.e. the counterfactual should lie in a high-density region of the data space for the full dataset.

To perform this multi-objective optimisation, a modified version of the well-known NSGA-II algorithm [12] is used to return a set of non-dominated solutions (counterfactuals), based on the concept of Pareto dominance. The NSGA-II's modifications were designed to make it more tailored for the task of finding fair counterfactuals. In particular, two types of NSGA-II modifications suggested in [10] are as follows.

First, instead of randomly generating the initial population as usual, the population can be initialised with training data instances that have the protected value of the sensitive feature.

Second, specialised mutation operators can be used that constraint the allowable changes to feature values, possibly based on background knowledge. For example, since an individual can hardly change her/his gender, this feature can have its value frozen, i.e. not being allowed to undergo mutation.

The experiments reported in [10] used two datasets to evaluate the counterfactuals generated by the GA: a synthetic dataset, created using the data-generating process of a law school dataset [24], and the Adult (census) dataset from the well-known UCI Machine Learning repository—often used as a benchmark in fair classification. The counterfactuals generated by the GA were compared against two baselines for generating counterfactuals: at random and using a nearest neighbour approach. The results showed that overall the GA generated good counterfactuals, but the use of only two datasets (only one being a real-world one) makes it difficult to draw a robust conclusion about the GA's performance.

17.5 Discussion and Conclusions

We have reviewed several EAs developed for ‘fair classification’, i.e. EAs that aim to evolve candidate solutions for a classification problem which exhibit both high predictive accuracy and a high degree of fairness. This is naturally cast as a multi-objective optimisation problem, involving a trade-off between the objectives of optimising accuracy and fairness. Hence, it is not surprising that all the EAs reviewed in this chapter use some multi-objective optimisation approach for addressing this problem.

The six EAs reviewed in this chapter address in general different types of tasks within the broader process of classification, reflecting the fact that EAs are very flexible and generic search and optimisation methods. Three of the six EAs addressed a data pre-processing method for classification: one GP method for feature construction, and two (independently developed) GAs for feature selection. Another reviewed method was a GA for optimising the hyper-parameters of a base classification algorithm; and another method was an EA to evolve an ensemble of ANN models. Finally, another reviewed method was a GA for finding fair counterfactuals.

In general, these EAs work together with a given classification algorithm to optimise the classification model to be learned from the data by that algorithm, i.e. the EAs find candidate solutions customised not only to the input dataset but also to the chosen algorithm—which could be in principle any generic classification algorithm. The exception is the reviewed GA for finding fair counterfactuals, which does not require any classification algorithm, since it simply evolves counterfactuals, without learning classification models.

Regarding multi-objective optimisation approaches, most of the reviewed EAs use the Pareto dominance approach, in general, using some variant of the very popular NSGA-II algorithm based on that approach. Exceptions are an EA using stochastic ranking selection for many-objective optimisation; a GA for feature selection based on the lexicographic approach, where objectives are optimised in decreasing order of priority; and a version of a GP using lexicase selection (broadly related to the lexicographic approach).

As discussed earlier, the Pareto and lexicographic approaches have different pros and cons. In the context of the problem of evolving accurate and fair classifiers, in the case of real-world applications, arguably the choice between these approaches should depend mainly on the needs and preferences of users. In particular, if the user does not want to specify a priority order for the objectives of accuracy and fairness, and if the user would like to examine many non-dominated solutions in order to subjectively choose the one with their favourite trade-off between those objectives, then the Pareto approach seems a natural choice. On the other hand, if the user clearly assigns different priorities to the different objectives (e.g. considering accuracy more important than fairness), and if the user would find it difficult to choose among many non-dominated solutions, it seems natural to incorporate the user's priority order of objectives into the search algorithm, to make the search more efficient, and in this case, the lexicographic approach seems a natural choice.

In any case, the current literature lacks an empirical comparison between the Pareto and lexicographic approaches for fair classification, so this is a natural direction for future research.

In addition, there is a clear need for further work on evaluating EAs for fair classification on more datasets, since the number of datasets in nearly all the experiments evaluating the reviewed EAs varied from only 2–7. The exception is that the experiments with 8 or 15 datasets were reported in [45]. Experiments with many datasets are needed to draw more robust conclusions about the EAs' performances. Moreover, the reviewed experiments focused on measuring fairness using mathematical formulas only, but recall that (as mentioned in Sect. 17.2) there is no guarantee that maximising such fairness measures would lead to really fair predictions from a user's perspective, given the subjective nature of fairness. For real-world applications, it is important that real-world users evaluate the fairness of the learned classification models.

Another direction for future research would be to consider more objectives, including different types of objectives, like some measure of interpretability or simplicity of the classifiers—an issue that has also become more popular in machine learning in recent years [5]. Actually, in many applications involving data about people, both fairness and interpretability are needed in order to produce classifiers that can be really trusted by users and deployed in their real-world business activities.

References

1. Angwin, J., Larson, J., Mattu, S., Kirchner, L.: Machine bias: there's software used across the country to predict future criminals, and it's biased against blacks (2016)
2. Binns, R.: Fairness in machine learning: lessons from political philosophy. *J. Mach. Learn. Res.* **81**, 1–11 (2018)
3. Brookhouse, J., Freitas, A.A.: Fair feature selection with a lexicographic multi-objective genetic algorithm. In: Proceedings of the 2022 Parallel Problem Solving from Nature Conference (PPSN 2022), LNCS 13399, pp. 151–163. Springer (2022)
4. Buolamwini, J., Gebru, T.: Gender shades: intersectional accuracy disparities in commercial gender classification. In: Proceedings of Machine Learning Research: Conference on Fairness, Accountability and Transparency, vol. 81, pp. 1–15 (2018)
5. Burkart, N., Huber, M.F.: A survey on the explainability of supervised machine learning. *J. Mach. Learn. Res.* **70**, 245–317 (2021)
6. Calders, T., Verwer, S.: Three naive bayes approaches for discrimination-free classification. *Data Min. Knowl. Disc.* **21**(2), 277–292 (2010)
7. Chouldechova, A.: Fair prediction with disparate impact: a study of bias in recidivism prediction instruments. *Big Data* **5**(2), 153–163 (2017)
8. Corbett-Davies, S., Goel, S.: The measure and mismeasure of fairness: a critical review of fair machine learning (2018). [arXiv:1808.00023](https://arxiv.org/abs/1808.00023)
9. Dandl, S., Molnar, C., Binder, M., Bischl, B.: Multi-objective counterfactual explanations. In: Proceedings of PPSN 2020 (Parallel Problem Solving from Nature Conference), LNCS 12269, pp. 448–469. Springer (2020)
10. Dandl, S., Pfisterer, F., Bischl, B.: Multi-objective counterfactual fairness. In: Proceedings of the GECCO'22 Companion (Genetic and Evolutionary Computation Conference), pp. 328–331. ACM Press (2022)
11. Deb, K.: Multi-Objective Optimization Using Evolutionary Algorithms. Wiley (2002)
12. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Trans. Evol. Comput.* **6**(2), 182–197 (2002)
13. Dua, D., Graff, C.: UCI machine learning repository (2017)
14. Fernandez-Delgado, M., Cernadas, E., Barro, S., Amorin, D.: Do we need hundreds of classifiers to solve real-world classification problems? *J. Mach. Learn. Res.* **15**, 3133–3181 (2014)
15. Freitas, A.A.: Data Mining and Knowledge Discovery with Evolutionary Algorithms. Springer (2002)
16. Freitas, A.A.: A critical review of multi-objective optimization in data mining: a position paper. *ACM SIGKDD Explor. Newsl.* **6**(2), 77–86 (2004)
17. Haas, C.: The price of fairness—a framework to explore trade-offs in algorithmic fairness. In: Proceedings of the 40th International Conference on Information Systems (ICIS) (2019)
18. Hardt, M., Price, E., Srebro, N.: Equality of opportunity in supervised learning. In: Advances in Neural Information Processing Systems, pp. 3315–3323 (2016)
19. Helmuth, T., Pantridge, E., Spector, L.: Lexicase selection of specialists. In: Proceedings Genetic and Evolutionary Computation Conference (GECCO-2019), pp. 1030–1038. ACM Press (2019)

20. Kearns, M., Neel, S., Roth, A., Wu, Z.S.: An empirical study of rich subgroup fairness for machine learning. In: Proceedings of the 2019 ACM Conference on Fairness, Accountability and Transparency (FAT), pp. 100–109. ACM (2019)
21. Kearns, M., Neel, S., Roth, A., Wu, Z.S.: An empirical study of rich subgroup fairness for machine learning. In: Proceedings of the Conference on Fairness, Accountability, and Transparency, pp. 100–109 (2019)
22. Kenfack, P.J., Khan, A.M., Kazmi, S.A., Hussain, R., Oracevic, A., Khattak, A.M.: Impact of model ensemble on the fairness of classifiers in machine learning. In: Proceedings of the 2021 International Conference on Applied Artificial Intelligence (ICAPAI), pp. 1–6. IEEE (2021)
23. Kleinberg, J., Mullainathan, S., Raghavan, M.: Inherent trade-offs in the fair determination of risk scores (2016). [arXiv:1609.05807](https://arxiv.org/abs/1609.05807)
24. Kusner, M.J., Loftus, J., Russell, C., Silva, R.: Counterfactual fairness. In: Advances on Neural Information Processing Systems, vol. 30 (2017)
25. Kusner, M.J., Loftus, J., Russell, C., Silva, R.: Counterfactual fairness. In: Advances in Neural Information Processing Systems, pp. 4066–4076 (2017)
26. La Cava, W., Moore, J.H.: Genetic programming approaches to learning fair classifiers. In: Proceedings of Genetic and Evolutionary Computation Conference (GECCO-2020), pp. 967–975 (2020)
27. La Cava, W., Singh, T.R., Taggart, J.P., Suri, S., Moore, J.H.: Learning concise representations for regression by evolving networks of trees. In: Proceedings of the International Conference on Learning Representations (ICLR 2019), 25 Mar 2019, p. 16 (2019). [arXiv:1807.00981v3 \[cs.NE\]](https://arxiv.org/abs/1807.00981v3)
28. Li, B., Tang, K., Li, J., Yao, X.: Stochastic ranking algorithm for many-objective optimization based on multiple indicators. *IEEE Trans. Evol. Comput.* **20**(6), 924–938 (2016)
29. Li, J., Cheng, K., Wang, S., Morstatter, F., Trevino, R.P., Tang, J., Liu, H.: Feature selection: a data perspective. *ACM Comput. Surv.* **50**(6), 94:1–94:45 (2017)
30. Lipton, Z., McAuley, J., Chouldechova, A.: Does mitigating ml’s impact disparity require treatment disparity? *Advances in Neural Information Processing Systems*, vol. 31 (2018)
31. Mehrabi, N., Morstatter, F., Saxena, N., Lerman, K., Galstyan, A.: A survey on bias and fairness in machine learning (2019). [arXiv:1908.09635](https://arxiv.org/abs/1908.09635)
32. Pleiss, G., Raghavan, M., Wu, F., Kleinberg, J., Weinberger, K.Q.: On fairness and calibration. In: Proceedings of 31st Conference on Neural Information Processing Systems (NIPS 2017) (2017)
33. Quadrianto, N., Sharmanska, V.: Recycling privileged learning and distributed matching for fairness. In: Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017), pp. 677–688 (2017)
34. Rehman, A.U., Nadeem, A., Malik, M.Z.: Fair feature subset selection using multiobjective genetic algorithm. In: Proceedings of the GECCO-22 Companion (Genetic and Evolutionary Computation Conference), pp. 360–363. ACM Press (2022)
35. Skeem, J.L., Lowenkamp, C.T.: Risk, race, & recidivism: predictive bias and disparate impact. *Criminology* **54**, 680 (2016)
36. Srinivas, N., Deb, K.: Multiobjective optimization using nondominated sorting in genetic algorithms. *Evol. Comput.* **2**(3), 221–248 (1994)
37. Telikani, A., Tahmassebi, A., Banzhaf, W., Gandomi, A.H.: Evolutionary machine learning: a survey. *ACM Comput. Surv.* **54**(8), 161:1–161:35 (2021)
38. Tian, Y., Si, L., Zhang, X., Cheng, R., He, C., Tan, K.C., Jin, Y.: Evolutionary large-scale multi-objective optimization: a survey. *ACM Comput. Surv.* **54**(8), 174:1–174:34 (2021)
39. Valdavia, A., Sanchez-Monedero, J., Casillas, J.: How fair can we go in machine learning? assessing the boundaries of accuracy and fairness. *Int. J. Intell. Syst.* **36**(4), 1619–1643 (2021)
40. Verma, S., Rubin, J.: Fairness definitions explained. In: 2018 IEEE/ACM International Workshop on Software Fairness (FairWare), pp. 1–7. IEEE (2018)
41. Witten, I.H., Frank, E., Hall, M.A., Pal, C.J.: Data Mining: Practical Machine Learning Tools, 4th edn. Morgan Kaufmann (2016)

42. Wu, H., Ma, C., Mitra, B., Diaz, F., Liu, X.: A multi-objective optimization framework for multi-stakeholder fairness-aware recommendation. *ACM Trans. Inf. Syst.* **41**(2) (2022)
43. Zemel, R., Wu, Y., Swersky, K., Pitassi, T., Dwork, C.: Learning fair representations. In: International Conference on Machine Learning, pp. 325–333 (2013)
44. Zhang, C., Liu, C., Zhang, X., Almanidis, G.: An up-to-date comparison of state-of-the-art classification algorithms. *Expert. Syst.* **82**, 128–150 (2017)
45. Zhang, Q., Liu, J., Zhang, Z., Wen, J., Mao, B., Yao, X.: Mitigating unfairness via evolutionary multi-objective ensemble learning. *IEEE Trans. Evol. Comput.* (2022). <https://doi.org/10.1109/TEVC.2022.3209544>
46. Zoller, M.A., Huber, M.F.: Benchmark and survey of automated machine learning frameworks. *J. Artif. Intell. Res.* **70**, 409–472 (2021)

Part V

Applications of Evolutionary Machine Learning

In which we bring together a few of the contemporary application areas for evolutionary computation and machine learning: Medicine, Robotics, Finance, Science and Engineering, Environmental Science, Games and Control.

Chapter 18

Evolutionary Machine Learning in Science and Engineering



Jianjun Hu, Yuqi Song, Sadman Sadeed Omee, Lai Wei, Rongzhi Dong,
and Siddharth Gianey

Abstract Evolutionary machine learning (EML) has been increasingly applied to solving diverse science and engineering problems due to the global search, optimization, and multi-objective optimization capabilities of evolutionary algorithms and the strong modeling capability of complex functions and processes by machine learning (ML) and especially deep neural network models. They are widely used to solve modeling, prediction, control, and pattern detection problems. Especially EML algorithms are used for solving inverse design problems ranging from neural network architecture search, inverse materials design, control system design, and discovery of differential equations.

18.1 Introduction

Several common fundamental research themes frequently arise in many science and engineering domains such as modeling physical and chemical processes, prediction, pattern classification, abnormality recognition, generation of structures, control, and inverse design. In these problems, the traditional analytical models are increasingly replaced or complemented by data-driven machine ML and especially deep neural network models [70], due to their strong capability to learn complex nonlinear relationships and inherent representations that lead to high-performance prediction models. For example, ab initio crystal structure prediction (CSP) has been a long-time challenging problem due to the expensive first principle calculations needed to evaluate the candidate structures during the search. However, currently, significant progress is being made in learning neural network-based interatomic potentials to speed up the CSP or molecular dynamics simulation process [60]. As a result, ML and especially deep learning (DL) have been transforming almost every discipline of science and engineering.

J. Hu (✉) · Y. Song · S. S. Omee · L. Wei · R. Dong · S. Gianey
Department of Computer Science and Engineering, University of South Carolina, Columbia,
SC 29201, USA
e-mail: jianjunh@cse.sc.edu

While deep neural network models are good at modeling, their gradient-descent-based training algorithms can only be applied to differentiable networks. However, there are many science and engineering problems that need strong global (multi-objective) optimization or search capability in vast design space, e.g., of materials and molecules or searching discrete structures such as neural network architectures. In this regard, it is natural to hybridize ML models with evolutionary algorithms to achieve synergistic high performance in problem-solving. Evolutionary computation (EC) has been applied to various stages of ML and DL for model search, feature selection, or hyperparameter tuning. It has played an increasing role in a range of scientific research tasks due to its global search and multi-objective optimization capabilities such as crystal structure prediction or inverse design [5] in which a surrogate performance evaluation model is trained and then used as the objective function in inverse design search. With these complementary roles of ML and evolutionary computation, there emerges the EML paradigm with unique capabilities and applications in diverse disciplines.

EML methods can be understood from different perspectives. From the ML problem point of view, EML has been used in solving supervised learning, unsupervised learning, and reinforcement learning problems. For example, it has been used for clustering [54], classification [106], regression [33, 87], and ensemble learning [40]. From an algorithm and model point of view, EML can be classified into two categories including: (1) EC as ML tools, in which evolutionary algorithms are directly used to solve diverse ML problems such as clustering, classification [41], or regression (2) EC for ML, in which EC is used to improve the model design (such as network architecture as shown in the evolutionary DL [126]), training of ML models, hyperparameter search, feature engineering, and explainability.

Some prominent applications of EML in science and engineering are shown in Fig. 18.1. These applications have some unique characteristics. First, most physical, chemical, and engineering processes are highly nonlinear and are difficult to model explicitly, so neural networks have thus been exploited to model such complex processes. Similarly, the complexity of patterns from these systems has also called for the application of ML and DL in these areas. So both EC as ML and EC for ML have been widely used in science and engineering for clustering, classification, regression, pattern detection, etc. However, there is a special category of application of EML in science and engineering: the (inverse) design problem, which ranges from neural network architecture [15, 66, 111] and parameter design, to partial differential equation discovery [121], inverse materials design [122], neural control system design [84], crystal structure prediction [37], and engineering system design. In these problems, usually, the objective function or performance evaluation is modeled using an ML or neural network model, and then genetic algorithms (GAs) are used to search the candidate solutions in the design space using ML as the objective function. This problem-solving strategy also applies to the problem of discovering physics equations, in which the terms of (partial) differential equations can be discovered by evolutionary algorithms and then assembled either by GAs or by DL.

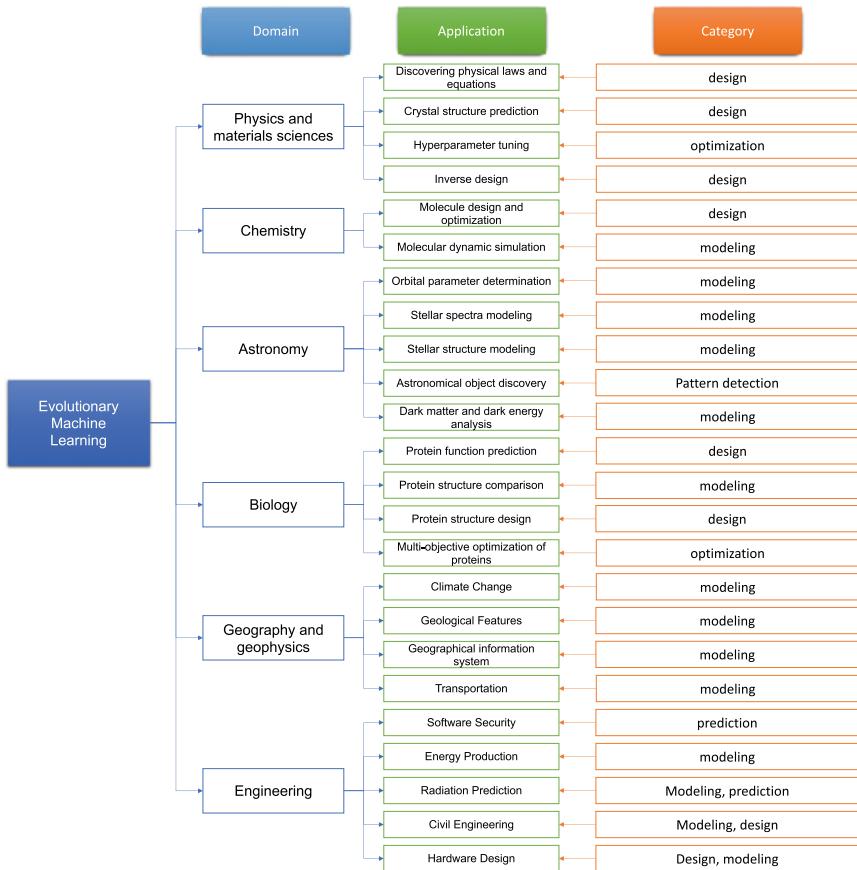


Fig. 18.1 Categories of problems suitable for solving using EML

18.2 Applications of EML in Science and Engineering

EML has gained widespread recognition and has been extensively utilized in diverse fields of science and engineering. In the subsequent sections, we provide a comprehensive overview of the manifold applications of EML in various domains, including physics and materials sciences, chemistry, astronomy, biology, geography, and other branches of engineering. The comprehensive breakdown of the contents pertaining to the applications of EML in these domains is graphically represented in Fig. 18.1.

18.2.1 EML for Physics and Materials Sciences

The application of ML and especially DL in physics and materials science has been accelerating due to the complexity in manually modeling the structure-property relationship, the challenge in ab initio simulation for materials characterization, and the challenge in sampling the huge chemical space. To address these issues, ML and deep neural networks are now commonly used to learn the interatomic potentials for Density Functional Theory (DFT) or molecular dynamics simulation, which can then be combined with global optimization algorithms for crystal structure prediction [8]. Neural networks are also trained with ab initio data to speed up materials property calculation so that they can be used as fast surrogate models for inverse design of materials using evolutionary algorithms [5]. Evolutionary algorithms can also be combined with ML to discover physical laws as represented by partial differential equations [119, 121]. Finally, evolutionary algorithms are also routinely used for training special neural network models due to their gradient-free global optimization capability.

18.2.1.1 Discovering Physical Laws and Equations

Discovering physical laws represented as mathematical models such as partial differential equations (PDE) is one of the most challenging tasks in physics. Data-driven methods have been routinely used for PDE discovery, but they usually require all potential terms to be specified. Xu et al. [119, 121] proposed a novel method, DLGA-PDE for PDE discovery without such constraints. It works by training a DL model of the physical process for generating meta-data and derivatives and then using a GA to search for the combination of such terms. Their method has been shown to achieve good performance in the discovery of the Korteweg-de Vries (KdV) equation, the Burgers equation, the wave equation, and the Chaffee-Infante equation, all with an incomplete term library and even with noisy and limited data. In [26], Chen et al. used an evolutionary strategy to train a discrete feed-forward convolutional neural network model for modeling variational wave functions for correlated many-body quantum systems. They found that networks can converge with high accuracy to the analytically known sign structures of ordered phases. In [120], a robust PDE discovery framework called the robust DL GA (R-DLGA) was proposed, which combines the physics-informed neural network (PINN) and DLGA for potential terms discovery. The terms discovered by DLGA are added to the loss function of the PINN as physical constraints to improve the accuracy of the derivative calculation. The authors [121] further proposed an EML method for DL of parametric partial differential equations from sparse and noisy data. The EML method has also been applied to learn models that balance accuracy with parsimony in classical mechanics and the melting temperature prediction of materials [38]. The EML approach allowed them to discover interpretable physical laws from data based on parsimonious neural networks (PNNs) combined with evolutionary optimization. The EML approach

has also been applied to learn parametric partial differential equations including the Burgers equation, the convection-diffusion equation, the wave equation, and the KdV equation from sparse and noisy data [121], in which the neural network is first trained to calculate derivatives and generate meta-data, which solves the problem of sparse noisy data. Then, the GA is used to discover the form of PDEs and corresponding coefficients.

18.2.1.2 Crystal Structure Prediction

One of the major challenges in materials science is to predict a structure given only its composition. While several crystal structure prediction algorithms based on global optimization and first principle (e.g., Density Functional Theory (DFT)) have been proposed, the computational complexity of such DFT simulations have made it difficult to predict structures for most materials. To address this issue, a series of research combined deep neural networks with GAs for crystal phase determination [8, 37]. In [8], Artrith et al. trained a specialized ML neural network potential using around 1000 first-principles calculations, which can help sample low-energy atomic configurations in the entire amorphous Li_xSi phase space. The result is comparable to that of ANN trained with extensive molecular dynamics simulations with $\approx 45\,000$ first-principles calculations. Such neural network potentials (including graph neural network potentials) have since been further improved or developed and applied with evolutionary algorithms [17, 28]. In [130], Bayesian optimization is combined with graph neural network (GNN) potential to do crystal structure relaxation. The GNN is then later combined with a GA for de novo crystal structure prediction [31]. Wanzenböck et al. [76] proposed an algorithm that explores the rich phase diagram of TiO_x overlayer structures on SrTiO₃(110) by combining the covariance matrix adaptation evolution strategy (CMA-ES) and a neural network force field (NNFF) as a surrogate energy model, which dramatically reduces the computational resources needed by DFT simulation. While most neural network potentials are trained before the genetic search of crystal phases, it is also possible to conduct both simultaneously, which is how active learning works. In [91], Podryabinkin et al. proposed a methodology for crystal structure prediction based on the evolutionary algorithm and the active learning of neural network interatomic potentials. Their approach allows for an automated construction of an interatomic interaction model from scratch achieving a speedup of several orders of magnitude. They have benchmarked their algorithms on crystal structure prediction of carbon, high-pressure phases of sodium, and boron allotropes, including those that have more than 100 atoms in the primitive cell, all with satisfactory results. Kang et al. [60] trained a deep neural network interatomic potential model and combined it with a GA for crystal structure prediction. By avoiding the expensive DFT calculations of formation energy and harnessing the speed and accuracy of neural network potentials (NNPs), their algorithm navigates configurational spaces $10^2 - 10^3$ times faster than DFT-based methods. Their SPINNER algorithm has identified more stable phases in many cases than the data-mined template-based method and DFT-based evolutionary algorithm methods.

18.2.1.3 Hyperparameter Tuning

Another major application of EML in physics and materials is hyperparameter tuning of ML models as shown in [109]. Both particle swarm optimization (PSO) and GAs are routinely used to select optimal hyperparameter values autonomously. Tani and Rand evaluated how PSO and GA can improve the performance of their XGBoost ML model for the ATLAS Higgs boson ML challenge (HBC) [3], which represents a typical application of ML algorithms to the field of high energy physics. The task of the HBC is to separate the Standard Model (SM) Higgs boson signal from the large SM background. They showed that compared to using the default hyperparameters, the optimization of the hyperparameter values by GA or PSO improves the sensitivity of the data analysis, by 12-13%, demonstrating that the optimization of hyperparameters is a worthwhile task for data analyses in the field of HEP. In [50], DL techniques are applied to learn the physics of extensive air showers in which the inner structure of the neural network is optimized through the use of GAs.

18.2.1.4 Inverse Design

Another major application of EML in physics is the inverse design in which the ML models are used to learn the relationship between structures and physical property while evolutionary algorithms are used to search the design space. In [32, 46], Comin and Hartschuh combined neural networks with a GA for optimizing spectral-phase shaping of an incident field to achieve second harmonic generation hotspot switching in plasmonic nanoantennas design. They first trained a neural network to predict the relative intensity of the second-harmonic hotspots of the nanoantenna for a given spectral phase and then used a GA to generate a wide range of nanoantenna designs to be fed into the neural network. Taking advantage of the multi-objective optimization feature of the GA, Li et al. [75] applied a multi-objective GA to the optimization of the apertures of the National Synchrotron Light Source II (NSLI-II) Storage Ring. To maintain the diversity of the population of the GA, a K-means clustering algorithm is applied to the population to group individuals into clusters of different fitness levels. Then individuals from the “Good” and “Poor” clusters are mixed with the “Best/elite” individuals to ensure population quality improvement without losing diversity. In [29], Chen et al. proposed an EML approach for physics-guided ML-based inverse design of acoustic metamaterials. They used a multi-layer perceptron (MLP) neural network to map the wave-field-to-wave propagation relationship and then used it as a surrogate model for GA for inverse design of the metabeam. While deep neural networks are widely used as surrogate models, they are not able to tune the output to encompass a range of input states. In this case, a nonlinear symbolic regression model by genetic programming is more desirable, which is used in metamaterials design [5]. In their work, the actual design is implemented using the MLP-based autoencoder network model. The EML-based inverse design has also been applied to designing solid oxide fuel cells (SOFCs) [122], in which a deep neural network is used to map the current density, anode flow rate, cathode flow rate, and temperatures

to outputs that reflect the efficiency and thermal behavior of the SOFC cell. Moreover, the EML has also been used in inverse design of photonics in [52].

There are several other novel applications of EML in physics problems. In [26], Chen et al. used a MLP network to approximate a novel class of variational wave functions for correlated many-body quantum systems. They encode the all-important rugged sign structure of a quantum wave function in a convolutional neural network with discrete output, which is then trained with a gradient-free evolution strategy (ES) algorithm rather than the commonly used back-propagation algorithm. They found that while the stochastic gradient descent (SGD) algorithm is better for optimizing continuous functions, the ES method is the better choice for optimizing the variational wave function while SGD is no longer applicable. In [116], a multi-objective GA is combined with deep neural network models for improving the performance and durability of direct internal reforming solid oxide fuel cells.

18.2.2 *EML for Chemistry*

Evolutionary algorithms (EAs) are generic, population-based, metaheuristic optimization methods. The mechanisms by which EAs operate are inspired by biological evolutionary operations such as selection, mutation, recombination, and reproduction. Combined with ML, GAs provide a novel tool for the investigation of molecule design, optimization, and molecular dynamic simulation.

18.2.2.1 *EML for Molecule Design and Optimization*

The rise of machine intelligence provides a grand opportunity to expeditiously design and discover novel molecules through smart search. The discovery of new functional molecules has led to many technological advances and remains one of the most critical approaches to overcoming technical problems in various industries, such as those in organic semiconductors, displays, and batteries.

Evolutionary design has gained significant attention as a useful tool to accelerate the design process by automatically modifying molecular structures to obtain molecules with the target properties. However, devising a way to rapidly evolve the molecule while maintaining its chemical validity is a challenge. Kwon et al. [68] proposed a GA along with RNN and DNN models that were used to evolve the fingerprint vectors of seed molecules. The RNN decoder reconstructed chemically valid molecular structures in the SMILES format from the evolved fingerprint vectors without resorting to predefined chemical rules. The method employed DL models to extract the inherent knowledge from a database of materials and is used to effectively guide the evolutionary design. Designing a new therapeutic drug can be a time-consuming and expensive process. It also enables rapid discovery of new drug candidates by performing intelligent searches in a wide molecular structure space. In [2], Abouchekeir et al. proposed a new approach called adversarial deep

evolutionary learning (ADEL) to search for novel molecules in the latent space of an adversarial generative model and keep improving the latent representation space. In [125], Yoshikawa et al. proposed a new population-based approach using a grammatical evolution named ChemGE that can update a large population of molecules concurrently and ChemGE succeeded in finding hundreds of candidate molecules whose affinity for thymidine kinase is better than that of known binding molecules in a database (DUD-E). Li et al. [74] proposed a deep evolutionary learning (DEL) process that integrates a fragment-based deep generative model and multi-objective evolutionary computation for molecular design, which can generate promising novel molecular structures.

Optimizing molecules for desired properties is a fundamental yet challenging task in chemistry, material science, and drug discovery. In [27], Chen et al. developed a novel algorithm for optimizing molecular properties via an Expectation Maximization (EM) like an explainable evolutionary process. They showed that the evolution-by-explanation algorithm is 79% better than the best baseline in terms of a generic metric combining aspects such as success rate, novelty, and diversity.

Evolutionary algorithms have found increasing applications in both the discovery and optimization of novel molecular structures. Artificial evolutionary methods, such as GAs, can not only explore large and complex search spaces very efficiently, but also can be applied to the identification and optimization of new molecules faster than pure physical experiments. ML models can enhance the suitability of experimentally measured molecules to accelerate the discovery of useful and novel molecules in a broad composition or property space. For example, Tu C. Le and Nhiem Tran [69] reviewed how GAs have been used to solve optimization problems in computational drug design including catalyst discovery and optimization. They also describe the use of both experimental and computational fitness functions to evolve materials into promising areas of catalyst space. Among these applications, neural networks have been used widely as in silico fitness functions based on these neural structure-property models.

18.2.2.2 EML for Molecular Dynamic Simulation

Molecular dynamics (MD) has become a powerful tool for studying biophysical systems due to increased computing power and the availability of software. Efficient computational strategies for the targeted generation and screening of molecules with desired therapeutic properties are therefore urgently required.

Because of the negative environmental impact of damaging organic solvents and the high cost of chemical waste disposal, the search for alternative, renewable solvents is a top priority in the chemical industry. Zhong et al. [128] facilitated the development of optimized potentials for liquid simulation (OPLS)-based force field (FF) parameters for eight unique deep eutectic solvents (DESs) based on three ammonium-based salts and five HBDs at multiple salt-HBD ratios. DESs are a class of solvents often composed of ammonium-based chloride salts and a neutral hydrogen bond donor (HBD) at specific ratios. These cost-effective and environmentally friendly

solvents have seen significant growth in multiple fields, including organic synthesis, and in materials and extractions because of their desirable properties. Coronavirus disease 2019 (COVID-19) caused by severe acute respiratory syndrome coronavirus type 2 (SARS-CoV-2) has led to a global pandemic. DL techniques and molecular dynamics (MD) simulations are two mainstream computational methods for studying the geometric, chemical, and structural features of proteins and guiding protein structures. Sun et al. [107] introduced the latest progress of the DL-based molecular dynamic simulation approaches in structure-based drug design (SBDD) for SARS-CoV-2 which could address the problems of protein structure and binding prediction, drug virtual screening, molecular docking, and complex evolution. MD can also help researchers develop new molecular materials in biomaterials science. Collagen is the most abundant structural protein in humans, providing crucial mechanical properties, including high strength and toughness, in tissues. In [61], Khare et al. developed a general model using a GA within a DL framework to design collagen sequences with specific Tm values. They discovered that the number of hydrogen bonds within collagen calculated with molecular dynamics (MD) is directly correlated to the experimental measurement of triple-helical quality.

18.2.3 EML for Astronomy

The fundamental advantage of EML is that they deliver high-quality results even when computational resources are constrained. The so-called comprehensive optimization methods are primarily constrained by the scale of the problem or data since the field of astronomy often involves optimizing problems of great complexity or processing enormous amounts of data. For this reason, EML has been used in various significant applications in astronomy such as orbital parameter determination, stellar spectra modeling, stellar structure modeling, planet search, gamma emission analysis.

18.2.3.1 Orbital Parameter Determination

Due to the sheer number of unknown parameters involved in the problem of finding orbital parameters, it is considered a highly difficult task. As a result, in the absence of an efficient approach, one is forced to either accept an extremely coarse-grained parameter space scan or limit the search space by setting the ranges of certain parameters. Wahde [112] proposed a method based on GA for efficiently searching vast space of possible orbits. The goal of this paper was to evaluate the effectiveness of using a GA-based method to determine the orbital parameters of interacting galaxies provided photometric observations and systemic velocities of the pair of galaxies. Later Wahde and Donner [113] extended the simulation part of this GA-based approach to investigate the impact of past interactions between the NGC5195 and the Messier M51 galaxies. In another work, Theis and Kohle [110] showed that

if adequate data are provided, their GA-based method can reliably calculate orbital parameters. The use of a GA theoretically enables a uniqueness test of a preferred parameter combination and so the authors use it on the parameter region determined after the fast restricted N-body method. Cantó et al. [21] designed a modification of the canonical GA to observe the orbital parameters of the planets orbiting 55 Cancri. The GA is predominantly used here to maximize a function where traditional methods are ineffective.

18.2.3.2 Stellar Spectra Modeling

Modeling a good fit of cosmic stellar spectra is a very challenging objective. However, this task is very crucial as a wide range of stellar attributes can be inferred from this. The first known evolutionary computing-based analysis of stellar spectra was performed by Metcalfe [79]. The work proposed employing a GA to compare the observed light curves to those produced by theoretical models to determine the properties of binary stellar systems. In this work, GA was used to randomly populate the defined parameter space, which also allowed the trial parameter sets to evolve over time. The optimal set of parameters and the mean set of parameters have very negligible differences after 100 generations. Mokiem et al. [82] developed a parallelized GA which served as the foundation for an autonomous fitter of the spectra of massive stars with stellar winds. With the utilization of a rapid performance stellar atmosphere code named FASTWIND [92] and a fitting method based on GA named PIKAIA [25], a fast and efficient method for automating the fitting of the continuum normalized spectra of O⁻ and early B⁻ type stars with stellar winds is described in this paper. The GA-based routine PIKAIA is employed for parameter optimization of FASTWIND. PIKAIA optimizes a population till a predetermined number of generations is reached rather than until a specified criterion is met. PIKAIA is also used on other several noteworthy papers. One such important application can be found in the work by Baier et al. [14]. They combined the radiative transfer code DUSTY [58] with PIKAIA to notably enhance the spectral fit of the dust spectra of AGB stars. PIKAIA, which is based on the evolutionary natural selection process, seeks to maximize the function $g(\lambda) = [F(\lambda) - F_m(\lambda)]^{-2}$, where $F(\lambda)$ is the observed spectrum and $F_m(\lambda)$ is a model spectrum computed using DUSTY.

18.2.3.3 Stellar structure Modeling

Another important application of EML in astronomy is stellar structure modeling. The internal structures of stellar objects vary depending on their classifications and ages, reflecting the components they are made of and how they transfer energy. Metcalfe and Charbonneau [80] obtained a number of intriguing physical discoveries for stellar structure modeling of white dwarf stars, thanks to the effective, concurrent exploration of parameter space enabled by GA-based numerical optimization. The authors also use the GA-based routine PIKAIA [25] (described in the

paragraph above) and re-implement it as a fully parallel routine to provide an objective estimate of the globally optimal parameters for a particular model versus an observational dataset. A total of five parameters were allowed to evolve by the parallel GA for obtaining structural and physical details about the white dwarf stars. Zhang et al. [127] calculated stellar effective temperatures and identified angular parameters using a stochastic PSO on known stellar flux data in specific bands. The system’s input settings were first set. Second, each particle’s fitness value was determined. The fitness values of each particle with the prior best predictions were then contrasted. Following the generation of the new particle, other particles’ positions and velocities were updated. When a specific stopping criterion is met, the stochastic PSO terminates. Another work describes a novel approach based on GA for estimating the age and relative contribution of various stellar populations in galaxies [9]. Using charge-coupled device (CCD) images in the U, B, V, R, and I bands, the authors apply this technique to the barred spiral galaxy NGC 3384. Using the hypothesis that just two stellar populations, each with a different color, age, metallicity, etc., are responsible for the observed light from a galaxy, the GA is used to solve the equation set that describes the relationships characterizing the two stellar populations’ mixing-tracks found from [1].

18.2.3.4 Astronomical Object Discovery

Even though there are a great number of astronomical objects, we have only discovered and examined a small fraction of them. This necessitates the creation of algorithms for efficiently finding astronomical objects. Nesseris and Shafieloo [86] developed a null test for the cosmological constant model using the so-called Om statistic in combination with GAs to recreate the expansion history of the universe in a model-independent way. In this paper, the GA is applied on the SNIa dataset (with the selected execution parameters) [99] to find a solution for the “distance modulus” term of its fitness function. Based on evolutionary optimization of the classifiers, Wierzbinski et al. [117] created an effective and precise classifier for cosmic objects that are mostly used to discover the best parameters for the voting classifiers to categorize stellar spectra of stars, quasars, and galaxies. With their default parameters as a starting point, the authors trained a collection of 21 classifiers. Then, they optimized the hyperparameters using GA. Cassisi [23] used an evolutionary algorithm in which the equations for convective mixing and nuclear burning are solved using a single common scheme to undertake the first complete evolutionary computations of stars undergoing “He flash mixing”. For the evolutionary calculations, the authors employ an evolutionary code from [103]. Joseph et al. [59] used GA to search through four nonlinear parameters of each planet to fit a full Keplerian orbit. Chan et al. [24] used a GA for locating stars formed after supernovae explosion and eventually backtracking to stars formed after the big bang. As applying a complete search is computationally very expensive, the authors employed a GA with a local search boosted with random initial solutions. In a study by Geyter et al. [34], the SKIRT [12], a Monte Carlo radiative transfer code designed to examine the impact of dust absorption and

scattering on the gas and star kinematics of dusty galaxies, its output is optimized using the GA library GAlib [114] by searching through the vast model parameter space.

18.2.3.5 Dark Matter and Dark Energy Analysis

Dark energy makes up around 68% of the universe and dark matter about 27%. They are predominantly responsible for the bulk of galaxies and galaxy clusters as well as the large-scale organization of galaxies. So they are very significant to understand the formation of the universe. Bogdanos and Nesseris [18] employed GAs to study standard SNIa data [67] to extract model-independent restrictions on the evolution of the dark energy equation of state. With the selected execution settings, the GA is applied to the original SNIa dataset to produce a solution for the reduced distance modulus. Ruiz et al. [95] applied PSO to analyze merger trees obtained from a common Lambda Cold Dark Matter N-body simulation and the Semi-analytic Model of Galaxy Formation (SAM). The PSO is mainly used here to calculate the best possible set of SAM parameters. Moster et al. [83] devised a reinforcement learning approach to compute the galaxy properties for dark matter haloes and train the parameters using a PSO technique. The authors compute the galaxy properties for all haloes for a specific set of weights and biases and then produce mock statistics for GalaxyNet, which is trained using a reinforcement learning approach. These statistics are compared to the observations to determine the model loss, which is minimized using PSO.

18.2.4 EML for Biology

The protein structure design problem is one of the most exciting challenges of modern computational biology. Because of its scientific complexity, research on understanding the function of proteins, and studying the relationship between amino acid sequences and protein structures is very difficult. ML-guided evolution is a new paradigm for a biological design that enables optimizing complex procedures. And the multi-objective evolutionary algorithm is introduced as it can deal with several functions when designing protein structures.

18.2.4.1 Protein Function Prediction

Protein function prediction methods are techniques that bioinformatics researchers use to understand the biological or biochemical roles of proteins. Proteins often function poorly when used outside their natural contexts and directed evolution can be used to engineer them to be more efficient in new roles. Wu et al. [118] incorporate ML into the protein's directed evolution workflow, to reduce the experimental effort and to

explore the sequence space encoded by mutating multiple positions simultaneously. They have validated their approach on a large published empirical fitness landscape and demonstrated that ML-guided directed evolution finds variants with higher fitness than other evolution approaches.

18.2.4.2 Protein Structure Comparison

Identifying structural similarities is essential for assessing the relationship between structure and function in proteins. Szustakowski and Weng [108] developed a structure alignment algorithm using GA for three-dimensional structures of proteins. They first align the proteins' cores, as represented by their secondary structure elements, by minimizing the difference of distance matrices using a GA. And then extend the alignment to include any positions in loops or turns deemed equivalent in a convergent process. Carr et al. [22] developed a new approach to structural comparison by using a Multimeme evolutionary algorithm. In a Multimeme algorithm, an individual is composed of its genetic material (that represents the solution to the problem being solved) and its memetic material (that defines the kind of local searcher to use). During crossover, both genetic and memetic transmission will be done. In [11], Bacardit et al. used GA to design automated procedures to reduce the dimension of protein structure prediction datasets by simplifying how the primary sequence of a protein is represented. Reducing the size of the alphabet used for prediction from twenty to just three letters resulted in more compact and human-readable classifiers. And the loss of accuracy accrued by this substantial alphabet reduction is not statistically significant compared to the full alphabet.

18.2.4.3 Protein Structure Design

ML-guided directed protein structure design enables optimization of complex functions. Pegg et al. [90] developed a GA for structure-based de novo design. They use molecular interactions evaluated with docking calculations as a fitness function to reduce the search space. Durrant et al. [39] developed a protein inhibitor design algorithm that uses a growth strategy to build the core scaffold, molecular fragments are added at random to this scaffold. An evolutionary algorithm is then used to evaluate the scores of each population member, and the best ones become founders of the subsequent generation.

18.2.4.4 Multi-objective Optimization of Proteins

Using only one energy function is insufficient to characterize proteins because of their complexity. Multi-objective algorithms can provide a better protein with less computational resource requirement than DL methods. Brasil et al. [19] developed a multi-objective evolutionary algorithm for ab initio protein structure prediction

without using any earlier knowledge from similar protein structures. Gao et al. [44] adopted a solvent-accessible surface area into a multi-objective evolutionary algorithm with three objective functions to improve protein structure prediction accuracy and efficiency. Traditional multi-objective algorithms cannot obtain the desired solution because the selection pressure decreases as the number of objectives increases. To address this problem, Lei et al. [71] proposed a many-objective evolutionary algorithm with four types of objectives to alleviate the impact of imprecise energy functions for predicting protein structures.

18.2.5 EML for Geography and Geophysics

Process in evolutionary ML has contributed to significant advances in geography and geophysics research, such as climate change [94], geological features, geographical information systems, and transportation.

18.2.5.1 Climate Change

Recently, one of the biggest problems confronting humanity is climate change. Storms, droughts, fires, and flooding have become stronger and more frequent [94]. The main strategies for addressing climate change include reducing greenhouse gas (GHG) emissions and preparing for resilience and disaster management, named mitigation and adaption respectively. Mitigation of GHG emissions requires changes to electrical systems, transportation, buildings, industry, and land-use. Many experts are exploring how to use ML methods to tackle the issue of climate change. Considering both adaption and mitigation response to climate change, Paton et al. [89] incorporated GHG emissions into the multi-objective evolutionary algorithm (MOEA). The application of this method in Adelaide, Australia's southern water supply system has illustrated the framework's useful management implications. For spatially allocating land-use, analyzing climate change impacts may be a useful and fundamental long-term adaptation strategy. Joo Yoon and co-workers[124] utilized multi-objective GA to identify climate adaptation scenarios based on existing extents of three land-use classes in South Korea. Specifically, five objectives were established for predicting climate change impacts and regional economic conditions: (1) minimization of disaster damage, (2) existing land-use conversion, (3) maximization of rice yield, (4) protection of high species richness areas, and (5) economic value. This method showed better performance than other spatial land-use compositions for all adaptation objectives. Climate change trends have already affected many ecosystems, such as species range and diversity, and this effect is different and varies from place to place. Rezayan et al. [97] provided an optimal combination of the common species distribution models (SDMs), and employed a GA to model the climate change effects on the spatial distribution of *Quercus brantii* in the west of Iran.

18.2.5.2 Geological Features

Several attempts have been made to predict and analyze geographic properties by EML methods, such as earthquake hypocenter location [64, 96, 102], surface water reservoir control [100], ocean wave height [115], and magnetic anomalies [16].

In the geophysical research area, the determination of reliable and accurate earthquake hypocenters is a crucial task. In 1993, Sambridge and Gallagher [102] published a paper in which they used GAs to predict earthquake hypocenter location, which can refine a population of hypocenters collectively by exploiting information from the group as a whole, rather than relying on only local information about a single hypocenter. Kim et al. [64] applied a GA and a two-point ray tracing method [63] to solve non-uniqueness problems in determining reliable hypocentral parameters including latitude, longitude, source depth, and origin time. With the increasing risk of flood and drought impacts and the changing water allocation requirements among complex users, an efficient multipurpose reservoir management strategy is critical. Salazar et al. [100] carried out a diagnostic assessment framework of the surface water reservoir control for the Conowingo reservoir in the Lower Susquehanna River Basin, Pennsylvania, USA. Specifically, they use Evolutionary Multi-Objective Direct Policy Search (EMODPS) [48] as the decision analytic framework where reservoirs' candidate operating policies are represented using parameterized global approximators. And then, they use multi-objective evolutionary algorithms to optimize those parameters for discovering the Pareto approximate operating policies. To predict the heights of ocean waves accurately and quickly, which is an important problem in marine detection and warning, Wang et al. [115] reported a hybrid Mind Evolutionary Algorithm-BP neural network strategy (MEA-BP). It can avoid early convergence and improve the prediction accuracy by combining the local searching capabilities of the BP neural network and the global searching ability of the MEA. Their experiments cover a wide range of geographical locations (from 12 observation points across two geographically distinct regions - the Bohai Sea, and the Yellow Sea) and different weather (from Jan 1, 2016, to Dec 31, 2016) and show faster running time and high prediction accuracy. Balkaya and co-workers [16] demonstrated the differential evolution algorithm for 3D nonlinear inversion of total field magnetic anomalies caused by vertical-sided prismatic bodies. Li et al. [72] used an artificial neural network (ANN) to capture ionospheric spatiotemporal characteristics with a powerful capacity and capability and used a GA to improve the ANN's learning efficiency for predicting ionospheric peak parameters including $foF2$ and $hmF2$.

18.2.5.3 Geographical Information Systems

GAs have recently generated much interest in the field of geographical information system (GIS) including optimal location search [73], and land partitioning [35]. Optimal location search is usually required in many urban applications for establishing one or more facilities. When it involves multiple sites, various constraints, and multiple objectives, the search task is very complex. In work [73], it demonstrated that

GAs can be used with GIS to effectively solve the spatial decision problems for n facilities. Land partitioning is a basic process of land consolidation [105], it involves the subdivision of land into smaller sub-spaces subject to several constraints [36]. Demetriou et al. [35] proposed the GIS and GAs integrated model: Land Parceling System (LandParcelS) which automates the land partitioning process by designing and optimizing land parcels according to their shape, size, and value.

18.2.5.4 Transportation

Bicycle-sharing systems have become an important part of urban transportation systems [43]. To increase the number of bicycles available for rent and improve profits, it is better to collect bicycles in the evening and redistribute them to the main stations. This relies on the model that accurately forecasts rental demand. In [45], Gao et al. presented a moment-based rental prediction model by a fuzzy C-means (FCM)-based GA with a back propagation network (BPN) with more than ten factors, including the date, time, weather (e.g., temperature, humidity, and wind speed) and season, all of which make different contributions to the final demand. Firstly, they use the unsupervised FCM-based GA method to pre-classify historical rental records into groups. Next, the classification results are fed into a BPN predictor which is trained using these categorized records. After training, the BPN predictor can predict the demand at future moments.

18.2.6 EML for Engineering

Evolutionary algorithms are being used to enhance and optimize traditional ML systems. The quick convergence and flexibility that comes with evolutionary algorithms make their performance so useful in so many different areas. The use of evolutionary algorithms with ML is growing, and it is being used in many engineering areas such as software security, energy production, radiation prediction, civil engineering, and hardware design.

18.2.6.1 Software Security

Cybersecurity and preventing unwanted attacks on computer networks is a very important and constantly changing field. Intrusion detection systems (IDS) are being used to keep malicious software out and are constantly being developed. With the addition of ML algorithms, these IDSs can classify network traffic as normal or not. Hosseini and Zade [53] made a novel hybrid intrusion detection method that has two phases, a feature selection phase, and an attack detection phase. In the feature selection phase, a GA is modified to have a new multi-parent crossover operator and combined with a support vector machine (SVM) to improve performance and help

reduce the dimensions and select relevant features. For the attack detection phase, an ANN was used to detect any attacks. To train the ANN, a hybrid system was created using hybrid gravitation search and PSO algorithms. Al-Yaseen et al. [4] proposed another IDS that involved the use of a wrapper-based differential evolution algorithm for feature selection and an extreme learning machine as the classifier. The differential evolution algorithm is similar to a GA, but mutation occurs first before crossover. The feature selection from this algorithm is fed into an ML algorithm to classify and detect intruders. In [51], Halim et al. proposed a similar IDS system that used a GA for feature selection as well as an ML algorithm-based prediction system. Another aspect of cyber security that is often developing is the detection of spam and phishing websites. Spammers are creating new techniques that dodge filters and phishing attacks are becoming a larger problem for web security. In [42], Farris et al. proposed a new detection system for spam using a GA in combination with a random weight network. This new system can better handle the massive data flow that needs to be processed to be able to accurately filter out spam. The system uses a GA to find possible feature subsets and an RNW as the base classifier. Ali and Ahmed [6] created a new hybrid phishing website prediction model. The model uses a GA for feature selection, figuring out the most important features and providing proportional weight to these features for optimal results, and applies it to a deep neural network to accurately predict which websites are phishing sites. Detecting where future attacks on software could come from is also very important to the security of that software. These are done using a vulnerability prediction model (VPM). Sahin et al. [98] designed a new model to predict vulnerabilities in software, applying symbiotic GA and DL methods. The proposed VPM used a deep neural network-based symbiotic GA to predict where the software may be lacking in security. The method used two versions of GA and a population-based dominance mechanism to identify the dominant-feature representations.

18.2.6.2 Energy Production

Energy production and consumption are both very necessary for this world. The shift to more natural energy sources has already started, but to see if they can even keep up with what a place or country needs, the amount of renewable energy produced needs to be optimized and measured, and the energy demand needs to be predicted. Prediction systems are developed to help with this, but the feature selection process can be improved with the help of evolutionary approaches. Zhou et al. [129] created a prediction system of photovoltaic cells to see solar energy production. The system utilized a GA and a customized similar day analysis based on an extreme learning machine to predict the photovoltaic output. In [101], Salcedo-Sanz et al. focused on renewable energy uses and developed a new system that used a coral-reef optimization algorithm with a substrate layer for feature selection. Then an extreme learning machine was used to predict renewable energy production. Hu and Chen [55] were looking more specifically at wind energy and wind speed prediction. The proposed model included a differential evolutionary algorithm in conjunction with a Long

Short Term Memory (LSTM) to accurately predict wind speeds and wind energy production. Neshat et al. [85] also used an LSTM in combination with six different evolutionary algorithms to test them all for wind speed and wind energy production predictions. Prediction systems for the demand of places are also being developed and enhanced using evolutionary algorithms. Mason et al. [78] created a new system that applied a Covariance matrix adaptation evolutionary strategy to train neural networks to predict the energy demand for Ireland. It resulted in accurate predictions with fast convergence times. Seyedzadeh et al. [104] similarly made an energy consumption prediction model for non-domestic buildings. A gradient-boosted regression tree model was used for prediction, and an evolutionary algorithm was used to optimize the model by adjusting the hyperparameters.

18.2.6.3 Radiation Prediction

Ghimire et al. [47] proposed a self-adaptive differential evolutionary extreme learning machine (ELM) to predict daily solar radiation for solar-rich cities. For the feature selection for the predictors of solar radiation, a swarm-based ant colony algorithm was used. The learning algorithm used was optimized from a standard ELM: a self-adaptive differential evolutionary extreme learning machine. This system was compared to others used to accomplish the same task, and it performed on par, if not more accurately for predicting solar radiation forecasts. In [62], Kilic et al. were looking at global solar radiation prediction. They implemented a hybrid ANN system, using an evolutionary algorithm for feature selection and to help train the ANN. The hybrid system was compared to usual ML algorithms for the prediction of radiation, including an ANN, SVM, and a DL model. Guijo-Rubio et al. [49] did something similar, using an evolutionary algorithm to train and evolve an ANN to predict solar radiation. Marzouq et al. [77] utilized a GA for the selection of critical inputs for the ANN to predict solar radiation. This new evolutionary ANN model outperformed usual algorithmic models for solar radiation prediction. Amiri et al. [7] took the solar radiation prediction further, predicting on a tilted plane. The evolutionary algorithm was used to find the proper topology, and the study was conducted in two ways, assuming the satellites where the data was pulled from had the same tilt angle, and assuming the angles could be different.

18.2.6.4 Civil Engineering

Inducing blasts is a common occurrence in rock mining and construction. Being able to predict the vibrations of these blasts and the impact they will have on the surroundings is key to the safety of the operation and the workers themselves. When researching the vibrations, the goal is to measure the peak particle velocity as a base parameter. Chen et al. [30] proposed new solutions to the problem, combining various evolutionary algorithms with either an SVM or an ANN. A firefly algorithm, GA, PSO algorithm, and a modified firefly algorithm were all hybridized with both

an SVM and an ANN in two separate systems, and the results of all were compared. The systems were all accurate at predicting peak particle velocity (PPV), with the modified firefly algorithm support vector regression (SVR) system performing the best. Azimi et al. [10] did something similar, focusing on using a GA to optimize an ANN to predict blast vibrations for quarry mining. When constructing different reinforced concrete structures, beam-column joints are an important point in the construction, often being the first place to shear and break off. Yaseen et al. [123] predicted the shearing of these joints with the use of a GA combined with a deep neural network. The GA was used for the input selection during the modeling phase, and the DLNN was used for the prediction phase: to identify any structural problems. Huang et al. [56] also predicted the shearing of steel fiber-reinforced concrete columns. Two hybrid systems were created, combining a GA with an ANN and a PSO algorithm with an ANN. In [20], Cai et al. used a GA hybridized with a back-propagation neural network to predict the capacity reinforced concrete beams had to flex in the structure. Moayedi et al. [81] created a prediction system for ultimate bearing capacity, to see the possible load a bearing can handle. Many systems were developed, including a hybrid GA and ANN system, and a PSO and ANN system to predict the bearing capacity.

18.2.6.5 Hardware Design

The optimal design for hardware and technologies is a big part of engineering and production. Owoyele et al. [88] proposed an automated ML GA as the system for optimizing the design of engines. Using computational fluid dynamic simulations, the system can optimize engines with a hands-off approach. In [13], Bagheri et al. designed a system that would optimize the design of piezoelectric energy harvesters using simulations. The system included a NN which was trained by a GA to optimize the prediction of the optimal design. Technology is becoming more and more advanced, and in so-called advanced technologies, standard cell layouts are done manually for the design of their nodes. Ren and Fojtik [93] proposed a system using reinforcement learning in combination with a GA to help predict the optimal layout of the standard cells. Materials that are for the absorption of microwaves are important to technologies related to stealth. Huang et al. [57] proposed a large mutation GA to optimize the honeycomb meta-structure to have absorption coverage over broad-band microwaves as well as have successful mechanical resistance. The deployment of new technology is also very important, trying to efficiently upgrade a system with minimal downtime. Ko [65] designed a system using reinforcement learning hybridized with a GA to create a system to deploy a new wireless charging tram system optimally.

18.3 Discussion

EML has experienced a significant upward trend and facilitates fundamental research in various science and engineering fields, including physics, chemistry, astronomy, biology, geography, and engineering. The technology has proven invaluable in the development of machine learning algorithms that can evolve and adapt in response to changing data and environmental conditions, enabling researchers to gain a deeper understanding of complex systems. To illustrate the scope and impact of EML-related research, we have included a summary of related works in Fig. 18.1. We hope this provides a glimpse into the exciting possibilities that EML can offer to researchers across various disciplines.

Despite the promising advancements in EML, there are still several challenges that need to be addressed. One of the key challenges is the optimization of EML algorithms, which requires the use of efficient search techniques to identify the best solutions in large and complex search spaces. Another challenge is the interpretability of EML models, which can be difficult due to the complex and nonlinear relationships between variables. Additionally, EML requires significant computational resources, including high-performance computing and large-scale data storage, which can be costly and difficult to maintain. Furthermore, EML models may suffer from overfitting or underfitting, leading to poor generalization and limited applicability in real-world scenarios.

Challenges also represent opportunities, EML can also be applied to a wide range of domains, including finance, business, healthcare, and others. In these domains, EML can be used to develop predictive models that can inform decision-making and improve efficiency. Additionally, EML can be used for optimization tasks, such as scheduling and resource allocation, to improve performance and reduce costs. Furthermore, EML can be used in combination with other advanced deep learning techniques, such as transformer and diffusion models, to improve model performance and achieve more accurate predictions. By leveraging the strengths of both approaches, researchers can develop more robust and effective algorithms and models.

18.4 Conclusion

The combination of evolutionary computation and ML has unique advantages in solving challenging modeling and design problems arising in diverse fields of sciences and engineering. This is due to the powerful gradient-free evolutionary search and the strong modeling capability of ML and especially deep neural network models. In this chapter, we have shown that the EML paradigm has been widely applied in physics, materials science, chemistry, astronomy, life sciences, geology, and engineering, ranging from black-box modeling, analytical modeling, data-driven

differential equation learning, inverse design, hyperparameter optimization, training of deep neural network models, multi-objective optimization and design using ML surrogate models.

18.5 Contribution

Conceptualization, J.H.; resources, J.H.; writing—original draft preparation, J.H., Y.S., L.W., S.O., R.D., S.G.; writing—review and editing, J.H.; visualization, J.H. and Y.S.; supervision, J.H.; funding acquisition, J.H.

Acknowledgements The research reported in this work was supported in part by National Science Foundation under the grant 2110033, 1940099, and 1905775. The views, perspectives, and content do not necessarily represent the official views of the NSF. We appreciate the help of Dylan Johnson for proofreading the manuscript.

References

1. Abdel-Hamid, H., Notni, P.: Stellar population analysis from broad-band colours. *Astron. Nach.* **321**(5–6), 307–314 (2000)
2. Abouchekeir, S., Tchagang, A., Li, Y.: Adversarial deep evolutionary learning for drug design. In: 2021 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB), pp 01–09. IEEE (2021)
3. Adam-Bourdarios, C., Cowan, G., Germain, C., Guyon, I., Kégl, B., Rousseau, D.: The higgs boson machine learning challenge. In: NIPS 2014 Workshop on High-Energy Physics and Machine Learning, pp. 19–55. PMLR (2015)
4. Al-Yaseen, W.L., Idrees, A.K., Almasoudy, F.H.: Wrapper feature selection method based differential evolution and extreme learning machine for intrusion detection system. *Pattern Recogn.* **132**, 108912 (2022)
5. Alderete, N.A., Pathak, N., Espinosa, H.D.: Machine learning assisted design of shape-programmable 3d kirigami metamaterials. *NPJ Comput. Mater.* **8**(1), 1–12 (2022)
6. Ali, W., Ahmed, A.A.: Hybrid intelligent phishing website prediction using deep neural networks with genetic algorithm-based feature selection and weighting. *IET Inf. Secur.* **13**(6), 659–669 (2019)
7. Amiri, B., Gómez-Orellana, A.M., Gutiérrez, P.A., Dizène, R., Hervás-Martínez, C., Dahmani, K.: A novel approach for global solar irradiation forecasting on tilted plane using hybrid evolutionary neural networks. *J. Clean. Prod.* **287**, 125577 (2021)
8. Artrith, N., Urban, A., Ceder, G.: Constructing first-principles phase diagrams of amorphous li x si using machine-learning-assisted sampling with an evolutionary algorithm. *J. Chem. Phys.* **148**(24), 241711 (2018)
9. Attia, A.-F., Ismail, H.A., Selim, I.M., Osman, A.M., Isaa, I.A., Marie, M.A., Shaker, A.A.: Stellar population analysis of galaxies based on genetic algorithms. *Chinese J. Astron. Astrophys.* **5**(4), 347 (2005)
10. Azimi, Y., Khoshrou, S.H., Osanloo, M.: Prediction of blast induced ground vibration (bigv) of quarry mining using hybrid genetic algorithm optimized artificial neural network. *Measurement* **147**, 106874 (2019)

11. Bacardit, J., Stout, M., Hirst, J.D., Sastry, K., Llora, X., Krasnogor, N.: Automated alphabet reduction method with evolutionary algorithms for protein structure prediction. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, pp. 346–353 (2007)
12. Baes, M., Verstappen, J., De Looze, I., Fritz, J., Saftly, W., Pérez, E.V., Stalevski, M., Valcke, S.: Efficient three-dimensional nltc dust radiative transfer with skirt. *Astrophys J. Suppl. Ser.* **196**(2), 22 (2011)
13. Bagheri, S., Nan, W., Filizadeh, S.: Application of artificial intelligence and evolutionary algorithms in simulation-based optimal design of a piezoelectric energy harvester. *Smart Mater. Struct.* **29**(10), 105004 (2020)
14. Baier, A., Kerschbaum, F., Lebzelter, T.: Fitting of dust spectra with genetic algorithms-i. perspectives and limitations. *Astron. Astrophys.* **516**, A45 (2010)
15. Baldominos, A., Saez, Y., Isasi, P.: On the automated, evolutionary design of neural networks: past, present, and future. *Neural Comput. Appl.* **32**(2), 519–545 (2020)
16. Balkaya, Ç., Ekinci, Y.L., Göktürkler, G., Turan, S.: 3d non-linear inversion of magnetic anomalies caused by prismatic bodies using differential evolution algorithm. *J Appl. Geophys.* **136**, 372–386 (2017)
17. Behler, J.: Four generations of high-dimensional neural network potentials. *Chem. Rev.* **121**(16), 10037–10072 (2021)
18. Bogdanos, C., Nesseris, S.: Genetic algorithms and supernovae type ia analysis. *J. Cosmol. Astropart. Phys.* **2009**(05), 006 (2009)
19. Brasil, C.R.S., Delbem, A.C.B., da Silva, F.L.B.: Multiobjective evolutionary algorithm with many tables for purely ab initio protein structure prediction. *J. Comput. Chem.* **34**(20), 1719–1734 (2013)
20. Cai, B., Pan, G.-L., Fu, F.: Prediction of the post-fire flexural capacity of rc beam using ga-bpnn machine learning. *J Perform Const Facil* (2020)
21. Cantó, J., Curiel, S., Martínez-Gómez, E.: A simple algorithm for optimization and model fitting: Aga (asexual genetic algorithm). *Astron. Astrophys.* **501**(3), 1259–1268 (2009)
22. Carr, B., Hart, W., Krasnogor, N., Hirst, J., Burke, E., Smith, J.: Alignment of protein structures with a memetic evolutionary algorithm. In: Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation, pp. 1027–1034 (2002)
23. Cassisi, S., Schlattl, H., Salaris, M., Weiss, A.: First full evolutionary computation of the helium flash-induced mixing in population II stars. *Astrophys. J.* **582**(1), L43 (2002)
24. Chan, C., Aleti, A., Heger, A., Smith-Miles, K.: Evolving stellar models to find the origins of our galaxy. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1129–1137 (2019)
25. Charbonneau, P.: Genetic algorithms in astronomy and astrophysics. *Astrophys. J. Suppl. Ser.* **101**, 309 (1995)
26. Chen, A., Choo, K., Astrakhantsev, N., Neupert, T.: Neural network evolution strategy for solving quantum sign structures. *Phys. Rev. Res.* **4**(2), L022026 (2022)
27. Chen, B., Wang, T., Li, C., Dai, H., Song, L.: Molecule optimization by explainable evolution. In: International Conference on Learning Representation (ICLR) (2021)
28. Chen, C., Ong, S.P.: A universal graph deep learning interatomic potential for the periodic table (2022). [arXiv:2202.02450](https://arxiv.org/abs/2202.02450)
29. Chen, J., Chen, Y., Xianchen, X., Zhou, W., Huang, G.: A physics-guided machine learning for multifunctional wave control in active metabeams. *Extrem. Mech. Lett.* **55**, 101827 (2022)
30. Chen, W., Hasanipanah, M., Rad, H.N., Armaghani, D.J., Tahir, M.M.: A new design of evolutionary hybrid optimization of svr model in predicting the blast-induced ground vibration. *Eng. Comput.* **37**
31. Cheng, G., Gong, X.-G., Yin, W.-J.: Crystal structure prediction by combining graph network and optimization algorithm. *Nat. Commun.* **13**(1), 1–8 (2022)
32. Comin, A., Hartschuh, A.: Efficient optimization of shg hotspot switching in plasmonic nanoantennas using phase-shaped laser pulses controlled by neural networks. *Opt. Exp.* **26**(26), 33678–33686 (2018)

33. de França, F.O., Aldeia, G.S.I.: Interaction-transformation evolutionary algorithm for symbolic regression. *Evol. Comput.* **29**(3), 367–390 (2021)
34. De Geyter, G., Baes, M., Fritz, J., Camps, P.: Fitskirt: genetic algorithms to automatically fit dusty galaxies with a monte carlo radiative transfer code. *Astron. Astrophys.* **550**, A74 (2013)
35. Demetriou, D., See, L., Stillwell, J.: A spatial genetic algorithm for automating land partitioning. *International Journal of Geographical Information Science* **27**(12), 2391–2409 (2013)
36. Demetriou, D., Stillwell, J., See, L.M.: LandParcelS: A Module for Automated Land Partitioning. University of Leeds, School of Geography (2012)
37. Deringer, V.L., Caro, M.A., Csányi, G.: Machine learning interatomic potentials as emerging tools for materials science. *Adv. Mater.* **31**(46), 1902765 (2019)
38. Desai, S., Strachan, A.: Parsimonious neural networks learn interpretable physical laws. *Sci. Rep.* **11**(1), 1–9 (2021)
39. Durrant, J.D., Amaro, R.E., McCammon, J.A.: Autogrow: a novel algorithm for protein inhibitor design. *Chem. Biol. Drug Des.* **73**(2), 168–178 (2009)
40. Dushatskiy, A., Alderliesten, T., Bosman, P.A.N.: A novel surrogate-assisted evolutionary algorithm applied to partition-based ensemble learning. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 583–591 (2021)
41. Evans, B., Al-Sahaf, H., Xue, B., Zhang, M.: Evolutionary deep learning: a genetic programming approach to image classification. In: 2018 IEEE Congress on Evolutionary Computation (CEC), pp 1–6. IEEE (2018)
42. Faris, H., Ala'M, A.-Z., Heidari, A.A., Aljarah, I., Mafarja, M., Hassonah, M.A., Fujita, H.: An intelligent system for spam detection and identification of the most relevant features based on evolutionary random weight networks. *Inf. Fus.* **48**, 67–83 (2019)
43. Fishman, E.: Bikeshare: A review of recent literature. *Trans. Rev.* **36**(1), 92–113 (2016)
44. Gao, S., Song, S., Cheng, J., Todo, Y., Zhou, M.: Incorporation of solvent effect into multi-objective evolutionary algorithm for improved protein structure prediction. *IEEE/ACM Trans. Comput. Biol. Bioinform.* **15**(4), 1365–1378 (2017)
45. Gao, X., Lee, G.M.: Moment-based rental prediction for bicycle-sharing transportation systems using a hybrid genetic algorithm and machine learning. *Comput. Ind. Eng.* **128**, 60–69 (2019)
46. Genty, G., Salmela, L., Dudley, J.M., Brunner, D., Kokhanovskiy, A., Koblsev, S., Turitsyn, S.K.: Machine learning and applications in ultrafast photonics. *Nat. Photo.* **15**(2), 91–101 (2021)
47. Ghimire, S., Deo, R.C., Downs, N.J., Raj, N.: Self-adaptive differential evolutionary extreme learning machines for long-term solar radiation prediction with remotely-sensed modis satellite and reanalysis atmospheric products in solar-rich cities. *Remote Sens. Environ.* **212**, 176–198 (2018)
48. Giuliani, M., Castelletti, A., Pianosi, F., Mason, E., Reed, P.M.: Curses, tradeoffs, and scalable management: Advancing evolutionary multiobjective direct policy search to improve water reservoir operations. *J. Water Res. Plan. Manag.* (2):04015050 (2016)
49. Guijo-Rubio, D., Durán-Rosal, A.M., Gutiérrez, P.A., Gómez-Orellana, A.M., Casanova-Mateo, C., Sanz-Justo, J., Salcedo-Sanz, S., Hervás-Martínez, C.: Evolutionary artificial neural networks for accurate solar radiation prediction. *Energy* **210**, 118374 (2020)
50. Guillén, A., Bueno, A., Carceller, J.M., Martínez-Velázquez, J.C., Rubio, G., Todero Peixoto, C.J., Sanchez-Lucas, P.: Deep learning techniques applied to the physics of extensive air showers. *Astropart. Phys.* **111**, 12–22 (2019)
51. Zahid Halim, Muhammad Nadeem Yousaf, Muhammad Waqas, Muhammad Sulaiman, Ghulam Abbas, Masroor Hussain, Iftekhar Ahmad, and Muhammad Hanif. An effective genetic algorithm-based feature selection method for intrusion detection systems. *Computers & Security*, 110:102448, 2021
52. Hegde, R.S.: Photonics inverse design: pairing deep neural networks with evolutionary algorithms. *IEEE J. Sel. Top. Quantum Electron.* **26**(1), 1–8 (2019)
53. Hosseini, S., Zade, B.M.H.: New hybrid method for attack detection using combination of evolutionary algorithms, SVM, and ANN. *Comput. Netw.* **173**, 107168 (2020)

54. Eduardo Raul Hruschka, Ricardo JGB Campello, Alex A Freitas, et al. A survey of evolutionary algorithms for clustering. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 39(2):133–155, 2009
55. Ya-Lan, H., Chen, L.: A nonlinear hybrid wind speed forecasting model using lstm network, hysteretic elm and differential evolution algorithm. *Energy Conver. Manag.* **173**, 123–142 (2018)
56. Huang, J., Zhou, M., Zhang, J., Ren, J., Vatin, N.I., Sabri, M.M.S.: The use of ga and pso in evaluating the shear strength of steel fiber reinforced concrete beams. *KSCE J. Civil Eng.* **26**(9), 3918–3931 (2022)
57. Huang, Y., Dong, W., Chen, M., Zhang, K., Fang, D.: Evolutionary optimization design of honeycomb metastructure with effective mechanical resistance and broadband microwave absorption. *Carbon* **177**, 79–89 (2021)
58. Ivezić, Ž., Elitzur, M.: Self-similarity and scaling behaviour of infrared emission from radiatively heated dust-i. theory. *Mon. Not. R. Astronom. Soc.* **287**(4), 799–811 (1997)
59. Joseph, T., Lazio, W., Cordes, J.M., Novak, J.: The genetic algorithm: searching for planets around pulsars. In: *Bulletin of the American Astronomical Society*, vol. 25, p. 1366 (1993)
60. Kang, S., Jeong, W., Hong, C., Hwang, S., Yoon, Y., Han, S.: Accelerated identification of equilibrium structures of multicomponent inorganic crystals using machine learning potentials. *NPJ Comput. Mater.* **8**(1), 1–10 (2022)
61. Khare, E., Yu, C.-H., Obeso, C.G., Milazzo, M., Kaplan, D.L., Buehler, M.J.: Discovering design principles of collagen molecular stability using a genetic algorithm, deep learning, and experimental validation. *Proc. Nat. Acad. Sci.* **119**(40), e2209524119 (2022)
62. Kılıç, F., Yılmaz, İ.H., Kaya, Ö.: Adaptive co-optimization of artificial neural networks using evolutionary algorithm for global radiation forecasting. *Renew. Energy* **171**, 176–190 (2021)
63. Kim, W., Baag, C.-E.: Rapid and accurate two-point ray tracing based on a quadratic equation of takeoff angle in layered media with constant or linearly varying velocity functions. *Bull. Seismol. Soc. Am.* **92**(6), 2251–2263 (2002)
64. Kim, W., Hahm, I.-K., Ahn, S.J., Lim, D.H.: Determining hypocentral parameters for local earthquakes in 1-d using a genetic algorithm. *Geophys. J. Int.* **166**(2), 590–600 (2006)
65. Young Dae Ko: An efficient integration of the genetic algorithm and the reinforcement learning for optimal deployment of the wireless charging electric tram system. *Comput. Ind. Eng.* **128**, 851–860 (2019)
66. Kondo, T.: Evolutionary design and behavior analysis of neuromodulatory neural networks for mobile robots control. *Appl. Soft Comput.* **7**(1), 189–202 (2007)
67. Kowalski, M., Rubin, D., Aldering, G., Agostinho, R.J., Amadon, A., Amanullah, R., Balland, C., Barbary, K., Blanc, G., Challis, P.J. et al.: Improved cosmological constraints from new, old, and combined supernova data sets. *Astrophys. J.* **686**(2), 749 (2008)
68. Kwon, Y., Kang, S., Choi, Y.-S., Kim, I.: Evolutionary design of molecules based on deep learning and a genetic algorithm. *Sci. Rep.* **11**(1), 1–11 (2021)
69. Le, T.C., Winkler, D.A.: Discovery and optimization of materials using evolutionary approaches. *Chem. Rev.* **116**(10), 6107–6132 (2016)
70. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. *Nature* **521**(7553), 436–444 (2015)
71. Lei, Z., Gao, S., Zhang, Z., Zhou, M.C., Cheng, J.: Mo4: a many-objective evolutionary algorithm for protein structure prediction. *IEEE Trans. Evol. Comput.* **26**(3), 417–430 (2021)
72. Li, W., Zhao, D., He, C., Andong, H., Zhang, K.: Advanced machine learning optimized by the genetic algorithm in ionospheric models using long-term multi-instrument observations. *Remote Sens.* **12**(5), 866 (2020)
73. Xia Li and Anthony Gar-On Yeh: Integration of genetic algorithms and GIS for optimal location search. *Int. J. Geograph. Inf. Sci.* **19**(5), 581–601 (2005)
74. Li, Y., Ooi, H.K., Tchagang, A.: Deep evolutionary learning for molecular design (2020). [arXiv:2102.01011](https://arxiv.org/abs/2102.01011)
75. Li, Y., Cheng, W., Yu, L.H., Rainer, R.: Genetic algorithm enhanced by machine learning in dynamic aperture optimization. *Phys. Rev. Accel. Beams* **21**(5), 054601 (2018)

76. Wanzenböck, R., Arrigoni, M., Bichelmaier, S., Buchner, F., Carrete, J., Madsen, G.K.H.: Neural-network-backed evolutionary search for srtio3(110) surface reconstructions. *Digit. Disc.* **8** (2022)
77. Marzouq, M., Bounoua, Z., Fadili, H.E., Mechaqrane, A., Zenkouar, K., Lakhliai, Z.: New daily global solar irradiation estimation model based on automatic selection of input parameters using evolutionary artificial neural networks. *J. Clean. Produ.* **209**, 1105–1118 (2019)
78. Mason, K., Duggan, J., Howley, E.: Forecasting energy demand, wind generation and carbon dioxide emissions in Ireland using evolutionary neural networks. *Energy* **155**, 705–720 (2018)
79. Metcalfe, T.S.: Genetic-algorithm-based light-curve optimization applied to observations of the w ursae majoris star bh cassiopeiae. *Astronom. J.* **117**(5), 2503 (1999)
80. Metcalfe, T.S., Charbonneau, P.: Stellar structure modeling using a parallel genetic algorithm for objective global optimization. *J. Comput. Phys.* **185**(1), 176–193 (2003)
81. Moayedi, H., Moatamedian, A., Nguyen, H., Bui, X.-N., Bui, D.T., Rashid, A.S.A.: Prediction of ultimate bearing capacity through various novel evolutionary and neural network models. *Eng. Comput.* **36**(2), 671–687 (2020)
82. Mokiem, M.R., de Koter, A., Puls, J., Herrero, A., Najarro, F., Villamariz, M.R.: Spectral analysis of early-type stars using a genetic algorithm based fitting method. *Astron. Astrophys.* **441**(2), 711–733 (2005)
83. Moster, B.P., Naab, T., Lindström, M., O’Leary, J.A.: Galaxynet: connecting galaxies and dark matter haloes with deep neural networks and reinforcement learning in large volumes. *Mon. Not. R. Astronom. Soc.* **507**(2), 2115–2136 (2021)
84. Nanayakkara, V.K., Ikegami, Y., Uehara, H.: Evolutionary design of dynamic neural networks for evaporator control. *Int. J. Ref.* **25**(6), 813–826 (2002)
85. Neshat, M., Nezhad, M.M., Abbasnejad, E., Mirjalili, S., Tjernberg, L.B., Garcia, D.A., Alexander, B., Wagner, M.: A deep learning-based evolutionary model for short-term wind speed forecasting: a case study of the lillgrund offshore wind farm. *Energy Convers. Manag.* **236**, 114002 (2021)
86. Nesseris, S., Shafieloo, A.: A model-independent null test on the cosmological constant. *Mon. Not. R. Astron. Soc.* **408**(3), 1879–1885 (2010)
87. Nunkesser, R., Morell, O.: An evolutionary algorithm for robust regression. *Comput. Stat. Data Anal.* **54**(12), 3242–3248 (2010)
88. Owoyele, O., Pal, P., Torreira, A.V., Probst, D., Shaxted, M., Wilde, M., Senecal, P.K.: Application of an automated machine learning-genetic algorithm (automl-ga) coupled with computational fluid dynamics simulations for rapid engine design optimization. *Int. J. Engine Res.* **23**(9), 1586–1601 (2022)
89. Paton, F.L., Maier, H.R., Dandy, G.C.: Including adaptation and mitigation responses to climate change in a multiobjective evolutionary algorithm framework for urban water supply systems incorporating ghg emissions. *Water Res. Res.* **50**(8), 6285–6304 (2014)
90. Pegg, S.C.-H., Haresco, J.J., Kuntz, I.D.: A genetic algorithm for structure-based de novo design. *J. Comput.-Aided Mol. Des.* **15**(10), 911–933 (2001)
91. Podryabininkin, E.V., Tikhonov, A.V., Shapeev, A.V., Oganov, A.R.: Accelerating crystal structure prediction by machine-learning interatomic potentials with active learning. *Phys. Rev. B* **99**(6), 064114 (2019)
92. Puls, J., Urbaneja, M.A., Venero, R., Repolust, T., Springmann, U., Jokuthy, A., Mokiem, M.R.: Atmospheric NLTE-models for the spectroscopic analysis of blue stars with winds-II. line-blanketed models. *Astron. Astrophys.* **435**(2), 669–698 (2005)
93. Ren, H., Fojtik, M.: Standard cell routing with reinforcement learning and genetic algorithm in advanced technology nodes. In: Proceedings of the 26th Asia and South Pacific Design Automation Conference, pp. 684–689 (2021)
94. Rolnick, D., Donti, P.L., Kaack, L.H., Kochanski, K., Lacoste, A., Sankaran, K., Ross, A.S., Milojevic-Dupont, N., Jaques, N., Waldman-Brown, A. et al.: Tackling climate change with machine learning. *ACM Comput. Surv. (CSUR)* **55**(2), 1–96 (2022)
95. Ruiz, A.N., Cora, S.A., Padilla, N.D., Domínguez, M.J., Vega-Martínez, C.A., Tecce, T.E., Orsi, Á., Yaryura, Y., Lambas, D.G., Gargiulo, I.D. et al.: Calibration of semi-analytic models of galaxy formation using particle swarm optimization. *Astrophys. J.* **801**(2), 139 (2015)

96. Ruuzek, B., Kvasnicka, M.: Differential evolution algorithm in the earthquake hypocenter location. *Pure Appl. Geophys.* **158**(4), 667–693 (2001)
97. Safaei, M., Rezayan, H., Firouzabadi, P.Z., Sadidi, J.: Optimization of species distribution models using a genetic algorithm for simulating climate change effects on zagros forests in Iran. *Ecol. Inf.* **63**, 101288 (2021)
98. Şahin, C.B., Dinler, Ö.B., Abualigah, L.: Prediction of software vulnerability based deep symbiotic genetic algorithms: phenotyping of dominant-features. *Appl. Intell.* **51**(11), 8271–8287 (2021)
99. Sahni, V., Starobinsky, A.: Reconstructing dark energy. *Int. J. Mod. Phys. D* **15**(12), 2105–2132 (2006)
100. Salazar, J.Z., Reed, P.M., Herman, J.D., Giuliani, M., Castelletti, A.: A diagnostic assessment of evolutionary algorithms for multi-objective surface water reservoir control. *Adv. Water Res.* **92**, 172–185 (2016)
101. Salcedo-Sanz, S., Cornejo-Bueno, L., Prieto, L., Paredes, D., García-Herrera, R.: Feature selection in machine learning prediction systems for renewable energy applications. *Renew. Sust. Energy Rev.* **90**, 728–741 (2018)
102. Sambridge, M., Gallagher, K.: Earthquake hypocenter location using genetic algorithms. *Bull. Seismol. Soc. Am.* **83**(5), 1467–1491 (1993)
103. Schlattl, H., Cassisi, S., Salaris, M., Weiss, A.: On the helium flash in low-mass population III red giant stars. *Astrophys. J.* **559**(2), 1082 (2001)
104. Seyedzadeh, S., Rahimian, Oliver, F.P.S., Rodriguez, S., Glesk, I.: Machine learning modelling for predicting non-domestic buildings energy performance: a model to support deep energy retrofit decision-making. *Appl. Energy* **279**, 115908 (2020)
105. Sonnenberg, J. et al.: Fundamentals of land consolidation as an instrument to abolish fragmentation of agricultural holdings (2002)
106. Soto, D., Soto, W.: Evolutionary algorithm for solving supervised classification problems: an experimental study. In: 2022 6th International Conference on Intelligent Systems, Meta-heuristics and Swarm Intelligence, pp. 24–29 (2022)
107. Y. Sun, Y. Jiao, C. Shi, Y. Zhang, Deep learning-based molecular dynamics simulation for structure-based drug design against sars-cov-2. *Comput. Struct. Biotechnol. J.* (2022)
108. Szustakowski, J.D., Weng, Z.: Protein structure alignment using a genetic algorithm. *Prot.: Struct. Funct. Bioinf.* **38**(4), 428–440 (2000)
109. Tani, L., Rand, D., Veelken, C., Kadastik, M.: Evolutionary algorithms for hyperparameter optimization in machine learning for application in high energy physics. *European Phys. J. C* **81**(2), 1–9 (2021)
110. Theis, C.H., Kohle, S.: Multi-method-modeling of interacting galaxies-I. A unique scenario for NGC 4449? *Astron. Astrophys.* **370**(2), 365–383 (2001)
111. Ünal, H.T., Başçıçı, F.: Evolutionary design of neural network architectures: a review of three decades of research. *Artif. Intell. Rev.* 1–80 (2021)
112. Wahde, M.: Determination of orbital parameters of interacting galaxies using a genetic algorithm-description of the method and application to artificial data. *Astron. Astrophys. Suppl. Ser.* **132**(3), 417–429 (1998)
113. Wahde, M., Donner, K.J.: Determination of the orbital parameters of the m 51 system using a genetic algorithm. *Astron. Astrophys.* **379**(1), 115–124 (2001)
114. Wall, M.: Galib: a c++ library of genetic algorithm components. *Mech. Eng. Dep. MA Inst. Technol.* **87**, 54 (1996)
115. Wang, W., Tang, R., Li, C., Liu, P., Luo, L.: A BP neural network model optimized by mind evolutionary algorithm for predicting the ocean wave heights. *Ocean Eng.* **162**, 98–107 (2018)
116. Wang, Y., Chengru, W., Zhao, S., Wang, J., Bingfeng, Z., Han, M., Qing, D., Ni, M., Jiao, K.: Coupling deep learning and multi-objective genetic algorithms to achieve high performance and durability of direct internal reforming solid oxide fuel cell. *Appl. Energy* **315**, 119046 (2022)
117. Wierzbinski, M., Pławiak, P., Hammad, M., Acharya, U.R.: Development of accurate classification of heavenly bodies using novel machine learning techniques. *Soft Comput.* **25**(10), 7213–7228 (2021)

118. Wu, Z., Kan, S.B.J., Lewis, R.D., Wittmann, B.J., Arnold, F.H.: Machine learning-assisted directed protein evolution with combinatorial libraries. *Proc. Nat. Acad. Sci.* **116**(18), 8852–8858 (2019)
119. Hao, X., Chang, H., Zhang, D.: DLGA-PDE: Discovery of PDEs with incomplete candidate library via combination of deep learning and genetic algorithm. *J. Comput. Phys.* **418**, 109584 (2020)
120. Hao, X., Zhang, D.: Robust discovery of partial differential equations in complex situations. *Phys. Rev. Res.* **3**(3), 033270 (2021)
121. Hao, X., Zhang, D., Zeng, J.: Deep-learning of parametric partial differential equations from sparse and noisy data. *Phys. Fluids* **33**(3), 037132 (2021)
122. Haoran, X., Ma, J., Tan, P., Chen, B., Zhen, W., Zhang, Y., Wang, H., Xuan, J., Ni, M.: Towards online optimisation of solid oxide fuel cell performance: combining deep learning with multi-physics simulation. *Energy AI* **1**, 100003 (2020)
123. Yaseen, Z.M., Afan, H.A., Tran, M.-T.: Beam-column joint shear prediction using hybridized deep learning neural network with genetic algorithm. In: IOP Conference Series: Earth and Environmental Science, vol. 143, p. 012025. IOP Publishing (2018)
124. Yoon, E.J., Thorne, J.H., Park, C., Lee, D.K., Kim, K.S., Yoon, H., Seo, C., Lim, C.-H., Kim, H., Song, Y.-I.: Modeling spatial climate change landuse adaptation with multi-objective genetic algorithms to improve resilience for rice yield and species richness and to mitigate disaster risk. *Environ. Res. Lett.* **14**(2), 024001 (2019)
125. Yoshikawa, N., Terayama, K., Sumita, M., Homma, T., Oono, K., Tsuda, K.: Population-based de novo molecule generation, using grammatical evolution. *Chem. Lett.* **47**(11), 1431–1434 (2018)
126. Zhan, Z.-H., Li, J.-Y., Zhang, J.: Evolutionary deep learning: a survey. *Neurocomputing* **483**, 42–58 (2022)
127. Zhang, C.-X., Yuan, Y., Zhang, H.-W., Shuai, Y., Tan, H.-P.: Estimating stellar effective temperatures and detected angular parameters using stochastic particle swarm optimization. *Res. Astron. Astrophys.* **16**(9), 008 (2016)
128. Zhong, X., Velez, C., Acevedo, O.: Partial charges optimized by genetic algorithms for deep eutectic solvent simulations. *J. Chem. Theory Comput.* **17**(5), 3078–3087 (2021)
129. Zhou, Y., Zhou, N., Gong, L., Jiang, M.: Prediction of photovoltaic power output based on similar day analysis, genetic algorithm and extreme learning machine. *Energy* **204**, 117894 (2020)
130. Zuo, Y., Qin, M., Chen, C., Ye, W., Li, X., Luo, J., Ong, S.P.: Accelerating materials discovery with bayesian optimization and graph deep learning. *Mater. Today* **51**, 126–135 (2021)

Chapter 19

Evolutionary Machine Learning in Environmental Science



João E. Batista and Sara Silva

Abstract This chapter reviews the use of Evolutionary Machine Learning (EML) in environmental science. We cover the various steps of the machine learning pipeline, also addressing topics like model robustness, interpretability, and human-competitiveness. Environmental science is an interdisciplinary field mainly dedicated to climate change, natural resource management, conservation biology, and sustainability. We review applications such as forest monitoring, optimization of photovoltaic installations, improvement of traffic flow, and reduction of waste in animal farms, among others.

19.1 Introduction

This chapter covers Evolutionary Machine Learning (EML) methods for data preparation, feature engineering, regression and classification, among other common tasks in environmental science. While going through the different steps of the machine learning pipeline, we also address the robustness, interpretability, and human-competitiveness of the models obtained with EML methods. We follow the definitions proposed in [103], according to which an “interpretable” model can be understood by domain experts without requiring explanation algorithms, thanks to its size, operators, and number of used features. By opposition, some models are not inherently interpretable but can be “explained” using other algorithms.

EML adds flexibility to machine learning algorithms that are already highly versatile. Therefore, it is not surprising that EML is being applied to an endless list of tasks in various scientific domains. In environmental science, however, EML is still underused, with many experts still preferring other methods like random forests and Artificial Neural Networks (ANNs). Nowadays, ANNs are being used in several scientific domains but, although these models provide good results, they are not

J. E. Batista · S. Silva (✉)

LASIGE, Department of Informatics, Faculty of Sciences, University of Lisbon,
1749-016 Lisboa, Portugal
e-mail: sgsilva@fc.ul.pt

interpretable. EML-based models, on the other hand, are generally known for their potential interpretability [11].

Following the description by Miller et al. [83], environmental science is a vast interdisciplinary field that integrates knowledge from several fields such as biology, chemistry, geology, geography, economics, political science, philosophy, and ethics. Its major goal is to understand how nature works, how we interact with the environment, and how to deal with environmental problems and live sustainably. Machine learning is increasingly recognized as a strong ally in tackling environmental issues [101].

In this chapter, we review several examples of EML being successfully applied to different tasks in environmental science. We promote EML by surveying the application of methods based on Genetic Algorithms (GA) [52] for data cleaning; methods based on GA, Ant Colony Optimization (ACO) [57], Genetic Programming (GP) [96], Multi-Objective Evolutionary Algorithms (MOEA) [34], and Particle Swarm Optimization (PSO) [39] for feature selection; methods such as Evolutionary Feature Synthesis (EFS) [7], Feature Engineering Automation Tool (FEAT) [65] and Evolutionary Polynomial Regression (EPR) [46], and methods based on GA and GP, for feature construction; methods based on ACO, PSO, and GA for other tasks, e.g., parameter tuning. We also promote environmental science by surveying an extensive range of applications, among which land cover-type detection, identification of probable forest fire ignition points, and prediction of ground salinity in remote sensing; prediction of solar radiation for photovoltaic installations; improvement of gas turbine performance; animal farm management to avoid waste and other applications. Although not specifically focused on EML, a recent survey [101] also includes several applications of EML in environmental science.

19.2 Data Sources and Types

This field relies on data obtained from two sources: *in situ* measurements and remote sensing. While *in situ* measurements require the sensors to be in contact with the objects under study, remote sensing can obtain measurements without any physical contact. *In situ* measurements can be made using a large variety of instruments that are highly dependent on the study case. As an example, Johari et al. [55] mention that soil characteristics can be obtained using pressure plates, Büchner funnels, tensiometers, pressure membranes, filter paper, and heat dissipation sensors. Remote sensing [30] measurements can also be obtained through different sensors, like sonars or cameras installed in boats, and different types of sensors onboard Unmanned Aerial Vehicles (UAVs) and satellites, including optical and radar sensors. Since *in situ* measurements can be performed using an endless list of instruments that vary from task to task, in Table 19.1, we provide only a list of remote sensing instruments used in our bibliography.

Both UAV and satellite sensors measure radiance in specific wavelength intervals. Each of these intervals is called a band, and each sensor may perform measurements

Table 19.1 List of remote sensing instruments used in the surveyed works

Instrument type	Instruments
Sonars	Simdar EQ60 [110]
UAV Sensors	Compact Airborne Spectrographic Imager (CASI) [29]
	Fabry-Pérot interferometer (FPI) [13]
	Hyperspectral Digital Imagery Collection Experiment (HYDICE) [126]
	Airborne Visible InfraRed Imaging Spectrometer (AVIRIS) [106]
Satellites (SAR ^a)	Sentinel-1 [43]; E-SAR, RADARSAT-2 and AIRSAR [71] ^b
Satellites (Optical)	Landsat-8 (LS8) [90] and Sentinel-2 (S2) [42]

^a Synthetic Aperture Radar (SAR)

^b Described as Polarimetric SAR (PolSAR) imagery

in several different bands. For consistency with the machine learning nomenclature, we will freely call “feature” to each of these bands. UAVs allow the acquisition of very high-resolution imagery (e.g., 10cm per pixel, using the P4 multi-spectral UAV [82]). However, this data is limited in coverage. By opposition, satellite imagery is freely available and offers continuously updated worldwide coverage, for example, with Landsat-8 (LS8) [90] and Sentinel-2 (S2) [42]. The downside of free satellite imagery is the spatial resolution, which tends to be much lower than with UAVs, ranging from 10m to 60 m per pixel on both LS8 and S2.

The data used by the authors in our bibliography can be divided into two categories: uni-temporal and multi-temporal datasets. The first category includes datasets whose measurements were obtained in a single time frame and will be referred to as “uni-temporal datasets”, or simply “datasets”. The second category includes datasets whose values are obtained using several measurements, obtained in a succession of different time frames, and will be referred to as “time series”. Although uni-temporal data usually provides good results, time series normally contains more useful information and therefore can be used to solve harder tasks. One application mentioned later is the separation of similar types of vegetation through their phenological cycles [16].

This chapter covers the use of EML-based machine learning methods in uni-temporal datasets and time series, in several different tasks. Each section of this chapter covers a different kind of application, in the following order: data preparation, feature engineering (Sect. 19.4), split into feature selection and feature construction; regression tasks (Sect. 19.5), i.e., the prediction of a numerical value for each sample, like the amount of biomass in each location; classification tasks (Sect. 19.6), split into binary and multiclass classification, i.e., prediction of the nominal label of each sample, like the land cover type for each pixel in satellite imagery; other tasks (Sect. 19.7) like image alignment and parameter tuning. Other examples of applications are then given, focusing on the robustness of the EML models (Sect. 19.8) and highlighting the human-competitiveness of results obtained in selected work by the authors (Sect. 19.9).

19.3 Data Preparation

The machine learning pipeline includes several steps that should be followed to ensure the quality of the final model. After collecting the data, the first step is data preparation. Data preparation includes tasks such as dealing with missing values, removing duplicated samples and outliers, and data normalization, among other tasks that may depend on the algorithms to be used later on, e.g., one-hot encoding to convert categorical features into numerical or boolean features.

Dealing with missing values is normally done using statistical methods. However, some authors use machine learning, and a recent survey [49] provides a list of machine learning techniques used for this task. From the EML field, only methods based on GAs are mentioned, namely, the Multi-Objective GA for data Imputation (MOGAImp) [73]. Two other works using EML for missing value imputation describe the use of GAs [44] and ANNs assisted with GAs to minimize their error function [2]. Specific to environmental science, we only find two EML works that perform missing value imputation: one uses GAs for the measurement of gas turbine blades [40] and the other is the MOGAImp method [73] applied to the classification of land cover types using satellite imagery.

In some cases, the presence of a few bad samples within a dataset can lead to overfitting. Although their work is not focused on data preprocessing, in [111] the authors propose a semi-supervised GP method that effectively detects and avoids learning mislabeled data in a dataset regarding the detection of burnt areas in satellite imagery. Such methods can also be studied from the data preparation point of view.

19.4 Feature Engineering in Environmental Science

After data preparation, feature engineering is the next step in the machine learning pipeline. Feature engineering is necessary to ensure the machine learning algorithms use high-quality data to induce their predictive models, leading to better results and less overfitting [35]. This step usually requires an expert to indicate which variables correlate with the problem, which variables we should discard and which variables we should combine (e.g., to create indices that highlight characteristics of the dataset samples). This is traditionally a manual task that requires domain-specific knowledge, therefore it is time-consuming. However, feature engineering is becoming more automated, namely, with its integration in AutoML systems like TPOT [91]. According to [72], feature engineering can be partitioned into feature selection and feature construction.¹ This section addresses both types.

¹ Frequently also called feature extraction, feature generation, feature learning, feature discovery, feature synthesis, or constructive induction.

19.4.1 Feature Selection

Feature selection deals with reducing the number of features within a dataset. The objective is to facilitate the learning of the machine learning algorithm or to reduce the likelihood of overfitting by removing redundant or noisy features from the dataset [72]. Although this selection is usually made by experts using their domain-specific knowledge, many methods are being developed to perform this task automatically.

Over the years, many EML algorithms have been used to perform feature selection. GAs [52] are probably the oldest subtype of EML to be used for feature selection [68] and continue to be a very popular choice nowadays. There are other popular EML methods for feature selection, such as the ACO [57], GP [96], MOEA, and PSO [39] algorithms.

As a practical example of feature selection, consider a GA in a hypothetical dataset with 10 features, where eight are noise and two are necessary to reach a good solution. Following the illustration in Fig. 19.1, the GA randomly generates a population of possible solutions to the problem in the form of arrays of “allowed features” to be used by a decision tree and uses their accuracy on a validation set as fitness. Over the generations, the noisy features are dropped out of the population while the good features survive. In a perfect scenario, the final model uses all the good features and none of the noisy ones.

We now focus on methods created and studied to perform feature selection in environmental science tasks. In remote sensing aerial imagery, feature selection is commonly applied to select relevant bands in an image out of dozens or hundreds of bands. In [29], the authors use the CASI imager to generate high-resolution images using 72 spectral bands from 409 to 947nm. Then, the authors compare the feature selection performed by GP-SVI (their proposed GP-based method) and GA-PLS (a GA-based method) with traditional statistical methods, concluding that GP-SVI outperforms the traditional methods and that GA-PLS, although not outperforming the traditional methods, has a very low standard deviation on the test results. Similarly, in [51], the authors study three datasets obtained from images with a large number of

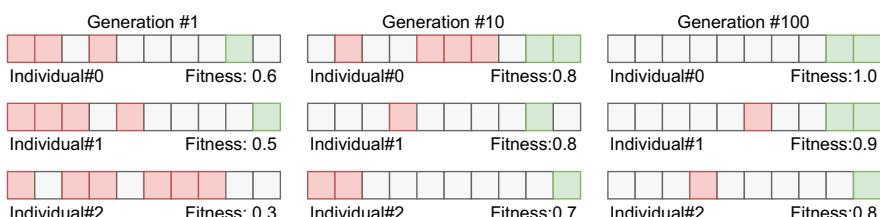


Fig. 19.1 Example of feature selection using a GA. Using a fitness function that promotes selecting the good (green) features while avoiding the noisy (red) features, the GA learns which features are useful for the problem over the generations. A possible fitness function is the performance of a learning algorithm using only the selected features, e.g., validation accuracy of a decision tree

bands (115, 220, and 224 bands). They compare HC-ABC (their proposed method, Hypergraph Clustering Artificial Bee Colony) with four non-EML algorithms and two EML algorithms and conclude that their method outperforms the other six in all three datasets. The PSO algorithm is used in [13] to select 15 bands out of the 380 bands available using the FPI sensor, and in [126] to select 7 bands out of 180 using the HYDICE sensor. Lastly, in [106], the authors compare 10 different EML algorithms for feature selection in three images obtained from the AVIRIS sensor with 176, 200, and 204 bands.

Overall, this illustrates the potential of EML-based algorithms for feature selection in datasets with a large number of features. While these works are specific for datasets obtained using UAV imagery, datasets with many more features exist, and works from other areas show that EML-based algorithms can find optimal features in datasets with numbers of features that range from one thousand to over one million [22, 66, 94, 99, 118, 122].

Unlike UAV imagery, satellite imagery from the most popular satellites has a smaller number of bands. For example, the LS8 and S2 satellites only use 11 and 13 bands, respectively. Since satellites usually use a smaller number of bands, the gap between the wavelengths measured by each band tends to be bigger. This results in satellite bands having well-known and different meanings, as explained in detail in [90]. As described in this web page about the LS8 satellite, healthy vegetation is very reflective in Near-Infrared (NIR) wavelengths (band 5), so this band is very important to study vegetation. Shortwave Infrared (SWIR, bands 6 and 7) bands are very useful to study the soil, and band 7 allows an easier detection of burnt areas, as illustrated in Fig. 19.2. Although satellite imagery datasets usually have a small number of features, it is useful to have feature selection methods that remove unnecessary bands from the dataset since this usually leads to the creation of simpler machine learning models.

In [56], the authors build an improved version of the Normalized Difference Vegetation Index (NDVI) [102] using a GA to evolve weights in a mathematical expression that uses several bands. After obtaining the final weights, the authors removed the bands that had little impact on the expression (i.e., the bands associ-

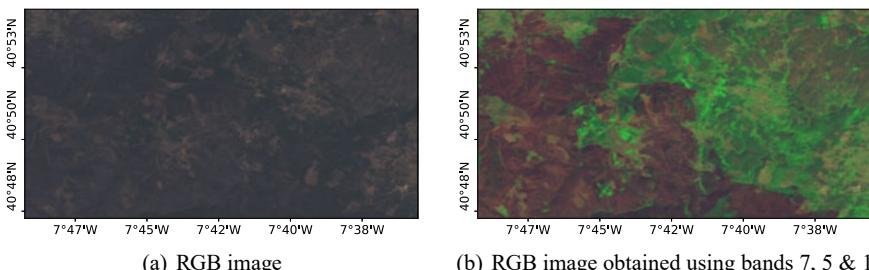


Fig. 19.2 Landsat-8 satellite imagery over Guarda (Portugal) in September 2022. The image in (b) uses band 7 to highlight burnt areas (brown) and band 5 to highlight healthy vegetation (green), facilitating the detection of burnt areas

ated with very small weights). In [71], the authors propose SAE-MOEA/D, a new MOEA algorithm with a Stacked AutoEncoder (SAE), to select bands to be used by convolutional neural networks (CNN) [47] in PolSAR imagery.

Some methods solve tasks by evolving models that do not use all the available features. Since feature selection is a collateral result of their primary objective, we consider this “implicit” feature selection. Since some GP-based algorithms evolved models that only use a subset of the features, this implicit feature selection is a frequent side effect of their evolution. The following works perform implicit feature selection, although their main objective is feature construction. As such, we will explain them in greater detail in Sect. 19.4.2.

In [16, 18, 82], the authors use GP-based methods and comment on the features selected for the final models. In [82], the authors report that the red and blue wavelengths seem to be highly preferred by the models when detecting mistletoe, while the red-edge infrared and NIR wavelengths seem to be avoided. In [16], the authors use time series to separate two similar land cover types using data from several months and comment that the features of August are frequently selected by the models, indicating that this month is important in the separation of the two land cover types. In [18], the authors evolve features that detect water in cloudy imagery and comment that, besides the models selecting the same features as remote sensing experts do to detect water, they also use the features that are associated with detecting clouds. Other works also use GP-based algorithms for classification [15, 17, 76] and regression [55, 93, 97, 115] tasks. However, the authors do not give emphasis on the feature selection side of these algorithms.

These algorithms can also be applied to feature selection in datasets that are not based on remote sensing. In [127], the authors apply the Standard GP (Std-GP) [96] algorithm to a regression dataset to predict fish weights based on the measurements of the fish. Then, the authors comment on the use frequency of each feature for each fish species, indicating that the weight of each fish species is correlated with the measurements of different body parts. In [38], the authors propose the evolutionary feature selection algorithm and compare this method’s accuracy and number of selected features with other EML-based algorithms in a benchmark that, among other non-EML datasets, include a dataset for the classification of animal types based on their physical characteristics. Although this work is not focused on conservation biology, this dataset may help to study habitats in future.

19.4.2 Feature Construction

Feature construction combines or modifies features to create more informative ones [72]. Ideally, these new features are easier to handle by machine learning algorithms and, assuming their interpretability, also by human experts. However, due to the large number of possible combinations of features and the computational cost of testing all combinations, this task is usually performed manually by experts. In this section, we talk about EML for automatic feature construction.

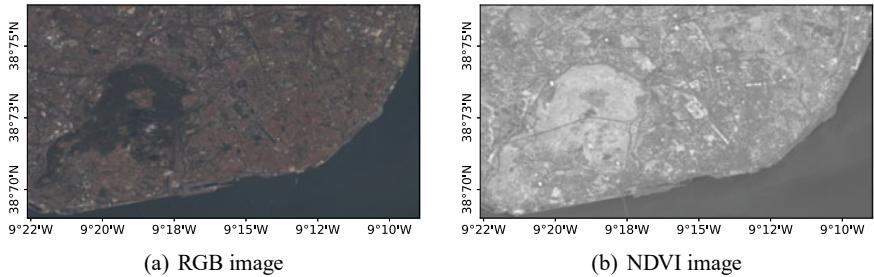


Fig. 19.3 Landsat-8 satellite imagery, obtained over Lisbon (Portugal) in September 2022. The NDVI in (b) is used to highlight the vegetation within the image

Domain experts can use their knowledge to perform manual feature construction, selecting features that are considered relevant to the problem and combining them. For example, in the remote sensing field, experts know that some ratios between certain wavelengths are informative for specific problems. Those ratios can be added to the datasets, thus facilitating learning. The NDVI [102], displayed in Eq. 19.1, is an example of a popular ratio. This popular index highlights healthy vegetation in remote sensing data, as exemplified in Fig. 19.3.

$$NDVI = \frac{NIR - Red}{NIR + Red} \quad (19.1)$$

Although feature construction is traditionally a manual task, over the years, many evolutionary computation methods have been developed to perform automatic feature construction (e.g., EFS [7] and FEAT [65]). Depending on the feature construction algorithm and the complexity of the problem, the evolved set of features varies in the number of features, their complexity, and the number of original features used in each new feature's expression. This leads to some variance in the interpretability of the final models. Some machine learning algorithms, namely those based on deep learning [47], are known not only for their state-of-the-art results but also for their lack of model interpretability, despite some visualization methods [128] and other recent efforts [120, 129] that allow them to be explained. By opposition, EML algorithms are known for their potential interpretability [11, 103]. It should be noted that some EML algorithms, like the ones of neuroevolution, evolve models that are not considered interpretable. Some works may also use algorithms known for providing interpretable models and fail to obtain them due to, e.g., the complexity of the problems or the use of sub-optimal parameters.

As a practical example of feature construction, consider the Std-GP algorithm in a hypothetical dataset where the optimal solution is the sum of two specific features and the other features are noise. Following the illustration in Fig. 19.4, the Std-GP randomly generates a population of possible solutions in the form of mathematical expressions that convert a dataset sample into a single numerical value. Over the generations, the features selected by the model (implicit feature selection) and the

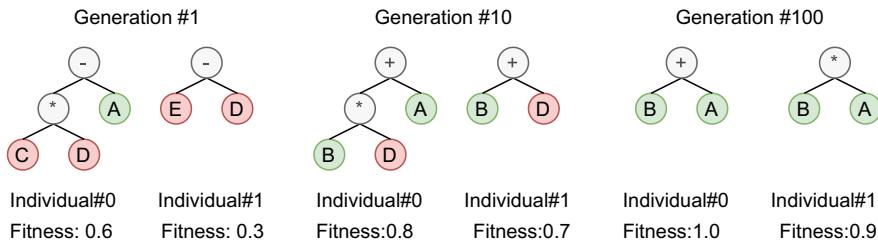


Fig. 19.4 Example of feature construction using Std-GP. Here, the optimal solution is $A+B$. Using a fitness function that promotes the search for this expression (e.g., accuracy), the population evolves models that select the correct features (implicit feature selection) while using the correct operators

operators and shape of the model (feature construction) will change while optimizing the fitness function. In this case, the fitness function can be the correlation between the model's output and the prediction, making a model applicable to regression tasks and binary classification (e.g., by converting class labels to numerical values 0 and 1). More complex methods, like M3GP [85], evolve a set of trees, resulting in multidimensional model output. Similarly to the GA in the previous section's example, these models may use a second learning algorithm to calculate their fitness. Due to the multidimensionality of the output, these models can also be used for feature construction in multiclass datasets.

The typical way to perform feature construction involves the creation of features and adding them to the dataset, as seen in several works that add manually created features to the dataset [15, 95]. Although some works replace the original features with those evolved by the machine learning algorithms [16, 17, 75], other works suggest that replacing the original features might not always be the best option. Instead, these works add the evolved features to the dataset [15]. However, depending on the classifier, this approach should be avoided since it increases the dimensionality of the dataset, possibly leading to a higher computational cost when training other models in the extended dataset.

The following works dealt with the evolution of features whose expression is small enough to be considered easily interpretable. In [31], the authors use a GA to evolve weights for an arithmetic expression that combines three features and three weights, with the objective of mimicking the NDVI using the RGB values. This allows the use of cameras without infrared sensors in the study of healthy vegetation. Similarly, in [56], the authors create an improved version of the NDVI by evolving a set of 12 weights and, after training the model, remove the features associated with small weights, simplifying the expression evolved by the algorithm.

The previous works use GA-based algorithms to evolve features with a fixed structure predefined by authors. In opposition, GP-based algorithms do not require a predefined structure for the models since they are mutable and adapted over the evolutionary cycle. Even among GP-based algorithms, the structure of the models can be decomposed into two types: models composed of a single tree and models composed of multiple trees. While the first approach is usually far more limited, it

can be applied to regression and binary classification tasks in a straightforward way, while multiclass classification tasks usually require multiple features, i.e., a model with multiple trees/outputs.

Typically, the authors use GP-based algorithms, namely the Std-GP algorithm, to evolve models composed of a single tree, both for regression tasks [55, 93, 97, 113, 115, 127] and for binary classification [18, 21, 76, 123]. Other applications of GP-based methods in remote sensing can also be seen in [67]. Unlike Std-GP, other EML-based feature construction methods, such as the already mentioned EFS [7], EPR [46] and M3GP [85] algorithms, evolve a set of trees, rather than a single tree, resulting in a multidimensional output from the model. Although these algorithms show potential in creating interpretable models, this interpretability may depend on the number of trees and the mathematical operators used. In [81], the authors review several works related to feature construction using GP-based methods in the context of explainable artificial intelligence. Those works have a wide range of application fields, including environmental science.

For more complex tasks, such as multiclass classification and harder binary classification and regression tasks, the authors typically use methods that evolve several features. These features can work independently from each other or in a cooperative manner. In works that treat multiclass classification as a succession of binary classification problems [110], we consider that the features are independent, since each feature separates a different set of classes. In more advanced multiclass classification methods, such as the M3GP [85] and M4GP [66], and the Std-GP algorithm with COMB functions [75], the algorithms evolve a set of cooperative features (a transformation) that converts the feature space, and then perform multiclass classification in this new space, separating all classes using the same transformation. Another method derived from M3GP, called eM3GP [109], assumes that an ensemble of transformations, each proving to be good for discriminating a single class, may do a better job than a single transformation used for discriminating all the classes. In [87], a set of features is evolved to be jointly used in a single linear regression model.

19.5 Regression Tasks for Environmental Science

In regression tasks, the machine learning methods attempt to predict a numerical value to be associated with each sample in a dataset. Taking the NDVI [102] as an example, this index associates each pixel in an image with a value from -1 to 1 , where a higher value indicates healthier green vegetation. In a regression task to detect vegetation, a machine learning model should have a similar kind of behavior. In this section, we will give an overview of works related to the detection of vegetation and salinity levels, the prediction of necessary soil nutrients for crop development, the prediction of soil properties, and the prediction of solar radiation.

The most common application of regression models in environmental science is precisely the creation of models to detect some property on the terrain, such as vegetation, biomass, or salinity. In [97, 117], the authors apply the Std-GP algorithm to

satellite imagery to create vegetation indices that correlate with the Revised Universal Soil Loss Equation (RUSLE) C factor, an index that assesses how land use affects soil loss and sediment generation [8]. The authors improve previous work that could only be applied to green vegetation by creating models that also correlate with the RUSLE C factor in dry vegetation. In [31], the authors use GAs to evolve weights to be used in a function that tries to predict the NDVI values using only the visible wavelengths sensor (i.e., without using the NIR wavelength). Similarly, in [56], the authors use GAs to evolve weights to create a new vegetation index. In [5], the authors use the Std-GP algorithm to evolve indices for phenology analysis using time series data obtained from a hemispherical lens camera set in an 18-meter tall tower. In [28], Std-GP is used to predict levels of chlorophyll concentration, obtaining better results than traditional methods. In [108], the authors apply 14 machine learning methods, including Std-GP and Geometric Semantic GP (GSGP) [84] in the prediction of forest biomass. With this work, the authors conclude that even the best methods overfit the training data, indicating that this is a complex task, especially when using less than 100 samples for training. In [3], the authors use a GA-based algorithm to recommend nutrients for optimal crop development, improving soil fertility using time series, resulting in increased production. Similarly, in [29], the authors use a GP-based algorithm for precision farming, obtaining better results than those obtained using traditional approaches. In [130], the authors use GA to optimize the number of hidden layers and nodes in a CNN. In this work, the model is applied to the prediction of evapotranspiration values of soybean plantations in different growth stages. In [1], the authors use Std-GP to predict the fraction of calves that survive per cow each year in several farms. This work on precision farming leads to a better estimate of the number of calves that the farms should produce to minimize waste and maximize profits.

Some works are related to the application of the Std-GP algorithm to the prediction of soil properties. In [115], the authors use satellite imagery to predict the soil electrical conductivity and soil surface salinity. In [55], the authors study the Soil–Water Characteristic Curve (SWCC) of different kinds of soils from several soil properties. In [93], the authors study the saturated hydraulic conductivity of different soil types.

As an important theme in environmental science, several works focus on predicting several aspects related to energy. In [45], the authors use several EML algorithms to estimate the amount of solar radiation that would hit Queensland (Australia) using time series obtained from monthly predictors over 5 years. Similarly, in [9], the authors study the prediction of solar power for the integration of a photovoltaic site to improve the reliability of photovoltaic systems using time series, with measures of solar power every 5 minutes, over several days, in three different cities in Florida (USA). In [89], the authors use GAs to search for the optimal locations for photovoltaic installations in La Palma Del Condado (Spain) under a series of constraints to minimize environmental, safety, and economic risks. Those constraints make the algorithm avoid suggestions near environmentally protected areas, roads, urban areas, and vineyards (due to their importance to the local economy). Similarly, in [98], the authors also deal with the prediction of solar power production from solar panels, in Potsdam University (Germany). Gas turbines are also of great

importance for power generation. However, due to their complexity and their execution environment, they can have a high failure rate that leads to severe consequences. To minimize this problem, in [41], Std-GP and several variants of GSGP are used to predict the fuel flow and the exhaust gas temperature in two separate datasets, to improve the performance of the gas turbines.

Especially in summer, fire is the most destructive force in the Portuguese forest. Typically, there are between 15.000 and 25.000 forest fires each year across the whole country [79]. In [121], the authors use a GA to evolve the weights of ANNs. This model is then used to predict probabilities of fire ignitions in several municipalities. In [26], the authors compare GSGP with other well-known methods, including ANNs, to predict the sizes of burnt areas, while taking into consideration the characteristics of the terrain.

With more than half the world's population living in urban areas, air pollution is another theme of high importance in environmental science. Ozone concentration levels are related to air pollution, and low concentration levels have a negative impact on human health. The authors of [23] propose a new variant to the GSGP algorithm and compare it with GSGP and other well-known methods, including ANNs, to predict ozone concentration levels in the region of Yuen Long, one of the most polluted regions in China. This work proposes a faster method with better results and lower variance across the different runs. In [78], the authors propose a GA-based algorithm based on iterative rule learning. This algorithm is then used to detect atmospheric pollution by evolving a set of rules that are applied to the detection of high ozone concentration levels, nitrogen monoxide and sulfur dioxide. Later, in [77], the authors propose a new GA-based algorithm to evolve association rules for the detection of high ozone concentration levels.

Water-related environmental issues have also been the theme of some works. In [61], the authors use Std-GP to predict the quantity of the *Microcystis aeruginosa* bacteria in the Nakdong River in South Korea. In this work, the authors use two variants of Std-GP. In the first approach, the models exclusively use arithmetical operators, while in the second approach, the models also use the logical IF-THEN-ELSE operators. In [27], the authors use GP models to study the patterns of total phosphorus in Tampa Bay, Florida. In this work, GP models are trained to estimate the changing levels of total phosphorus in the bay and, after validation, are used to generate the map to be studied. In [19], the authors use the EPR algorithm to predict the number of pipe failures in water distribution systems from their attributes.

Some authors used the ENC and ENH datasets [119] to benchmark their methods (MRGP [6], MGP [64], FEAT [65], EPLEX and EPLEX-1M [63] and GSGP [25]). These are regression datasets to predict the energy performance of the residential buildings for cooling (ENC) and heating (ENH).

Finally, in [113], the authors use GP to build interpretable models to predict global mean temperatures. Their results suggest that GP can evolve interpretable models that are comparable to complex climate systems, even without resorting to historical data when making predictions.

19.6 Classification Tasks in Environmental Science

In classification tasks, the machine learning model has to associate each sample in a dataset with a nominal label. For example, a model trained to detect burnt areas should associate each dataset sample with the label “Burnt” or “Non-burnt”. Classification includes two categories: binary classification and multiclass classification. In binary classification, the model has to associate the sample with one of two classes. This task is usually considered simple. Regression models can also perform binary classification by defining a threshold that separates both classes. In multiclass classification, the model has to associate the sample with one of three or more classes.

19.6.1 Binary Classification

Std-GP and other GP algorithms are some of the popular choices for binary classification tasks. These algorithms are used to detect riparian zones using mixed data from two satellites [76], to detect water in cloudy satellite images [18], burnt areas in satellite images [15, 17, 21, 100, 107, 111, 112], to separate forest from savanna [4], and to identify cropland in high-resolution satellite imagery [37, 123]. Binary classification in remote sensing can often be associated with detecting a specific land cover type, as seen in Fig. 19.5, where a GP model is used to detect burnt areas [17]. GP algorithms can also be applied to high-resolution UAV imagery. In [69], the authors apply the Std-GP algorithm to an image segmentation problem to detect algae in river images obtained using a sensor onboard a UAV.

Outside the remote sensing area, Std-GP is used for the prediction of production errors in the steel industry [74]. Although this is not a direct application in environmental science, the authors state that a higher success rate in production results in

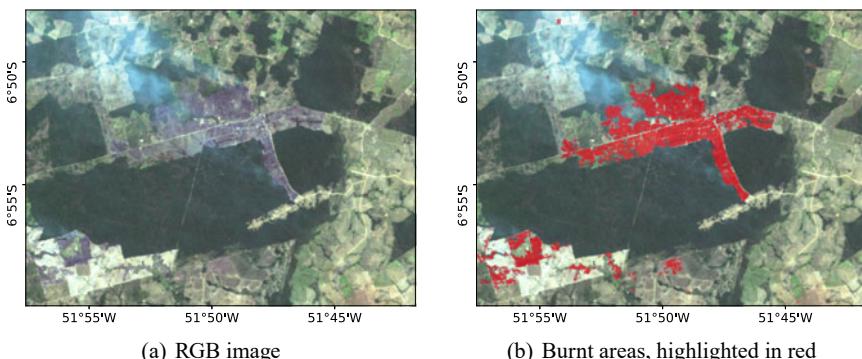


Fig. 19.5 Landsat-8 satellite imagery, obtained over the state of Pará (Brazil) in August 2015. The red areas marked in **b** highlight the areas classified as “burnt” by the machine learning model

lower energy requirements, which leads to fewer greenhouse gas emissions. Other authors apply GP [70] and GA [124] algorithms to the prediction of traffic congestion.

One of the advantages of using GP-based methods to detect specific classes is the flexibility of the application of the features produced. In [18], the authors compare the application of water detection indices [88] with those evolved by the Std-GP in a dataset that includes many pixels from cloudy locations. The authors conclude that the evolved features are better than some human-made indices and equally good to others. From a practical point of view, the features evolved by [76] can be considered indices to detect riparian zones. This statement implies that GP can produce indices that detect highly difficult land cover types, in opposition to detecting water and burnt areas.

Regression algorithms (e.g., Std-GP) usually produce a model equivalent to a mathematical expression. Since their output is numerical, it requires a final post-processing to perform binary classification. The typical approach is the use of a threshold that separates the two classes, as seen in [18, 100]. Since the classes are nominal and usually do not have an order, this approach is often not applicable to multiclass classification. An exception to this rule may be the discrimination between classes that are related to each other, e.g., barren areas, grasslands, and forests. In this example, we know that the classes have different biomass volumes, which can be used to easily order and classify the samples by using multiple thresholds [50].

19.6.2 Multiclass Classification

Multiclass classification is considered a more complex task, and two main approaches exist to solve it. The more direct option is to use a classifier that can perform multiclass classification by separating all classes at once, using a single model. However, these models are usually complex. Experts may decide to decompose a multiclass problem into several simpler, binary classification problems. There are multiple ways of doing this. The most common approach is the one-versus-all [110] strategy. This approach uses a divide-and-conquer strategy with an n -class problem divided into several binary classification problems. As exemplified in Fig. 19.6, the one-versus-all strategy performs a succession of binary classifications until the correct class is identified.

Some works apply this divide-and-conquer strategy to solve classification problems. In [110], the authors study a dataset obtained from sonar data to identify five types of seafloor habitats. By applying the one-versus-all strategy, the authors use four models to first separate sand from other habitats, then reef, then algae, and lastly to separate the *australis* and *sinuosa* seagrasses. In [32], the authors separate several species of vines by applying several methods, including Std-GP, using time series data that measure the intensity of fluorescence emissions using the Kautsky effect [58].

Many recent works focus on the application of multiclass classification models, leading to a simpler pipeline. In [24], the authors apply the Std-GP and GSGP

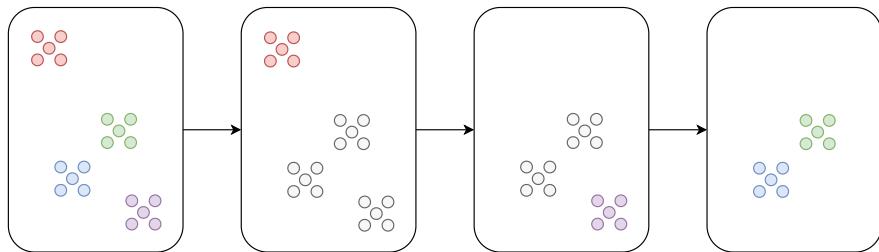


Fig. 19.6 One-versus-all strategy for multiclass classification. In this example, the prediction is made using, at most, three classifiers

algorithms for several multiclass datasets, including two datasets to detect land cover types. Although the GSGP produces better scores, both methods have low accuracy in a dataset containing land cover types in agroforest mosaics, reinforcing the difficulty of identifying plantations in agroforest land cover types. In [71], the authors propose the SAE-MOEA/D algorithm and use it to identify from three to five classes in several radar images from different sensors. In [12], the authors use a multi-objective GA algorithm (NSGA-II [33]) to identify croplands in satellite imagery. In [75], the authors propose a novel GP-based approach to evolve multiple features, testing it on a multiclass time series dataset. The evolved features are tested with several classifiers and the results indicate that these features are more robust than the traditional remote sensing indices developed by experts. In [17], the authors use the M3GP algorithm to detect burnt forest areas in satellite images. This algorithm created models that reduce the dimensionality of the problem from 10 features to 3 features. Besides increasing the accuracy of the machine learning models, this allowed the visualization of the new dataset in a 3D space. In [15], the authors use the M3GP algorithm in several remote sensing datasets for binary classification (detection of burnt areas) and multiclass classification (land cover type classification). The multiclass datasets included, among other classes, three subtypes of forest in satellite imagery. While most classes were easily separable, forest classes are often misclassified due to their spectral similarity. In this work, the authors compare the results from feature construction using M3GP with manual feature construction (i.e., using indices) and automatic feature construction using EFS [7] and the non-EML Fast Function Extraction (FFX) [80] algorithm. However, the automatic methods were limited to binary classification datasets. In [16], the authors use the M3GP in a multiclass time series dataset that includes, among other classes, two land cover types with similar spectral signatures, “natural forest” and “cocoa agroforest”. Since cocoa trees are planted in the shade of natural trees, the classes are nearly identical from the (optical) satellite point of view. Based on the analysis of the models and the frequency of selection of each feature, the authors conclude that, although the confusion matrices reveal some confusion between these classes, the features obtained in August are useful to detect cocoa. In [15], the authors use a group of classification datasets to benchmark their approach. The work includes two datasets related to the separation of land cover

types that several authors have used as a benchmark to test their algorithms, such as the M2GP [54], the eM3GP [109], the M3GP [85, 86], and the M4GP [66].

In [13], the authors use a PSO-based algorithm to identify six land cover types in a UAV imagery dataset. Other recent works also apply EML-based algorithms to classification in high-resolution aerial imagery with applications such as the identification of multiclass land cover-type detection using an evolutionary neural architecture search algorithm [20] and several EML algorithms [51, 106], and also to the detection of mistletoe using a GP-based approach [82].

19.7 Other Tasks in Environmental Science

Besides applying EML models to regression and classification problems, other authors apply EML methods to image alignment and parameter tuning.

Regarding image alignment, in [125], the authors propose LMACO (Multi-modal continuous ACO with Local search), an ACO-based algorithm that allows multiple images to be stacked automatically. The LMACO method outperforms other EML-based methods such as Comprehensive Learning PSO (CLPSO), Self-adaptive Differential Evolution (SaDE) [92], and the traditional ACO. According to the authors, this method can also be applied to image fusion, object recognition, and change detection.

Another popular application of EML-based models is parameter tuning. The objective of this task is to search for optimal parameters that maximize another algorithm's robustness. In [104], the authors use GA to tune the gain parameters on a Static Synchronous Compensator (STATCOM) to improve the amount of wind energy that is harvestable in their system. In [71], the authors propose a MOEA-based method that automatically finds sets of parameters and hyper-parameters to be used by stacked autoencoders in the classification of land cover types in PolSAR imagery.

In [45], the authors compare several EML and non-EML algorithms when predicting solar radiation. The main algorithms to solve this task are the Extreme Learning Machine (ELM), ANNs, Support Vector Regressors (SVR) [10], and the Multi-Gene GP (MGGP) [36] algorithms. However, except for MGGP, these algorithms rely on other methods for optimization. The authors use three versions of the ELM algorithm: the basic version, the online sequential ELM, and optimization using SaDE. The authors also use two versions of ANNs, where the weights are evolved using either the PSO or the GA. Lastly, the authors use three versions of the SVR, where some hyperparameters are picked using either grid search or PSO or GA. Other authors also apply ANNs to predict solar power [9] while using GA to optimize the model's weights and biases.

In [3], the authors use a GA-based algorithm to find an optimal set of parameters (optimal quantity of several nutrients) to be used for crop production. When compared to the traditional approach, the GA approach improved soil fertility, increasing the production of the crops in the study area.

19.8 Robustness of EML models

One of the known issues in remote sensing data is the radiometric variations across images caused by the presence of shadows, clouds, and moisture on the soil, among others. Consequently, machine learning algorithms may learn to identify a class based on a temporary characteristic or noise. This overfitting to the characteristics of a particular image may result in the model failing to detect the correct land cover types when given a different image.

Some works comment on the robustness of their models by applying them to different satellite images [16, 17] or different sensors [31]. In [31], the authors use GA to evolve the weights of an expression to simulate the NDVI index using visible wavelengths. This expression is applied to data from two quadcopters with different sensors (Matrice 210 and Phantom 4 Pro+), showing that this expression provides good results in multiple sensors with minimal input from the user. In [17], the authors use three burnt area detection datasets that were obtained from three different countries with different characteristics. Each of these datasets has training and test samples. The authors notice that if a model is trained in one dataset, although it has a very high test accuracy in the respective test set, the accuracy is reduced when applying the model to datasets from different locations. In an attempt to mitigate this lack of transferability, the authors show that several classification algorithms are more robust when trained using features evolved by the M3GP algorithm. Another approach used by the authors is to train models using samples from multiple datasets. In [16], the authors test the robustness of their algorithm by validating the models in an area much larger than the one used during the model's training phase. In that work, the lack of transferability is more evident in the non-EML classifiers that, while achieving higher accuracy values on the test set, are surpassed by the M3GP-based approaches when applied to the validation set.

In most environmental science applications, humans are the ones responsible for labeling samples in datasets. Still, datasets are prone to containing mislabeled samples, leading to imperfect datasets. If the machine learning algorithms are unable to identify mislabeled samples, the model will learn the incorrect labels, overfitting this erroneous data. In [111], the authors use a semi-supervised GP-based method that is able to ignore mislabeled samples by making use of unlabeled data together with the labeled set.

19.9 Examples of Human-Competitiveness

Over the years, EML algorithms produced human-competitive results in various applications from different research fields, including environmental science [59]. John R. Koza surveyed a large section of these works in 2010 [62] and, while this survey only contains works until 2009, other works can be seen in the “Humies” Awards website [53]. This award is given in a competition recognizing human-competitive

results produced by genetic and evolutionary computation. Besides the winners, the website also displays the work of the other applicants. This includes recent applications in environmental science and the EML methods used, such as the use of NSGA-II and Evolutionary Strategies (ES) in traffic light optimization [105] and for traffic flow optimization [114] using Green Swarm, a method-based on (10+2)-EA [116]. Other applications include predicting soil moisture using interpretable models [14] obtained from differential evolution and GAs, and the use of GP to predict wind damage [48].

As stated in the feature construction Sect. 19.4, machine learning experts spend a long time in feature engineering [35]. Although this task requires domain-specific knowledge, many automatic methods are being produced, simplifying this task. In this section, we follow the work of Batista et al., using their work as an example of human-competitiveness in automatic feature construction for remote sensing data. This is an extension of some of their work on the detection of burnt areas [17], separation of forest subtypes [15], separation of land cover types with nearly identical spectral signatures [16] and detection of water in cloudy satellite images [18].

In [17], the authors use the M3GP algorithm to evolve a set of robust features to detect burnt areas. In some cases, only three features were produced, which allowed the visualization of the new feature space in a 3D space, as shown in Fig. 19.7. These features were then directly applied to satellite images in order to understand what the models are “seeing” when classifying pixels, and it was noticed that some of the evolved features have a particular meaning. When applying the features to a satellite image, the authors noticed that each feature clearly highlights a particular terrain characteristic, rather than producing a noisy map that depends on the other

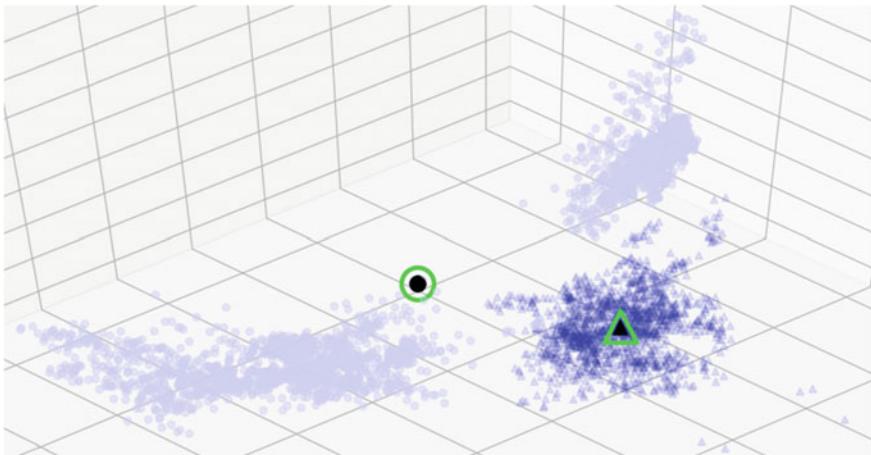


Fig. 19.7 Feature space of a dataset with the “burnt” (dark blue) and “non-burnt” (light blue) land cover types, after being converted by an M3GP model with three dimensions. The centroids of the clusters are marked in black and highlighted in green. This image shows in a clear way that the “non-burnt” cluster should be split into two clusters. *Image source* [17]

features to make sense. In one of the experiments, the final model contained three interpretable features. Two of those features are displayed in Eqs. 19.2 and 19.3. Equation 19.2 seems to highlight burnt areas, areas of low vegetation, and active fire. This statement can be justified by the use of the SWIR2 and Red bands in the equation. Equation 19.3 seems to highlight burnt areas, active fires, and water. This statement can also be justified by the bands selected by the M3GP algorithm for these features. By multiplying these two features, the resulting expression visibly highlights the burnt area better than the Normalized Burn Ratio (NBR) [60], an index of burn developed and used by remote sensing experts. This comparison is seen in Fig. 19.8.

$$(Blue + Red)/SWIR2 \quad (19.2)$$

$$(SWIR2 * Blue)/(Red * NIR) \quad (19.3)$$

In this case, the algorithm evolves features to detect burnt areas and, in [18], to detect water. Remote sensing experts have studied both tasks and indices to detect burnt [60] and water [88] exist. In [18], Std-GP is used to evolve a single feature that detects water pixels while being robust to cloud pixels. The authors study one dataset obtained from two satellite images over the Amazon River's delta and over Portugal. The accuracy values from five water indices are calculated and compared with the ones obtained with Std-GP. The results reveal that the evolved features are statistically better than three of the five human-made indices, and no statistically significant difference was found when comparing the results with the other two indices. Interestingly, Std-GP tends to pick the same features as those used by remote sensing experts, suggesting it can generate similar features without domain-specific knowledge.

In [15, 16], the M3GP algorithm evolves features that discriminate several similar land cover types (e.g., different forest types), a task that is harder than detecting burnt or water, or even simply detecting a forest class without subtypes. In the first article [15], several machine learning methods are used for automatic feature construction. The EFS, FFX, and M3GP algorithms are applied for binary classification (detection of burnt areas), and only the M3GP is applied for multiclass classification (mapping of land cover types). The results obtained using these features are compared with those obtained using indices created by remote sensing experts and without using feature construction methods. Given the difficulty of this multiclass classification task, and the fact that the M3GP-evolved features improved the generalization of the state-of-the-art classifiers, it seems clear that this kind of algorithm can compete with experts and their domain-specific knowledge in the development of indices.

In the second article [16], the authors compare the results from several classifiers (M3GP, multi-layer perceptron, random forest, ridge, ROCKET, and XGBoost) in three versions of a dataset containing several land cover types. One version is a uni-temporal dataset and two other versions are time-series datasets containing data from May to September. The data includes two land cover types with nearly

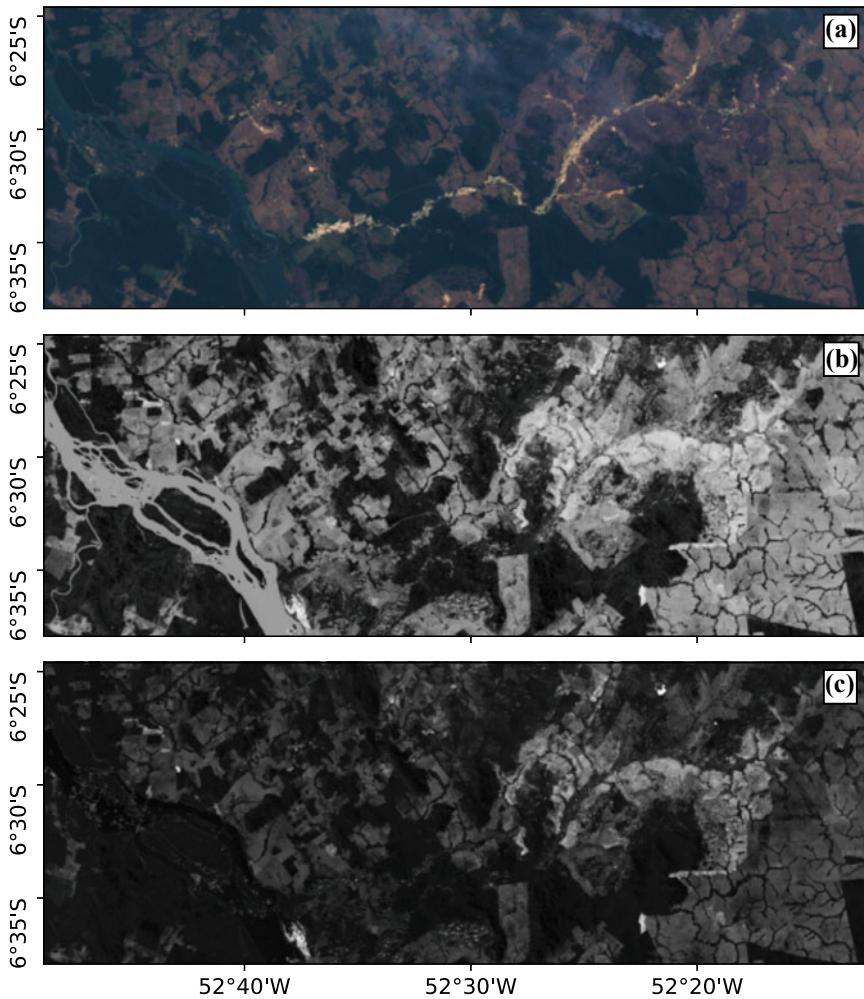


Fig. 19.8 Landsat-8 satellite imagery over Pará, Brazil during the 2015 wildfires. Figure **a** shows the RGB image; **b** shows the application of the NBR index to highlight burnt areas; **c** shows the application of the multiplication of Eqs. 19.2 and 19.3. Both (**b**) and (**c**) were normalized using their maximum and minimum values of the dataset used by the authors in [17]

identical spectral signatures: “forest” and “cocoa-agroforest”. Based on the positive results of using M3GP-evolved features in the previous work, the authors study the features generated by M3GP for this new task and learn that, although they do not significantly improve the results of the best classifier, M3GP frequently selects the features from August in the time series data. This reveals that this particular month is relevant to solve the problem. It also highlights the feature selection capabilities of the M3GP algorithm.

19.10 Conclusions

This chapter reviewed the use of Evolutionary Machine Learning (EML) in environmental science applications. The type of data used in these applications varies in several factors, such as source (*in situ* or remote sensing), temporality (uni-temporal or time series) and dimensionality (low to high number of features), and the applications include several different tasks (regression, classification, data cleaning, feature engineering, hyper-parameter optimization, image alignment, among others). This variety of data and application types shows the flexibility of EML.

From the papers reviewed, we observe that genetic algorithms and genetic programming have been popular choices in specific fields of environmental science. For at least three decades, genetic algorithms have been used for automatic feature selection, producing great results in high-dimensional datasets. Algorithms based on genetic programming are also popular choices for automatic feature construction applications. Genetic programming algorithms can be considered human-competitive since they are applied to feature construction tasks where they outperform the experts in the field. Among the EML-based algorithms, they also seem to be a popular choice for classification tasks.

In conclusion, even though EML is still not the first choice, or regarded as state of the art, for environmental science applications, many EML-based methods have been created and used in environmental science applications over the last decades. This reveals a fairly large interest from experts in this area, most probably connected to the advantageous characteristics of EML such as robustness and interpretability.

Acknowledgements This work was supported by the FCT, Portugal, through funding of the LASIGE Research Unit (UIDB/00408/2020 and UIDP/00408/2020); João Batista was supported by PhD grant SFRH/BD/143972/2019.

References

1. Abbona, F., Vanneschi, L., Bona, M., Giacobini, M.: A GP approach for precision farming. In: 2020 IEEE Congress on Evolutionary Computation (CEC). IEEE, July 2020
2. Abdella, M., Marwala, T.: The use of genetic algorithms and neural networks to approximate missing data in database. In: IEEE 3rd International Conference on Computational Cybernetics, 2005. ICCC 2005, pp. 207–212 (2005)
3. Ahmed, U., Chun-Wei Lin, J., Srivastava, G., Djenouri, Y.: A nutrient recommendation system for soil fertilization based on evolutionary computation. Comput. Electron. Agric. **189**, 106407 (2021)
4. Almeida, A.E., da Torres, R.S.: Remote sensing image classification using genetic-programming-based time series similarity functions. IEEE Geosci. Remote Sens. Lett. **14**(9), 1499–1503 (2017)
5. Almeida, J., dos Santos, J.A., Miranda, W.O., Alberton, B., Morellato, L.P.C., da Torres, R.S.: Deriving vegetation indices for phenology analysis using genetic programming. Ecol. Inf. **26**, 61–69 (2015)

6. Arnaldo, I., Krawiec, K., O'Reilly, U.-M.: Multiple regression genetic programming. In: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation. ACM, July 2014
7. Arnaldo, I., O'Reilly, U.-M., Veeramachaneni, K.: Building predictive models via feature synthesis. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, pp. 983–990, July 2015
8. Asadi, H., Dastorani, M.T., Khosravi, K., Sidle, R.C.: Applying the C-Factor of the RUSLE model to improve the prediction of suspended sediment concentration using smart data-driven models. *Water* **14**(19) (2022)
9. Asrari, A., Wu, T.X., Ramos, B.: A hybrid algorithm for short-term solar power prediction—sunshine state case study. *IEEE Trans. Sust. Energy* **8**(2), 582–591 (2017)
10. Awad, M., Khanna, R.: Support vector regression. In: Efficient Learning Machines, pp. 67–80. Apress (2015)
11. Bacardit, J., Brownlee, A.E.I., Cagnoni, S., Iacca, G., McCall, J., Walker, D.: The intersection of evolutionary computation and explainable AI. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO '22, pp. 1757–1762, New York, NY, USA. Association for Computing Machinery (2022)
12. Bandyopadhyay, S., Maulik, U., Mukhopadhyay, A.: Multiobjective genetic clustering for pixel classification in remote sensing imagery. *IEEE Trans. Geosci. Remote Sens.* **45**(5), 1506–1511 (2007)
13. Banerjee, B.P., Raval, S.: A particle swarm optimization based approach to pre-tune programmable hyperspectral sensors. *Remote Sens.* **13**(16), 3295 (2021)
14. Basak, A., Mengshoel, O.J., Schmidt, K., Kulkarni, C.: Wetting and drying of soil: From data to understandable models for prediction. In: 2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA). IEEE, Oct. 2018
15. Batista, J.E., Cabral, A.I.R., Vasconcelos, M.J.P., Vanneschi, L., Silva, S.: Improving land cover classification using genetic programming for feature construction. *Remote Sens.* **13**(9), 1623 (2021)
16. Batista, J.E., Rodrigues, N.M., Cabral, A.I.R., Vasconcelos, M.J.P., Venturieri, A., Silva, L.G.T., Silva, S.: Optical time series for the separation of land cover types with similar spectral signatures: cocoa agroforest and forest. *Int. J. Remote Sens.* **43**(9), 3298–3319 (2022)
17. Batista, J.E., Silva, S.: Improving the detection of burnt areas in remote sensing using hyper-features evolved by M3GP. In: 2020 IEEE Congress on Evolutionary Computation (CEC). IEEE, July 2020
18. Batista, J.E., Silva, S.: Evolving a cloud-robust water index with genetic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion. ACM, July 2022
19. Berardi, L., Giustolisi, O., Kapelan, Z., Savic, D.A.: Development of pipe deterioration models for water distribution systems using EPR. *J. Hydroinf.* **10**(2), 113–126 (2008)
20. Broni-Bediako, C., Murata, Y., Mormille, L.H., Atsumi, M.: Evolutionary NAS for aerial image segmentation with gene expression programming of cellular encoding. *Neural Comput. Appl.* **34**(17), 14185–14204 (2021)
21. Cabral, A.I.R., Silva, S., Silva, P.C., Vanneschi, L., Vasconcelos, M.J.: Burned area estimations derived from landsat ETM+ and OLI data: comparing genetic programming with maximum likelihood and classification and regression trees. *ISPRS J. Photogramm. Remote Sens.* **142**, 94–105 (2018)
22. Carvalho, P., Ribeiro, B., Rodrigues, N.M., Batista, J.E., Vanneschi, L., Silva, S.: Feature selection on epistatic problems using genetic algorithms with nested classifiers. In: Applications of Evolutionary Computation, pp. 656–671. Springer Nature Switzerland (2023)
23. Castelli, M., Gonçalves, I., Trujillo, L., Popović, A.: An evolutionary system for ozone concentration forecasting. *Inf. Syst. Front.* **19**(5), 1123–1132 (2016)
24. Castelli, M., Silva, S., Vanneschi, L., Cabral, A., Vasconcelos, M.J., Catarino, L., Carreiras, J.M.B.: Land cover/land use multiclass classification using GP with geometric semantic operators. In: Applications of Evolutionary Computation, pp. 334–343. Springer, Berlin, Heidelberg (2013)

25. Castelli, M., Trujillo, L., Vanneschi, L., Popović, A.: Prediction of energy performance of residential buildings: a genetic programming approach. *Energy Build.* **102**, 67–74 (2015)
26. Castelli, M., Vanneschi, L., Popović, A.: Predicting burned areas of forest fires: an artificial intelligence approach. *Fire Ecol.* **11**(1), 106–118 (2015)
27. Chang, N.-B., Xuan, Z., Yang, Y.J.: Exploring spatiotemporal patterns of phosphorus concentrations in a coastal bay with MODIS images and machine learning models. *Remote Sens. Environ.* **134**, 100–110 (2013)
28. Chen, L.: A study of applying genetic programming to reservoir trophic state evaluation using remote sensor data. *Int. J. Remote Sens.* **24**(11), 2265–2275 (2003)
29. Chion, C., Landry, J.-A., Da Costa, L.: A genetic-programming-based method for hyperspectral data information extraction: agricultural applications. *IEEE Trans. Geosci. Remote Sens.* **46**(8), 2446–2457 (2008)
30. Chuvieco, E.: *Fundamentals of Satellite Remote Sensing*. CRC Press, Jan. 2020
31. Costa, L., Nunes, L., Ampatzidis, Y.: A new visible band index (vNDVI) for estimating NDVI values on RGB images utilizing genetic algorithms. *Comput. Electron. Agric.* **172**, 105334 (2020)
32. da Silva, J.M., Figueiredo, A., Cunha, J., Eiras-Dias, J.E., Silva, S., Vanneschi, L., Mariano, P.: Using rapid chlorophyll fluorescence transients to classify vitis genotypes. *Plants* **9**(2), 174, (2020)
33. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* **6**(2), 182–197 (2002)
34. Deb, K.: Multi-objective optimisation using evolutionary algorithms: an introduction. In: *Multi-objective Evolutionary Optimisation for Product Design and Manufacturing*, pp. 3–34. Springer, London (2011)
35. Domingos, P.: A few useful things to know about machine learning. *Commun. ACM* **55**(10), 78–87 (2012)
36. Dominic, P., Leahy, D., Willis, M.: GPTIPS:an open source genetic programming toolbox for multigene symbolic regression. *Lect. Notes Eng. Comput. Sci.* **2180**, 12 (2010)
37. dos Santos, J.A., Ferreira, C.D., da Torres, R.S., Gonçalves, M.A., Lamparelli, R.A.C.: A relevance feedback method based on genetic programming for classification of remote sensing images. *Inf. Sci.* **181**(13), 2671–2684 (2011)
38. Dubey, A., Inoue, A.H., Birnmann, P.T.F., da Silva, S.R.: Evolutionary feature selection. In: *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM, July 2022
39. Eberhart, R., Kennedy, J.: A new optimizer using particle swarm theory. In: *MHS'S95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*. IEEE (1995)
40. Eklund, N.H.W.: Using genetic algorithms to estimate confidence intervals for missing spatial data. *IEEE Trans. Syst. Man Cybern. Part C (Applications and Reviews)* **36**(4), 519–523 (2006)
41. An experimental study: Enríquez-Zárate, J., Trujillo, L., de Lara, S., Castelli, M., Z-Flores, E., Muñoz, L., Popović, A.: Automatic modeling of a gas turbine using genetic programming. *Appl. Soft Comput.* **50**, 212–222 (2017)
42. ESA. Sentinel 2 User Guide, Accessed 31 Aug. 2022. <https://sentinels.copernicus.eu/web/sentinel/user-guides/sentinel-2-msi/resolutions/spatial>
43. ESA. Sentinel 1 User Guide, Accessed 22 May 2023. <https://sentinels.copernicus.eu/web/sentinel/user-guides/sentinel-1-sar>
44. García, J.C.F., Kalenatic, D., Bello, C.A.L.: Missing data imputation in multivariate data by evolutionary algorithms. *Comput. Hum. Behav.* **27**(5), 1468–1474 (2011). 2009 Fifth International Conference on Intelligent Computing
45. Ghimire, S., Deo, R.C., Downs, N.J., Raj, N.: Self-adaptive differential evolutionary extreme learning machines for long-term solar radiation prediction with remotely-sensed MODIS satellite and reanalysis atmospheric products in solar-rich cities. *Remote Sens. Environ.* **212**, 176–198 (2018)

46. Giustolisi, O., Savic, D.A.: A symbolic data-driven technique based on evolutionary polynomial regression. *J. Hydroinf.* **8**(3), 207–222 (2006)
47. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016). <http://www.deeplearningbook.org>
48. Hart, E., Sim, K., Gardiner, B., Kamimura, K.: A hybrid method for feature construction and selection to improve wind-damage prediction in the forestry sector. In: Proceedings of the Genetic and Evolutionary Computation Conference, New York, NY, USA, July 2017. ACM (2017)
49. Hasan, M.K., Alam, M.A., Roy, S., Dutta, A., Jawad, M.T., Das, S.: Missing value imputation affects the performance of machine learning: a review and analysis of the literature (2010–2021). *Inf. Med. Unlock.* **27**, 100799 (2021)
50. Hashim, H., Latif, Z.A., Adnan, N.A.: Urban vegetation classification with NDVI threshold value method with very high resolution (VHR) pleiades imagery. *Int. Archiv. Photogramm. Remote Sens. Spat. Inf. Sci.* **XLII-4/W16**, 237–240, Oct. 2019
51. He, C., Zhang, Y., Gong, D.: A pseudo-label guided artificial bee colony algorithm for hyperspectral band selection. *Remote Sens.* **12**(20), 3456 (2020)
52. Holland, J.H.: Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology Control and Artificial Intelligence. MIT Press, Cambridge, MA, USA (1992)
53. Humies. Annual “Humies” Awards For Human-Competitive Results. Accessed 22 May 2023. <https://www.human-competitive.org/>
54. Ingallali, V., Silva, S., Castelli, M., Vanneschi, L.: A multi-dimensional genetic programming approach for multi-class classification problems. In: Lecture Notes in Computer Science, pp. 48–60. Springer, Berlin Heidelberg (2014)
55. Johari, A., Habibagahi, G., Ghahramani, A.: Prediction of soil–water characteristic curve using genetic programming. *J. Geotech. Geoenvir. Eng.* **132**(5), 661–665 (2006)
56. Kabiri, P., Pandi, M.H., Nejat, S.K., Ghaderi, H.: NDVI optimization using genetic algorithm. In: 2011 7th Iranian Conference on Machine Vision and Image Processing. IEEE, Nov. 2011
57. Kanan, H.R., Faez, K., Taheri, S.M.: Feature selection using ant colony optimization (ACO): a new method and comparative study in the application of face recognition system. In: Perner, P. (ed.) Advances in Data Mining. Theoretical Aspects and Applications, pp. 63–76. Springer, Berlin, Heidelberg (2007)
58. Kautsky, H., Hirsch, A.: Neue versuche zur kohlensäureassimilation. *Die Naturwissenschaften* **19**(48), 964–964 (1931)
59. Keijzer, M., Baptist, M., Babovic, V., Uthurburu, J.R.: Determining equations for vegetation induced resistance using genetic programming. In: Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation, GECCO ’05, pp. 1999–2006, New York, NY, USA (2005). Association for Computing Machinery
60. Key, C., Benson, N.: Landscape Assessment: Ground measure of severity, the Composite Burn Index; and Remote sensing of severity, the Normalized Burn Ratio, pp. LA 1–51. USDA Forest Service, Rocky Mountain Research Station, Jan. 2006
61. Kim, D.-K., Cao, H., Jeong, K.-S., Recknagel, F., Joo, G.-J.: Predictive function and rules for population dynamics of microcystis aeruginosa in the regulated nakdong river (South Korea), discovered by evolutionary algorithms. *Ecol. Model.* **203**(1–2), 147–156 (2007)
62. Koza, J.R.: Human-competitive results produced by genetic programming. *Gen. Programm. Evolvable Mach.* **11**(3–4), 251–284 (2010)
63. La Cava, W., Helmuth, T., Spector, L., Moore, J.H.: A probabilistic and multi-objective analysis of lexicase selection and ε -lexicase selection. *Evol. Comput.* **27**(3), 377–402 (2019)
64. La Cava, W., Moore, J.H.: Semantic variation operators for multidimensional genetic programming. In: Proceedings of the Genetic and Evolutionary Computation Conference. ACM, July 2019
65. La Cava, W., Moore, J.H.: Learning feature spaces for regression with genetic programming. *Genet. Program. Evolvable Mach.* **21**(3), 433–467 (2020)

66. La Cava, W., Silva, S., Danai, K., Spector, L., Vanneschi, L., Moore, J.H.: Multidimensional genetic programming for multiclass classification. *Swarm Evol. Comput.* **44**, 260–272 (2019)
67. Lary, D.J., Alavi, A.H., Gandomi, A.H., Walker, A.L.: Machine learning in geosciences and remote sensing. *Geosci. Front.* **7**(1), 3–10 (2016)
68. Lardi, R., Boggia, R., Terrile, M.: Genetic algorithms as a strategy for feature selection. *J. Chemom.* **6**(5), 267–281 (1992)
69. Lensen, A., Al-Sahaf, H., Zhang, M., Verma, B.: Genetic programming for algae detection in river images. In: 2015 IEEE Congress on Evolutionary Computation (CEC). IEEE, May 2015
70. Li, X., Mabu, S., Zhou, H., Shimada, K., Hirasawa, K.: Genetic network programming with estimation of distribution algorithms for class association rule mining in traffic prediction. In: IEEE Congress on Evolutionary Computation. IEEE, July 2010
71. Liu, G., Li, Y., Jiao, L., Chen, Y., Shang, R.: Multiobjective evolutionary algorithm assisted stacked autoencoder for PolSAR image classification. *Swarm Evol. Comput.* **60**, 100794 (2021)
72. Liu, H., Motoda, H.: Feature Extraction, Construction and Selection: A Data Mining Perspective. Kluwer Academic Publishers, USA (1998)
73. Lobato, F., Sales, C., Araujo, I., Tadaiesky, V., Dias, L., Ramos, L., Santana, A.: Multi-objective genetic algorithm for missing data imputation. *Pattern Recogn. Lett.* **68**, 126–131 (2015)
74. Lotz, M., Silva, S.: Application of genetic programming classification in an industrial process resulting in greenhouse gas emission reductions. In: Applications of Evolutionary Computation, pp. 131–140. Springer, Berlin, Heidelberg (2010)
75. Miao, L., Bi, Y., Xue, B., Qiong, H., Zhang, M., Wei, Y., Yang, P., Wenbin, W.: Genetic programming for high-level feature learning in crop classification. *Remote Sens.* **14**(16), 3982 (2022)
76. Makkeasorn, A., Chang, N.-B., Li, J.: Seasonal change detection of riparian zones with remote sensing images and genetic programming in a semi-arid watershed. *J. Environ. Manag.* **90**(2), 1069–1080 (2009)
77. Martínez-Ballesteros, M., Martínez-Álvarez, F., Troncoso, A., Riquelme, J.C.: An evolutionary algorithm to discover quantitative association rules in multidimensional time series. *Soft Comput.* **15**(10), 2065–2084 (2011)
78. Martínez-Ballesteros, M., Troncoso, A., Martínez-Álvarez, F., Riquelme, J.C.: Mining quantitative association rules based on evolutionary computation and its application to atmospheric pollution. *Integr. Comput.-Aided Eng.* **17**(3), 227–242 (2010)
79. Mateus, P., Fernandes, P.M.: Forest fires in Portugal: Dynamics, causes and policies. In: Forest Context and Policies in Portugal, pp. 97–115. Springer International Publishing (2014)
80. McConaghay, T.: FFX: Fast, Scalable, Deterministic Symbolic Regression Technology, pp. 235–260. Springer, New York, NY (2011)
81. Mei, Y., Chen, Q., Lensen, A., Xue, B., Zhang, M.: Explainable artificial intelligence by genetic programming: a survey. *IEEE Trans. Evol. Comput.* **1**–1 (2022)
82. Mejia-Zuluaga, P.A., Dozal, L., Valdiviezo-N, J.C.: Genetic programming approach for the detection of mistletoe based on UAV multispectral imagery in the conservation area of mexico city. *Remote Sens.* **14**(3), 801 (2022)
83. Miller, G.T., Spoolman, S.: Environmental Science, 13th ed. Wadsworth Publishing, Belmont, CA, Jan. 2010
84. Moraglio, A., Krawiec, K., Johnson, C.G.: Geometric semantic genetic programming. In: Lecture Notes in Computer Science, pp. 21–31. Springer, Berlin, Heidelberg (2012)
85. Muñoz, L., Silva, S., Trujillo, L.: M3GP—multiclass classification with GP. In: Lecture Notes in Computer Science, pp. 78–91. Springer International Publishing (2015)
86. Muñoz, L., Trujillo, L., Silva, S.: Transfer learning in constructive induction with genetic programming. *Genetic Programm. Evol. Mach.* **21**(4), 529–569 (2019)
87. Muñoz, L., Trujillo, L., Silva, S., Castelli, M., Vanneschi, L.: Evolving multidimensional transformations for symbolic regression with M3GP. *Memetic Comput.* **11**(2), 111–126 (2019)

88. Mustafa, M.T., Hassoon, K.I., Hussain, H.M., Abd, M.H.: Using water indices (NDWI, MNDWI, NDMI, WRI and AWEI) to detect physical and chemical parameters by apply remote sensing and GIS techniques. *Int. J. Res.—Granthaalayah* **5**(10), 117–128 (2017)
89. Nagkoulis, N., Loukogeorgaki, E., Ghislanzoni, M.: Genetic algorithms-based optimum PV site selection minimizing visual disturbance. *Sustainability* **14**(19) (2022)
90. NASA. Landsat 8 Bands. Accessed 25 Aug. 2022. <https://landsat.gsfc.nasa.gov/landsat-8/landsat-8-bands>
91. Olson, R.S., Bartley, N., Urbanowicz, R.J., Moore, J.H.: Evaluation of a tree-based pipeline optimization tool for automating data science. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016, GECCO '16, pp. 485–492, New York, NY, USA. ACM (2016)
92. Omran, M.G.H., Salman, A., Engelbrecht, A.P.: Self-adaptive differential evolution. In: Computational Intelligence and Security, pp. 192–199. Springer, Berlin, Heidelberg (2005)
93. Parasuraman, K., Elshorbagy, A., Si, B.C.: Estimating saturated hydraulic conductivity using genetic programming. *Soil Sci. Soc. Am. J.* **71**(6), 1676–1684 (2007)
94. Pei, W., Xue, B., Shang, L., Zhang, M.: New fitness functions in genetic programming for classification with high-dimensional unbalanced data. In: 2019 IEEE Congress on Evolutionary Computation (CEC), pp. 2779–2786. IEEE (2019)
95. Pereira, S.C., Lopes, C., Pedroso, J.P.: Mapping cashew orchards in cantanhez national park (Guinea-Bissau). *Remote Sens. Appl.: Soc. Environ.* **26**, 100746 (2022)
96. Poli, R., Langdon, W.B., McPhee, N.: A Field Guide to Genetic Programming. Lulu Enterprises ltd., UK, Jan. 2008
97. Puente, C., Olague, G., Smith, S.V., Bullock, S.H., Hinojosa-Corona, A., González-Botello, M.A.: A genetic programming approach to estimate vegetation cover in the context of soil erosion assessment. *Photogramm. Eng. Remote Sens.* **77**(4), 363–376 (2011)
98. Quade, M., Abel, M., Shafi, K., Niven, R.K., Noack, B.R.: Prediction of dynamical systems by symbolic regression. *Phys. Rev. E* **94**(1), (2016)
99. Rodrigues, N.M., Batista, J.E., La Cava, W., Vanneschi, L., Silva, S.: Slug: feature selection using genetic algorithms and genetic programming. In: Medvet, E., Pappa, G., Xue, B. (eds.) Genetic Programming, Cham, pp. 68–84. Springer International Publishing (2022)
100. Rodrigues, N.M., Batista, J.E., Silva, S.: Ensemble genetic programming. In: Hu, T., Lourenço, N., Medvet, E., Divina, F. (eds.), Genetic Programming, Cham, pp. 151–166. Springer International Publishing (2020)
101. Rolnick, D., Donti, P.L., Kaack, L.H., Kochanski, K., Lacoste, A., Sankaran, K., Ross, A.S., Milojevic-Dupont, N., Jaques, N., Waldman-Brown, A., Lucchini, A.S., Maharaj, T., Sherwin, E.D., Mukkavilli, K., Kording, K.P., Gomes, C.P., Ng, A.Y., Hassabis, D., Platt, J.C., Creutzig, F., Chayes, J., Bengio, Y.: Tackling climate change with machine learning. *ACM Comput. Surv.*, **55**(2) (2022)
102. Rouse, J.W., Haas, R.H., Schell, J.A., Deering, D.W.: Monitoring vegetation systems in the great plains with ERTS. In: NASA Special Publication (1973)
103. Rudin, C.: Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nat. Mach. Intell.* **1**(5), 206–215 (2019)
104. Saxena, N.K., Kumar, A.: Reactive power control in decentralized hybrid power system with STATCOM using GA, ANN and ANFIS methods. *Int. J. Electr. Power Energy Syst.* **83**, 175–187 (2016)
105. Semet, Y., Berthelot, B., Glais, T., Isbérie, C., Varest, A.: Expert competitive traffic light optimization with evolutionary algorithms. In: Proceedings of the 5th International Conference on Vehicle Technology and Intelligent Transport Systems. SCITEPRESS—Science and Technology Publications (2019)
106. Shang, Y., Zheng, X., Li, J., Liu, D., Wang, P.: A comparative analysis of swarm intelligence and evolutionary algorithms for feature selection in SVM-based hyperspectral image classification. *Remote Sens.* **14**(13), 3019 (2022)
107. Silva, S., Dignum, S., Vanneschi, L.: Operator equalisation for bloat free genetic programming and a survey of bloat control methods. *Gen. Programm. Evol. Mach.* **13**(2), 197–238 (2011)

108. Silva, S., Ingallali, V., Vinga, S., Carreiras, J.M.B., Melo, J.B., Castelli, M., Vanneschi, L., Gonçalves, I., Caldas, J.: Prediction of forest aboveground biomass: an exercise on avoiding overfitting. In: Applications of Evolutionary Computation, pp. 407–417. Springer Berlin Heidelberg (2013)
109. Silva, S., Muñoz, L., Trujillo, L., Ingallali, V., Castelli, M., Vanneschi, L.: Multiclass classification through multidimensional clustering. In: Genetic Programming Theory and Practice XIII, pp. 219–239. Springer International Publishing (2016)
110. Silva, S., Tseng, Y.-T.: Classification of seafloor habitats using genetic programming. In: Lecture Notes in Computer Science, pp. 315–324. Springer, Berlin, Heidelberg (2008)
111. Silva, S., Vanneschi, L., Cabral, A.I.R., Vasconcelos, M.J.: A semi-supervised genetic programming method for dealing with noisy labels and hidden overfitting. *Swarm Evol. Comput.* **39**, 323–338 (2018)
112. Silva, S., Vasconcelos, M.J., Melo, J.B.: Bloat free genetic programming versus classification trees for identification of burned areas in satellite imagery. In: Applications of Evolutionary Computation, pp. 272–281. Springer, Berlin, Heidelberg (2010)
113. Stanislawkska, K., Krawiec, K., Kundzewicz, Z.W.: Modeling global temperature changes with genetic programming. *Comput. Math. Appl.* **64**(12), 3717–3728 (2012)
114. Stolfi, D.H., Alba, E.: Greener routes with bio-inspired techniques. *Green swarm. Appl. Soft Comput.* **71**, 952–963 (2018)
115. Taghizadeh-Mehrjardi, R., Ayoubi, S., Namazi, Z., Malone, B.P., Zolfaghari, A.A., Sadrabadi, F.R.: Prediction of soil surface salinity in arid region of central Iran using auxiliary variables and genetic programming. *Arid Land Res. Manag.* **30**(1), 49–64 (2016)
116. Back, T.: Evolutionary Algorithms in Theory and Practice. Oxford University Press, New York, NY (1996)
117. Trabucchi, M., Puente, C., Comin, F.A., Olague, G., Smith, S.V.: Mapping erosion risk at the basin scale in a mediterranean environment with opencast coal mines to target restoration actions. *Reg. Environ. Change* **12**(4), 675–687 (2012)
118. Tran, B., Xue, B., Zhang, M.: Genetic programming for feature construction and selection in classification on high-dimensional data. *Memet. Comput.* **8**(1), 3–15 (2015)
119. Tsanas, A., Xifara, A.: Accurate quantitative estimation of energy performance of residential buildings using statistical machine learning tools. *Energy Build.* **49**, 560–567 (2012)
120. Vadillo, J., Santana, R., Lozano, J.A.: When and how to fool explainable models (and humans) with adversarial examples (2021)
121. Vasconcelos, M., Silva, S., Tomé, M., Alvim, M., Pereira, J.: Spatial prediction of fire ignition probabilities: comparing logistic regression and neural networks. *Photogram. Eng. Remote Sens.* **67**, 73–81, 01 (2001)
122. Viegas, F., Rocha, L., Gonçalves, M., Mourão, F., Sá, G., Salles, T., Andrade, G., Sandin, I.: A genetic programming approach for feature selection in highly dimensional skewed data. *Neurocomputing* **273**, 554–569 (2018)
123. Wen, C., Miao, L., Bi, Y., Zhang, S., Xue, B., Zhang, M., Zhou, Q., Wenbin, W.: An object-based genetic programming approach for cropland field extraction. *Remote Sens.* **14**(5), 1275 (2022)
124. Wen, F., Zhang, G., Sun, L., Wang, X., Xiaowei, X.: A hybrid temporal association rules mining method for traffic congestion prediction. *Comput. Ind. Eng.* **130**, 779–787 (2019)
125. Yue, W., Ma, W., Miao, Q., Wang, S.: Multimodal continuous ant colony optimization for multisensor remote sensing image registration with local search. *Swarm Evol. Comput.* **47**, 89–95 (2019)
126. Yang, H., Du, Q.: Particle swarm optimization-based dimensionality reduction for hyperspectral image classification. In: 2011 IEEE International Geoscience and Remote Sensing Symposium. IEEE (2011)
127. Yang, Y., Xue, B., Jesson, L., Zhang, M.: Genetic programming for symbolic regression: a study on fish weight prediction. In: 2021 IEEE Congress on Evolutionary Computation (CEC). IEEE, June 2021

128. Zhang, Q.-S., Zhu, S.-C.: Visual interpretability for deep learning: a survey. *Front. Inf. Technol. Electron. Eng.* **19**(1), 27–39 (2018)
129. Zhang, Yu., Tiňo, P., Leonardis, A., Tang, K.: A survey on neural network interpretability. *IEEE Trans. Emerging Top. Comput. Intell.* **5**(5), 726–742 (2021)
130. Zhang, Y., Zhou, Y., Jiang, S., Ning, S., Jin, J., Cui, Y., Wu, Z., Feng, H.: A simulation study using machine learning and formula methods to assess the soybean groundwater contribution in a drought-prone region. *Water* **14**(19) (2022)

Chapter 20

Evolutionary Machine Learning in Medicine



Michael A. Lones and Stephen L. Smith

Abstract This chapter reviews applications of evolutionary machine learning within the medical domain. It is divided into three parts. The first two parts give examples of recent work in two important and representative diseases, cancer and COVID-19, showing how evolutionary methods can be applied to diverse tasks in diagnosis, epidemiological modelling, and the design of drug interventions and treatment plans. The third part presents a case study of our own work within the area of Parkinson's disease, demonstrating how an evolutionary machine learning approach has been successfully translated and applied within clinical settings.

20.1 Introduction

Medicine and evolutionary algorithms (EAs) have been bedfellows for quite some time. EAs work well on problems that are difficult and poorly understood, and medicine is full of problems that are difficult and poorly understood. Medicine is a broad field, and the applications of EAs within medicine reflect this. However, it is not practical, nor perhaps desirable, to review everything that has been done in this area. Rather, in this review, we predominantly focus on areas of medicine where EAs have been widely applied, and/or where we have done work in the past and have enough expertise to comment intelligently on what has been done. The review focuses on evolutionary machine learning (EML), i.e. where EAs are used as part of a data-driven machine learning process. However, where relevant, we also note applications that are not directly data-driven. We focus on three domains: cancer, COVID-19 and Parkinson's disease. The first of these, cancer, is a large and important area of medicine and gives us an opportunity to demonstrate the breadth of

M. A. Lones

Department of Computer Science, Heriot-Watt University, Edinburgh, UK

S. L. Smith (✉)

Department of Electronic Engineering, University of York, York, UK

e-mail: stephen.smith@york.ac.uk

EA-based approaches used in medicine. The second, COVID-19, is a recent application area (at least at the time of writing) and allows us to highlight EML methods which are currently popular in the medical domain. The third, Parkinson’s disease, allows us to present a case study of our own work where EML research has been taken through the translational pipeline and is now being used within a clinical setting. After reviewing these three areas, we conclude with some thoughts on recent trends and future directions within EML in a medical context.

20.2 Cancer

Cancer has a major impact on people and society, and, consequently, has become a popular target of ML methods. In this respect, EAs are no exception, and the literature on applying EAs to problems in cancer goes back for decades. In order to make this review more focused and contemporary, we mainly limit it to papers published in the last 5 years (for a review of earlier work, see [81]). However, we also reference seminal studies from earlier years. Potential applications in this area are diverse, since there are many types of cancer, and each has its own considerations in terms of factors such as diagnosis and clinical intervention. Nevertheless, existing work can generally be divided into that which seeks to improve diagnosis, and that which seeks to improve treatment, and this section is organised into two parts to reflect this. Table 20.1 summarises the papers covered in this review.

20.2.1 *Improving Diagnosis*

There are many papers that report results from applying EAs to cancer diagnosis. Many of these apply standard EA approaches to off-the-shelf data sets, e.g. those in the UCI repository [21]. In the interests of space, we do not review these. Instead, we focus on studies in which a clear medical or clinical need is being addressed.

One area in which EAs are often applied in a diagnostic context is feature selection. A good example of this is in microarray data analysis. Microarray data sets, which capture snapshots of the biochemical components present in a biological sample at a particular time, typically comprise many features for a comparatively small number of samples. To identify useful sets of biomarkers for use in clinical diagnosis, and avoid overfitting, it is important to reduce the number of features. Various studies have shown EAs to be effective at doing this. A recent example is reported in [72], where a multi-objective memetic algorithm was used to find molecular signatures for five different types of cancer. Notably, the authors found this approach to be more effective than other methods both for finding a minimal set of predictive biomarkers and for finding a set of biomarkers with high predictive accuracy. EAs have also been applied to feature selection within other kinds of cancer data. For instance, in

Table 20.1 Selected applications of EAs within cancer, ordered by publication date

Study	EA	Application
Walker et al. [79]	Cartesian GP	Breast cancer diagnosis from mammograms
Lones et al. [41]	Cartesian GP	Raman fingerprints of thyroid cancers
Fan et al. [24]	Not specified	MRI imaging biomarkers for breast cancer
Sadowski et al. [62]	MOEA	Optimisation of brachytherapy
Paruch [59]	GA/ES hybrid	Parameterisation of hyperthermic therapy
Beford et al. [9]	GA	Radiotherapy design and device evaluation
Luong et al. [46]	MOEA	Optimisation of brachytherapy plans
Lu et al. [44]	GA	Breast cancer prognosis
Wang et al. [80]	Memetic	Combination chemotherapy schedules
Wu et al. [82]	GP (and PSO)	Cancer diagnosis using DNA microarrays
Elia et al. [22]	Multi-gene GP	Early diagnosis from fluid samples
Fan et al. [25]	Not specified	Radiogenomic prediction of survival
Shindi et al. [67]	Hybrid MOEAs	Optimisation of chemotherapy
Su et al. [72]	MO memetic	Biomarkers for five cancer types
Alderdice et al. [4]	GA	Gene targets for immune-checkpoint therapy
Keshavarz et al. [53]	GA	Optimised model of lung cancer metastasis
Li et al. [34]	GA	Repurposing drugs for anti-cancer therapy
Maleki et al. [47]	GA	Lung cancer staging and prognosis
Panjwani et al. [58]	NSGA-II	Chemotherapy planning for late stage cancer
Stillman et al. [71]	GA	Design of anti-cancer nanomedicines
Taou et al. [74]	GA	Controlling gene expression states in cells
Tsompanas et al. [76]	GA	Nanomedicines for heterogenous tumours
Zhang et al. [86]	Surrogate-based ES	Classifying lung cancer malignancy
Ain et al. [2]	Multi-tree GP	Skin cancer diagnosis from images
D'Angelo et al. [15]	Multi-gene GP	Diagnosis of pancreatic cancer
Tian et al. [75]	Co-ev. MOEA	Optimisation of radiotherapy plans

[24], the authors used an EA to select MRI features for use in a logistic regression model used for breast cancer diagnosis. This allowed them to identify a number of 3D imaging features that could potentially be used as clinical biomarkers.

Other research has focused not only on the identification of biomarkers but also on understanding the interactions between biomarkers within the diagnostic process. A popular approach in this context is genetic programming (GP), specifically the use of symbolic regression to find equations that describe the relationship between a group of biomarkers and a diagnostic outcome. An early example of this is our own work on using Cartesian GP for diagnosing head and neck cancers from Raman spectroscopy data [41], where we were able to derive equations that captured the relationships between Raman spectra and disease severity. There are also a number of more recent examples, most of which have used GP variants that represent solutions

using multiple trees. For example, in [22], this approach was used to discover a minimal set of biomarkers for use in the early diagnosis of cancers from non-invasive fluid samples. A similar approach was later applied to diagnosing pancreatic cancer [15]. However, work by [2] particularly highlights the benefits of using multi-tree GP. The authors applied this approach to skin cancer diagnosis from image data, using individual trees within each solution to construct features from different low-level feature pools. The constructed features represented by the individual trees were then shown to be beneficial when building conventional ML models to perform diagnosis. The authors also demonstrated that the constructed features were interpretable, which is arguably an important advantage of GP over neural network-based approaches to feature construction.

Other work has focused on constructing features directly from image data, rather than relying on previously extracted low-level features. This includes early work by one of the authors, which used Cartesian GP to construct features and classifiers to diagnose breast cancer from mammograms [79]. However, whilst there has been more recent work on *de novo* feature construction and image classification using GP, this is an area that is now more commonly addressed using convolutional neural networks (CNNs). Instead, EAs are increasingly being used in concert with CNNs. We will discuss this later when we get to COVID-19 diagnosis. However, a good example within cancer diagnosis is reported in [86], where an evolutionary strategy was used to optimise the hyperparameters of a DNN model used for classifying lung cancer malignancy. Two notable aspects of this approach were the use of fitness surrogates, motivated by the expensive training times of DNNs, and an importance-based mutation operator. The latter allowed the algorithm to focus on hyperparameters which have a particularly large effect on performance, which in their case included the convolutional feature maps and the dropout rates.

Another paper worth noting is [82], where *complex networks* were evolved to diagnose cancer from DNA microarray data. Complex networks, within this context, are similar to neural networks, but with scale-free or small-world topologies, and non-standard activation functions. The authors used a combination of GP to evolve the topology, and PSO to optimise the parameters, and were able to generate ensemble classifiers that outperformed various conventional ML models, including ensembles of neural networks. This highlights another prominent advantage of EAs: the ability to optimise a wide range of models.

There are also examples of EAs being applied to prognosis rather than diagnosis [1, 25, 44, 47]. On the whole, these use similar methods to those already described. For example, in [25], an EA was used to carry out feature selection of imaging features in the construction of radiogenomic models used to predict survival outcomes of patients. In [1], the authors use an EA to optimise the hyperparameters of DNN models in order to predict the outcome of breast cancer, though it is notable that they used an MOEA (specifically NSGA-III) to explore trade-offs between different metrics.

20.2.2 *Improving Treatment*

EAs have been used in a number of studies that aim to design or optimise treatments for cancer, with the majority of these studies focusing on chemotherapy and radiotherapy.

The largest group of recent papers addresses the problem of designing treatment plans [46, 58, 62, 67, 75, 80]. For both chemotherapy and radiotherapy, there is typically a need to maximise the treated area (i.e. the removal of cancerous cells) whilst minimising the impact on healthy tissue. Such problems can naturally be posed as multi-objective optimisation problems and solved using various forms of multi-objective EAs, and there are a number of examples of this approach. For example, NSGA-II has been used to plan multi-drug chemotherapy schedules for treating late-stage drug-resistant tumours [58], and it has been shown that optimisation of chemotherapy schedules can be improved by hybridising multiple MOEAs [67]. In both [62] and [46], it was shown that MOEAs can be used to plan brachytherapy, a form of radiation therapy that involves moving radiation sources through the body. A particularly nice study of applying MOEAs can be found in [75], where the authors demonstrate how more advanced approaches can be used to solve radiotherapy planning problems that have a large number of objectives and parameters. In addition to generating better treatment plans, in both [46] and [75], it is noted that MOEA-based approaches also play an important role in reducing the amount of time it takes to plan treatments.

EAs are also being applied to the design of new cancer treatments. Both [4] and [34] used GAs in studies that addressed the early stages of the anti-cancer drug pipeline. In [4], the GA was used in the process of finding gene targets for immune-checkpoint therapy, i.e. looking for new drug targets that play an important role in regulating the activity of immune cells within cancerous tumours. The authors report that they were able to identify several potential drug targets using this approach. By comparison, [34] addressed the problem of repurposing existing drugs for anti-cancer treatment, by identifying potential associations between existing drugs and the genetic components of cancer cells. In both of these papers, the GA was used to identify gene associations.

In [74], we pursued a different approach to designing novel treatments. Cancer is often the result of a cell's gene regulatory dynamics entering an abnormal attractor, so we considered whether a GA could be used to design a controller to guide a diseased cell back to a healthy state. Specifically, we used a GA to optimise the structure and parameters of a Boolean network, since these have the potential to be deployed within a cellular environment using synthetic biology principles. Our results showed that it is possible to create optimal controllers for a range of different cell types, with good scalability. This is also another example of an EA being used to optimise a fairly non-standard computational model.

Another important issue in drug-based cancer therapies is the deployment of anti-cancer medications to affected cells and tissues. Both [71, 76] addressed this by focusing on the design of nanoparticle systems. By delivering anti-cancer drugs to

specific cells, nanoparticles offer significant potential for carrying out more targeted treatment at lower dosages. One difficulty is that there are many degrees of freedom when designing nanoparticles, and both groups of authors noted that their use of an EA was motivated by the need to find optimal solutions within these large search spaces. In [71], a GA was used to design the characteristics of the nanoparticles themselves, including their concentration, size, binding affinities and payloads. The candidate designs were evaluated using a complex agent-based simulation of tumour growth and nanoparticle transport, and could successfully target over 90% of cancer cells whilst maintaining a low dosage. Using this approach, the authors were able to find suitable designs for both homogenous tumours and mildly heterogeneous tumours. In [76], the authors focused on heterogeneous tumours with larger degrees of differentiation. In these kinds of tumours, the presence of multiple cell types can lead to drug resistance. To address this, a variable-length EA with parsimony pressure was used to optimise the application times of multiple nanoparticles, with this more dynamic approach allowing them to find solutions that could successfully target highly differentiated tumours, at least when evaluated in simulation.

Many of the approaches mentioned above rely on *in silico* simulators of cancer systems in order to evaluate their solutions. Consequently, the accuracy of these simulators is a key factor in the design of meaningful solutions. In this respect, it is also worth noting the work reported in [53], where a GA was used to parameterise a model of lung cancer metastasis in order to improve its fit to real-world data.

20.3 COVID-19

This review was written during the COVID-19 pandemic. Unsurprisingly, many recent papers focus on applications related to this disease. This section provides an overview of these works and highlights several key studies. It is divided into four parts: these review the use of EAs in diagnosing COVID-19, building models of how the disease spreads, designing therapeutic and preventative interventions and scheduling resources to manage the disease. Table 20.2 summarises the papers covered.

20.3.1 Diagnosis

Most of the EA literature in COVID-19 diagnosis focuses on classifying radiographic images, particularly using thoracic X-rays and computerised tomography (CT) scans. Many authors cite this as a potentially faster and more accurate route to screening than conventional diagnostic tests, and the barrier to entry for research in this area is low due to the availability of numerous public data sets.

The majority of these studies focus on the use of EAs within the training of deep neural networks (DNNs). A common approach is to use the EA to adapt a

Table 20.2 Applications of EAs to COVID-19, ordered by publication year then author

Study	EA	Application
Ghosh et al. [27]	GA	Probabilistic CA model of multiple countries
Hernandez et al. [29]	DE	In-host model of COVID-19 infection
Koziol et al. [31]	GA	SIR epidemiological model of Italy and Spain
Kwuimy et al. [32]	GA	SEIR epidemiological model of South Korea
Libotte et al. [36]	MODE	Control of vaccine administration in China
Miralles-Pechuan et al. [52]	GA	Optimising government interventions
Niazkar et al. [55]	GP	Pandemic trend forecasting for various countries
Olivier et al. [56]	GA	Optimal lockdown strategies in South Africa
Quaranta et al. [60]	DE	SIR epidemiological model of Italy
Salgotra et al. [64]	GP	Pandemic trend forecasting for various countries
Shaban et al. [65]	GA	Feature selection for CT classification
Yousefpour et al. [84]	MOGA	Optimal control policies
Akram et al. [3]	GA	Feature selection for CT classification
Anfèelifá et al. [5]	GP	Pandemic trend forecasting for various countries
Aversano et al. [6]	GA	Optimising DNNs for CT classification
Bartz-Beielstein et al. [7]	DE+ES	Optimising hospital simulation for planning
Basha et al. [8]	GA	Training fuzzy rules for X-ray classification
Bosowski et al. [11]	GA	Optimising ensembles for X-ray classification
Carvalho et al. [12]	GA	Feature selection for CT classification
Cheng et al. [13]	GA	Optimising drug candidates
Dallocchio et al. [14]	GA	Repurposing antivirals for COVID-19 treatment
De Falco et al. [17]	DE	SEIAR epidemiological model of Italy
Dixit et al. [20]	DE+PSO	Feature selection for X-ray classification
Li et al. [35]	GA	Optimisation of IR plasmonic sensor
Louati et al. [43]	GA	Optimising CNNs for X-ray classification
Luo et al. [45]	GA	Epidemiological model of various countries
Matabuena et al. [48]	ES	Stochastic epidemiological model of Spain
Miikkulainen et al. [49]	MOEA	Optimisation of RNN-generated interventions
Milani et al. [50]	DE	Optimisation of vaccine sites
Pinto Neto et al. [54]	MOGA	Compartmental model of Sao Paulo
Rodriguez et al. [33]	PSO	Evolving GANs for X-ray synthesis
Singh et al. [69]	MOGA	Optimising CNNs for X-ray classification
Shukla et al. [68]	MOGA	Fine-tuning DNNs for X-ray classification
Vieira et al. [78]	GA	Fine-tuning DNNs for X-ray classification
Yarsky et al. [83]	GA	SEIR epidemiological model of US states
Zhang et al. [85]	GA	Hybrid epidemiological model of Brazil
Bi et al. [10]	DE	Optimisation of intervention sequences
De Falco et al. [23]	DE	Training if-then rules for X-ray classification
De Freitas et al. [18]	ES+PSO	Feature selection for blood analysis
Gopalakrishnan et al. [28]	GA	Discovery of drug targets
Panigrahi et al. [57]	GA	Epidemiological models of India
Tange et al. [73]	NSGA-II	Optimisation of vaccine sites and allocation

complex pre-trained DNN model to the specific task of discriminating COVID-19 from normal images, typically by optimising hyperparameters of the model [6, 68, 78]. A good example is the work of [6], who used a GA to tune the block-level properties, the number and sizes of the fully connected layers, the location and characteristics of dropout layers and the choice of optimisers for three pre-trained DNN models (ResNet50, VGG19 and Xception). Others have used EAs to perform a more general architecture search, with the aim of finding new architectures that work well within this particular context [43, 69], and there has also been work on optimising the topologies of GANs to generate synthetic COVID-19 images for use within a data augmentation context [33]. Another notable study used EAs to train ensembles of DNNs [11]. Rather than optimising hyperparameters of the DNNs, this approach used a selection of 170 pre-trained DNNs and then used a GA to combine these base models into an ensemble. The authors explored various ensemble models and used the GA to optimise membership, weights, and, in the case of an MLP-based meta-model, the model's parameters. The authors found the resulting model to generalise particularly well across a number of different data sets.

A common criticism of DNN models is that they are difficult, and often impossible, to interpret. This is a particular problem in medicine, and has motivated some authors to focus on simpler models for COVID-19 diagnosis. In most cases, this involves a two-stage process of feature engineering followed by training a more traditional machine learning model, with the EA potentially involved in either of these two stages. There are various methods of extracting features from images, but these typically result in a lot of features, many of which are redundant. To address this, a popular use of GAs is to carry out feature selection [3, 12, 20, 65]. In [12], for example, the authors used a relatively shallow CNN model to extract convolutional features, and then a GA to further reduce these. Although their focus was on reducing computational overheads, this approach also goes some way towards making neural models more interpretable within a medical context.

Other authors have focused on using EAs to explicitly train more interpretable models [8, 23]. In [23], a DE-based approach was used to construct interpretable if-then rules, with each rule having a simple conjunction of feature value ranges as its condition. The authors used only 29 features, each capturing relatively interpretable characteristics of the image, such as colour and texture moments. However, it remains uncertain whether these features will be meaningful from a clinical perspective, but this highlights the implicit difficulty of building interpretable models from image data. There are also a few examples in the literature of authors applying EA techniques to non-image data sets. For example, in [18], a hybrid feature selection approach based around ES and PSO is used to reduce numerical and categorical features derived from blood analysis.

A further interesting example of an EA being used with a diagnostic context is work by [35], in which a GA is used to optimise the configuration of a novel sensing device. Their work was motivated by issues associated with existing screening techniques: for instance, the low speed of PCR tests, and the need to disinfect CT scanners between patients. The GA was used to optimise the arrangement of nano-metal structures within an infrared plasmonic sensor, and was able to find a

structure that detects the infrared fingerprint of COVID-19 molecules with high sensitivity.

20.3.2 Epidemiological Modelling

Epidemiological models are used to predict the temporal or spatial spread of a disease within one or more geographical areas. There is usually a need to fit certain parameters of the model to historical disease data, a process referred to as calibration. Since the relationship between model parameters and model behaviour is, in general, highly non-linear and poorly understood, it makes sense to use a gradient-free optimiser to carry out this calibration process. Within this context, there are many examples of evolutionary algorithms being used. Reflecting their wider popularity, GAs have been widely used for this [31, 32, 45, 54, 83, 85]. Differential evolution (DE) has also become popular within the broader model calibration literature, and there are also numerous examples of DE being used to calibrate COVID-19 models [17, 29, 57, 60]. Evolutionary strategies have also been used [48]—though perhaps less than might be expected, given that model evaluation is an expensive process where there might be a benefit to using algorithms with smaller populations.

These approaches have allowed many different types of epidemiological model to be fitted to COVID-19 data, including relatively unusual approaches such as probabilistic cellular automata [27]. In this respect, evolutionary approaches provide a useful tool for experimenting with new kinds of models. A further benefit to using evolutionary methods is the ease with which they can be extended to multi-objective problems, allowing the exploration of trade-offs between models. An example of this is [54], who used a multi-objective GA to consider interventions over a broad range of scenarios in their epidemiological model of São Paolo.

Once an epidemiological model has been calibrated, it can then be used to make predictions. However, there has also been some investigation of using evolutionary approaches to directly model disease trends from data, specifically by using GP to construct equations that predict the case numbers within a particular geographical region [5, 55, 63, 64]. As the authors of [64] note, a benefit of using GP in this context is that it leads to relatively interpretable models that can provide insight into the factors driving disease trends.

20.3.3 Designing Interventions

A further use of epidemiological models is to explore the effect of interventions on disease dynamics. However, a number of authors have gone further and used EAs in the process of designing interventions [10, 36, 49, 52, 56, 84]. These approaches generally view the design of interventions as an optimisation process and, given that the search spaces associated with interventions are generally large and complex,

EAs are a natural choice for exploring them. Typically, a pre-trained epidemiological model is coupled to some kind of controller whilst it executes, with the main difference between the various approaches being the form of the controller. In the simplest cases, the controller is a fixed sequence of interventions that are applied at regular intervals; for instance, [10] use a multi-population form of DE to explore sequences of weighted pre-specified intervention policies. Other approaches use conventional closed-loop controllers to control parameters of the epidemiological model, with the EA being used to tune parameters of the controller. For instance, in [56], a GA is used in this role to tune the control of lockdown levels. In the case of [49], the controller is a recurrent neural network whose form is optimised using a surrogate-assisted EA. A further benefit of using EAs in this context is the ease with which multiple objectives can be added, and both [49] and [10] make use of this feature to explore trade-offs between reduction in case numbers and the number and/or costs of non-pharmaceutical interventions. These two papers also demonstrate a clear benefit to using an EA-driven search for interventions over the use of other approaches.

EAs have also been used in the design process of pharmaceutical interventions. In fact, evolutionary approaches have been widely used in drug design thanks to the presence of EA-based optimisers in the popular open-source package AutoDock. This is a piece of software that implements *in silico* molecular docking as a means for carrying out virtual screening of drug candidates. It does this by simulating the fit between a drug candidate and a target molecule, and the EA is used within this process to determine the optimal fit between the two structures. In this context, EAs have been used in a number of studies of COVID-19 drug candidates; for example, in [14], the authors note that AutoDock's Lamarckian GA was used to determine the fit between existing antivirals and the crystal structure of COVID-19 molecular components. In [13], the authors also use AutoDock's GA to determine the fit between COVID-19 components and prospective drug candidates. However, they go one stage further and also use a GA in the search for drug candidates. Specifically, a graph variational autoencoder is used to learn the molecular structure of existing drugs, and is then used to generate an initial population of prospective candidates. A GA is then used to optimise this population, using various crossover operators to create new candidate structures, and these candidate structures are then evaluated using AutoDock to provide a measure of fitness. Using this approach, the authors were able to find a number of structures that had greater binding affinity to components of the COVID-19 virus than existing antivirals.

Another example of a GA being used in the drug development pipeline, described in [28], focuses on identifying the components of COVID-19 which are likely to be useful targets for drugs. Specifically, they use the GA within the process of modelling the virus's protein–protein interaction network, with the GA being used to fine-tune the weights of the network. Those components of the network with a high degree of centrality are then hypothesised to be the most influential within the infection process, and the authors use this analysis to identify several potential drug targets.

20.3.4 Scheduling Resources

Scheduling and planning is an area in which EAs have long been used, and there are several interesting examples of EAs being applied to problems related to COVID-19. Two of these studies relate to the planning of vaccine sites. In [50], the focus is on optimising the spatial distribution and capacities of vaccine units. Specifically, the authors developed a layered grid-based model of population distribution and transportation infrastructure, and used EAs to optimise a COVID facilities overlay. Notably, in an experimental comparison of optimisers, they found DE to work best for this particular problem. In [73], the authors used a somewhat different approach to address the somewhat similar problem of planning vaccine site location, the composition and replenishment of each vaccine site, and the allocation of recipients to sites, whilst optimising for both patient travel distance and operational cost. They formulated this as a mathematical programming model, and found that an EA-based approach (specifically using NSGA-II) was more effective than a conventional method, achieving significantly better savings for both objectives.

A different problem is targeted in [7], whose authors developed a discrete-event simulation of a hospital environment for use in capacity planning. Similarly to the epidemiology models discussed above, the problem here is to fit the parameters of the simulation to data. However, this example is particularly challenging due to noise and relatively high dimensionality and the authors tried out a number of different optimisation methods before finding one that worked—which turned out to be a surrogate-based EA. Specifically, this uses two component EAs: DE to select the hyperparameters of a Gaussian process-based surrogate model, and a form of ES to optimise the parameters of the simulation.

20.4 Parkinson’s Disease Case Study

The authors have been involved with a number of studies targeting Parkinson’s disease (PD) in which EML was used as a central mechanism. These include the development of models and tools to diagnose PD at an early stage [40, 70, 77], to measure symptoms and inform medication regimes [26, 38], interpret MRI imaging [19], for use within clinical trials [16], and in the study of animal models of the disease [30, 61].

One of the central methods used in a number of these studies was a form of GP known as implicit-context representation Cartesian Genetic Programming (IRCGP), a derivative of CGP [51] that uses a bottom-up representation introduced in [39]. This was used to fit symbolic arithmetic expressions to PD data, using the resulting model as a classifier. One of the benefits of this approach was that it offered a more expressive representation than traditional decision trees whilst maintaining a reasonable level of interpretability, something that could not readily be achieved using neural networks and other traditional ML models. A benefit to using

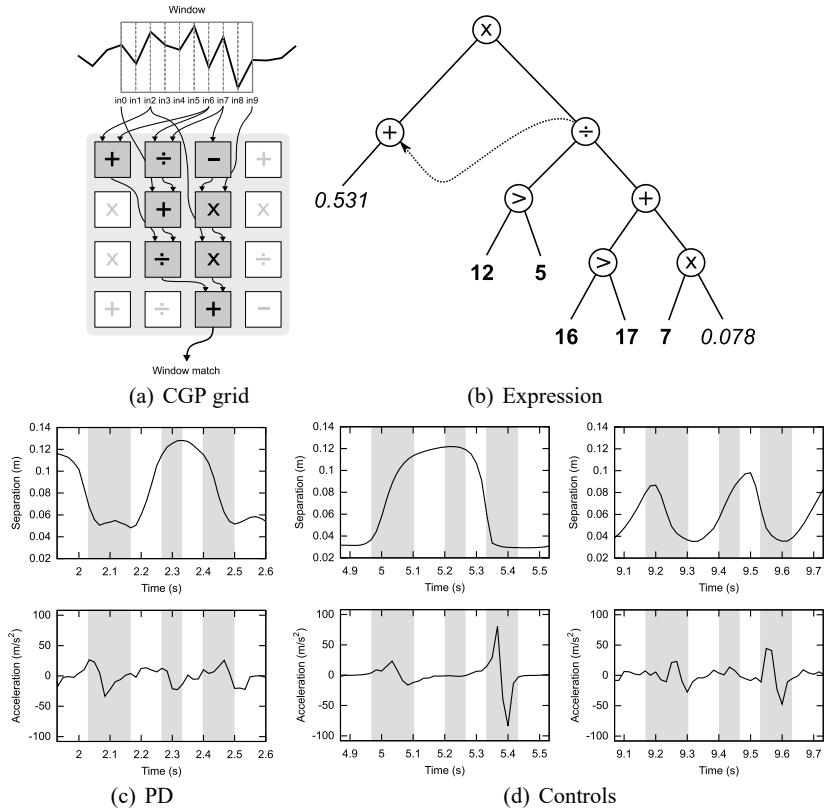


Fig. 20.1 A symbolic expression learnt to discriminate finger taps performed by Parkinson’s patients from those performed by aged-matched controls, showing **a** how expressions are laid out on a Cartesian plane and receive inputs from windows of movement data, **b** the evolved expression, with integer-valued terminals representing offsets within a window and the dotted-line showing reuse of a sub-expression, **c** a correctly classified PD finger tap, **d** two correctly classified control finger taps, highlighting the diversity in movements amongst subjects [40]

a CGP-derived method in particular was the use of a grid to constrain the size and topology of expressions, avoiding the kind of bloat seen in other forms of GP and increasing the likelihood of finding interpretable and generalizable expressions.

As an example of this, Fig. 20.1 shows a symbolic expression that was learnt to discriminate finger taps performed by Parkinson’s patients from those performed by aged-matched control subjects [40]. Whilst apparently quite simple, this achieved a level of discrimination that exceeded many trained clinicians. Further analysis in [37] revealed that it captured a number of subtle features of the movement process, including the peak magnitude of deceleration during the closing phase of a finger tap, which by itself turned out to be a better measure of PD severity than those commonly used in clinical assessment. This model went on to be used as a component of an ensemble model used in a tool for clinical diagnostic support [26].

A similar approach was later used for measuring the side effects of Parkinson’s medication [38]. Because of side effects and the non-linearity of drug response, the clinical management of PD drugs tends to be quite challenging. The system we developed specifically measured the side effects of levodopa, one of the primary drugs used in PD management, indicating whether the dosage level was too high. Prescribing levodopa is a fine balance between treating symptoms of the disease (such as tremor and impaired movement) and the occurrence of uncontrollable movements, known as levodopa-induced dyskinesia, or LID. Again, we found that a relatively simple symbolic expression could be used to recognise LID, and it was also able to correctly classify different severities of LID. This model is also now embedded in a tool being used in clinical settings.

In [30], this approach was used in a slightly different context to characterise movement disorder in zebrafish. In particular, gene editing had been used to induce a movement disorder in zebrafish which is analogous to PD in humans. The intention was to use this animal model to speed up the development of new treatments for PD in humans, whilst also reducing the need for vertebrate animals in research. Within this context, it is also worth noting our work in [61] which was also concerned with reducing the need for animals in research—in this case using an EA to fit a biophysical computational model of PD.

Although IRCGP has been a key method within many of these approaches, it is not the only evolutionary method that we have used. For example, another key element within the ensemble model [40] developed for clinical diagnostic support was an artificial biochemical network. This is a relatively unusual computational approach that came out of a project looking at developing new connectionist models motivated by cellular biochemical networks, with the aim of finding models complementary to existing artificial neural networks. Since we were doing this research in parallel with our PD work, it was natural to try out some of these models within a PD context. In particular, it turned out that artificial metabolic networks (whose function was motivated by cellular metabolism) were particularly good at discriminating signals in movement data.

More generally, this is a good example of the flexibility of EAs, which can (within reason of course) be used to fit the parameters of almost any kind of model to data. This is in contrast to learning algorithms commonly used in deep learning, which require model components to be differentiable in order to calculate gradient and back-propagate errors. In our case, it allowed us to investigate a wide range of models, some of which contained quite exotic components. The model we eventually selected was notable for using coupled discrete maps, which are non-linear systems that exhibit chaotic dynamics. In [42], we reasoned that these elements allowed the system to respond to quite subtle signals in the data stream.

20.5 Conclusions

In this review, we have presented and discussed various examples of how EAs, and EML in particular, have been used to solve medical problems. Generally speaking, the EA approaches used within this domain are broad and diverse, and have been applied at various stages of the ML pipeline. However, certain trends are apparent. Perhaps most notable, in terms of the number of citations, is the wealth of feature selection examples, an area in which EAs seem to offer useful benefits over more traditional ML approaches.

Arguably a more significant trend, however, is the increasing use of EAs for hyperparameter optimisation. Whilst there are still examples of EAs being used to directly optimise the parameters of ML models, e.g. when using GP or exotic models that do not have a natural optimiser pairing, more recent papers have tended to use EAs to optimise the hyperparameters of ML models. This is most evident when deep learning models are used, where the large number of parameters do not favour the use of EAs. However, EAs are clearly useful for tuning the hyperparameters of these models, as well as performing other related tasks such as model selection and ensemble learning. Beyond deep learning, their use for hyperparameter optimisation has also been amply demonstrated within the fitting of hyperparameters for data-driven epidemiological models.

A clear benefit of EML is the ready availability of mature multiobjective methods. Despite the ubiquitous occurrence of multiobjective problems in the real world, most techniques used within both traditional ML and deep learning do not seriously consider the trade-offs between objectives. In this context, multiobjective EML can offer a fresh perspective, and numerous studies have shown its benefits.

Nevertheless, these are challenging times for any methodology that is not deep learning. The recent advent of transformers, particularly in the area of medical imaging, has been rapid and significant. In a recent review article by Shamshad et al. [66], work is reported that claims to fully replace standard convolutions in deep neural networks by employing vision transformers operating on a sequence of image patches. These have been successfully applied to image operations, such as classification, object detection and registration, that are of significance to medical applications. However, EML has proven to offer a flexible means of investigating medical systems, particularly clinical assessments, where properties such as interpretability continue to be a valuable research tool to better understand the condition under examination.

References

1. Abdikenov, B., Iklassov, Z., Sharipov, A., Hussain, S., Jamwal, P.K.: Analytics of heterogeneous breast cancer data using neuroevolution. *IEEE Access* **7**, 18050–18060 (2019)
2. Ain, Q.U., Al-Sahaf, H., Xue, B., Zhang, M.: Genetic programming for automatic skin cancer image classification. *Exp. Syst. Appl.* **197**, 116680 (2022)

3. Akram, T., Attique, M., Gul, S., Shahzad, A., Altaf, M., Naqvi, S.S.R., Damaševičius, R., Maskeliūnas, R.: A novel framework for rapid diagnosis of covid-19 on computed tomography scans. *Pattern Anal. Appl.* **24**, 951–964, 8 (2021)
4. Alderdice, M., Craig, S.G., Humphries, M.P., Gilmore, A., Johnston, N., Bingham, V., Coyle, Vicky, S., Seedevi, L., Daniel, B., Loughrey, M.B., McQuaid, S., James, J.A., Salto-Tellez, M., Lawler, M., McArt, D.G.: Evolutionary genetic algorithm identifies il2rb as a potential predictive biomarker for immune-checkpoint therapy in colorectal cancer. *NAR Genom. Bioinf.* **3**, 4 (2021)
5. Andelic, N., Segota, S.B., Lorencin, I., Mrzljak, V., Car, Z.: Estimation of covid-19 epidemic curves using genetic programming algorithm. *Health Inf. J.* **27**, 1–40, 1 (2021)
6. Aversano, L., Bernardi, M.L., Cimitile, M., Pecori, R.: Deep neural networks ensemble to detect covid-19 from CT scans. *Pattern Recogn.* **120**, 108135, 12 (2021)
7. Bartz-Beielstein, T., Dröscher, M., Gür, A., Hinterleitner, A., Lawton, T., Mersmann, O., Peeva, D., Reese, L., Rehbach, N., Rehbach, F., Sen, A., Subbotin, A., Zaeffferer, M.: Optimization and adaptation of a resource planning tool for hospitals under special consideration of the covid-19 pandemic. In: 2021 IEEE Congress on Evolutionary Computation, CEC 2021—Proceedings, pp. 728–735 (2021)
8. Basha, S.H., Anter, A.M., Hassanien, A.E., Abdalla, A.: Hybrid intelligent model for classifying chest x-ray images of covid-19 patients using genetic algorithm and neutrosophic logic. *Soft Comput.* 1–16, 8 (2021)
9. Bedford, J.L., Ziegenhein, P., Nill, S., Oelfke, U.: Beam selection for stereotactic ablative radiotherapy using cyberknife with multileaf collimation. *Med. Eng. Phys.* **64**, 28–36, 2 (2019)
10. Bi, L., Mohammad, F., Hu, G.: Covid-19 forecasting and intervention planning using gated recurrent unit and evolutionary algorithm. *Neural Comput. Appl.* 1–19, 5 (2022)
11. Bosowski, P., Bosowska, J., Nalepa, J.: Evolving deep ensembles for detecting covid-19 in chest x-rays. In: 2021 IEEE International Conference on Image Processing (ICIP), pp. 3772–3776. Institute of Electrical and Electronics Engineers (IEEE), 8 (2021)
12. Carvalho, E.D., Silva, R.R.V., Araújo, F.H.D., de R.A.L. Rabelo, de Carvalho Filho, A.O.: An approach to the classification of covid-19 based on CT scans using convolutional features and genetic algorithms. *Comput. Biol. Med.* **136**, 104744, 9 (2021)
13. Cheng, T., Toutiao, Z., Alley, X., Fan, B.T., Wang, B.L.: Genetic constrained graph variational autoencoder for covid-19 drug discovery (2021). [arXiv:2104.11674v1](https://arxiv.org/abs/2104.11674v1)
14. Dalloccchio, R.N., Densi, A., De Vito, A., Delogu, G., Serra, P.A., Madeddu, G.: Early combination treatment with existing HIV antivirals: an effective treatment for covid-19? *European Rev. Med. Pharmacol. Sci.* **25**, 2435–2448 (2021)
15. D'angelo, G., Scoppettuolo, M.N., Anna, L.C., Rosati, A., Palmieri, F.: A genetic programming-based approach for classifying pancreatic adenocarcinoma: the sliced experience. *Soft Comput.* 1–12, 7 (2022)
16. Day, J.O., Smith, S., Noyce, A.J., Alty, J., Jeffery, A., Chapman, R., Carroll, C.: Challenges of incorporating digital health technology outcomes in a clinical trial: experiences from PD STAT. *J. Parkinson's Dis.* **12**, 1605–1609, 1 (2022)
17. de Falco, I., Cioppa, A.D., Scafuri, U., Tarantino, E.: Differential evolution to estimate the parameters of a seir model with dynamic social distancing: the case of covid-19 in Italy. *Data Science for COVID-19 Volume 1: Computational Perspectives*, pp. 75–90, 1 (2021)
18. de Freitas Barbosa, V.A., Gomes, J.C., de Santana, M.A., de Albuquerque, J.E.A., de Souza, R.G., de Souza, R.E., dos Santos, W.P.: Heg.ia: an intelligent system to support diagnosis of covid-19 based on blood tests. *Res. Biomed. Eng.* **38**, 99–116, 3 (2022)
19. Dehsarvi, A., Palomares, J.K.S., Smith, S.L.: Towards automated monitoring of parkinson's disease following drug treatment. In: Yacoubi, M.E., Granger, E., Yuen, P.C., Pal, U., Vincent, N. eds.: *Pattern Recognition and Artificial Intelligence*, Cham, pp. 196–207. Springer International Publishing (2022)
20. Dixit, A., Mani, A., Bansal, R.: Cov2-detect-net: Design of covid-19 prediction model based on hybrid DE-PSO with SVM using chest x-ray images. *Inf. Sci.* **571**, 676–692, 9 (2021)
21. Dua, D., Graff, C.: UCI machine learning repository (2017)

22. Elia, S., D'Angelo, G., Palmieri, F., Sorge, R., Massoud, R., Cortese, C., Hardavella, G., De Stefano, A.: A machine learning evolutionary algorithm-based formula to assess tumor markers and predict lung cancer in cytologically negative pleural effusions. *Soft Comput.* **24**, 7281–7293, 5 (2020)
23. De Falco, I., De Pietro, G., Sannino, G.: Classification of covid-19 chest x-ray images by means of an interpretable evolutionary rule-based approach. *Neural Comput. Appl.* 1–11, 1 (2022)
24. Fan, M., Wu, G., Cheng, H., Zhang, J., Shao, G., Li, L.: Radiomic analysis of dce-mri for prediction of response to neoadjuvant chemotherapy in breast cancer patients. *European J. Radiol.* **94**, 140–147, 9 (2017)
25. Fan, M., Xia, P., Clarke, R., Wang, Y., Li, L.: Radiogenomic signatures reveal multiscale intratumour heterogeneity associated with biological functions and survival in breast cancer. *Nat. Commun.* **11**(1), 11:1–12, 9 (2020)
26. Gao, C., Smith, S., Lones, M., Jamieson, S., Alty, J., Cosgrove, J., Zhang, P., Liu, J., Chen, Y., Juanjuan, D., Cui, S., Zhou, H., Chen, S.: Objective assessment of bradykinesia in parkinson's disease using evolutionary algorithms: clinical validation. *Trans. Neurodegen.* **7**, 18 (2018)
27. Ghosh, S., Bhattacharya, S.: A data-driven understanding of covid-19 dynamics using sequential genetic algorithm based probabilistic cellular automata. *Appl. Soft Comput.* **96**, 106692, 11 (2020)
28. Gopalakrishnan, S., Sridharan, S., Nayak, S.R., Nayak, J., Venkataraman, S.: Central hubs prediction for bio networks by directed hypergraph—GA with validation to covid-19 PPI. *Pattern Recogn. Lett.* **153**, 246–253, 1 (2022)
29. Hernandez-Vargas, E.A., Velasco-Hernandez, J.X.: In-host mathematical modelling of covid-19 in humans. *Ann. Rev. Control* **50**:448–456, 1 (2020)
30. Hughes, G.L., Lones, M.A., Bedder, M., Currie, P.D., Smith, S.L., Pownall, M.E.: Machine learning discriminates a movement disorder in a zebrafish model of parkinson's disease. *Dis. Models Mech.* **13**, dmm045815 (2020)
31. Koziot, K., Stanisławski, R., Bialic, G.: Fractional-order sir epidemic model for transmission prediction of covid-19 disease. *Appl. Sci.* **10**, 8316, 11 (2020)
32. Kwuimy, C.A.K., Nazari, F., Jiao, X., Rohani, P., Nataraj, C.: Nonlinear dynamic analysis of an epidemiological model for covid-19 including public behavior and government action. *Nonlinear Dyn.* **101**, 1545–1559, 8 (2020)
33. Rodríguez-De la Cruz, J.A., Acosta-Mesa, H.G., Mezura-Montes, E.: Evolution of generative adversarial networks using PSO for synthesis of covid-19 chest x-ray images. In: 2021 IEEE Congress on Evolutionary Computation, CEC 2021—Proceedings, pp. 2226–2233 (2021)
34. Li, D., Zhou, H., Hui, X., He, X., Mu, X.: Plasmonic biosensor augmented by a genetic algorithm for ultra-rapid, label-free, and multi-functional detection of covid-19. *Anal. Chem.* **93**, 9437–9444, 7 (2021)
35. Li, Y., Umbach, D.M., Krahn, J.M., Shats, I., Li, X., Li, L.: Predicting tumor response to drugs based on gene-expression biomarkers of sensitivity learned from cancer cell lines. *BMC Genom.* **22**, 1–18, 12 (2021)
36. Libotte, G.B., Lobato, F.S., Platt, G.M., Neto, A.J.S.: Determination of an optimal control strategy for vaccine administration in covid-19 pandemic treatment. *Comput. Methods Program. Biomed.* **196**, 105664, 11 (2020)
37. Lones, M.A., Alty, J.E., Lacy, S.E., Jamieson, S.D.R., Possin, K.L., Schuff, N., Smith, S.L.: Evolving classifiers to inform clinical assessment of parkinson's disease. In: 2013 IEEE Symposium Series on Computational Intelligence (SSCI 2013), Proceedings of the 2013 IEEE Symposium on Computational Intelligence in Healthcare and e-Health (CICARE 2013), pp. 76–82. IEEE (2013)
38. Lones, M.A., Alty, J.E., Cosgrove, J., Duggan-Carter, P., Jamieson, S., Naylor, R.F., Turner, A.J., Smith, S.L.: A new evolutionary algorithm-based home monitoring device for parkinson's dyskinesia. *J. Med. Syst.* **41**(11), 176 (2017)
39. Lones, M.A., Tyrrell, A.M.: Biomimetic representation with enzyme genetic programming. *Gen. Programm. Evol. Mach.* (2002)

40. Lones, M.A., Smith, S.L., Alty, J.E., Lacy, S., Possin, K., Jamieson, S., Tyrrell, A.M.: Evolving classifiers to recognise the movement characteristics of parkinson's disease patients. *IEEE Trans. Evol. Comput.* **18**(4), 559–576 (2014)
41. Lones, M.A., Smith, S.L., Harris, A.T., High, A.S., Fisher, S.E., Alastair Smith, D., Kirkham, J.: Discriminating normal and cancerous thyroid cell lines using implicit context representation cartesian genetic programming. In: *IEEE world congress on computational intelligence, WCCI 2010–2010 IEEE congress on evolutionary computation. CEC* (2010)
42. Lones, M.A., Smith, S.L., Tyrrell, A.M., Alty, J.E., Stuart Jamieson, D.R.: Characterising neurological time series data using biologically motivated networks of coupled discrete maps. *BioSystems* **112**(2), 94–101 (2013)
43. Louati, H., Bechikh, S., Louati, A., Aldaej, A., Said, L.B.: Evolutionary optimization of convolutional neural network architecture design for thoracic x-ray image classification. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 12798 LNAI:121–132 (2021)
44. Lu, H., Wang, H., Yoon, S.W.: A dynamic gradient boosting machine using genetic optimizer for practical breast cancer prognosis. *Exp. Syst. Appl.* **116**, 340–350, 2 (2019)
45. Luo, X., Duan, H., Xu, K.: A novel grey model based on traditional richards model and its application in covid-19. *Chaos Sol. Fract.* **142**, 110480, 1 (2021)
46. Luong, N.H., Alderliesten, T., Pieters, B.R., Bel, A., Niatsetski, Y., Bosman, P.A.N.: Fast and insightful bi-objective optimization for prostate cancer treatment planning with high-dose-rate brachytherapy. *Appl. Soft Comput.* **84**, 105681, 11 (2019)
47. Maleki, N., Zeinali, Y., Niaki, S.T.A.: A k-nn method for lung cancer prognosis with the use of a genetic algorithm for feature selection. *Exp. Syst. Appl.* **164**, 113981, 2 (2021)
48. Matabuena, M., Rodríguez-Mier, P., García-Meixide, C., Leborán, V.: Covid-19: estimation of the transmission dynamics in spain using a stochastic simulator and black-box optimization techniques. *Comput. Methods Program. Biomed.* **211**, 106399, 11 (2021)
49. Miikkulainen, R., Francon, O., Meyerson, E., Qiu, X., Sargent, D., Canzani, E., Hodjat, B.: From prediction to prescription: Evolutionary optimization of nonpharmaceutical interventions in the covid-19 pandemic. *IEEE Trans. Evol. Comput.* **25**, 386–401, 4 (2021)
50. Milani, A., Biondi, G.: Spatial assignment optimization of vaccine units in the covid-19 pandemics. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 12955 LNCS:448–459, 9 (2021)
51. Miller, J.F., Thomson, P.: Cartesian genetic programming. In: Poli, R., Banzhaf, W., Langdon, W.B., Miller, J., Nordin, P., Fogarty, T.C. (eds.), *Genetic Programming, Proceedings of EuroGP 2000*, Berlin, Heidelberg, pp. 121–132. Springer, Berlin, Heidelberg (2000)
52. Miralles-Pechuán, L., Jiménez, F., Ponce, H., Martínez-Villaseñor, L.: A methodology based on deep q-learning/genetic algorithms for optimizing covid-19 pandemic government actions. In: *International Conference on Information and Knowledge Management, Proceedings*, pp. 1135–1144, 10 2020
53. Motamed, P.K., Maftoon, N.: A systematic approach for developing mechanistic models for realistic simulation of cancer cell motion and deformation. *Sci. Rep.* **11**:1–18, 11 (2021)
54. Neto, O.P., Kennedy, D.M., Reis, J.C., Wang, Brizzi, Y.A.C.B., Zambrano, G.J., de Souza, J.M., Pedroso, W., de Mello Pedreiro, R.C., de Matos Brizzi, B., Abinader, E.O., Zângaro, R.A.: Mathematical model of covid-19 intervention scenarios for São Paulo-Brazil. *Nat. Commun.* **12**:1, 12:1–13, 1 (2021)
55. Niazkar, M., Niazkar, H.R.: Covid-19 outbreak: Application of multi-gene genetic programming to country-based prediction models. *Electron. J. Gen. Med.* **17**, em247, 5 (2020)
56. Olivier, L.E., Botha, S., Craig, I.K.: Optimized lockdown strategies for curbing the spread of covid-19: a South African case study. *IEEE Access* **8**, 205755–205765 (2020)
57. Panigrahi, S.S., Muthukumar, A.J., Thangavelu, S., Jeyakumar, G., Velayutham, C.S.: A comparative study on parameter estimation of covid epidemiological models using differential evolution algorithm. *Stud. Comput. Intell.* **1009**, 241–263 (2022)
58. Panjwani, B., Singh, V., Rani, A., Mohan, V.: Optimum multi-drug regime for compartment model of tumour: cell-cycle-specific dynamics in the presence of resistance. *J. Pharm. Pharmacodyn.* **48**, 4, 48:543–562, 3 (2021)

59. Paruch, M.: Identification of the degree of tumor destruction on the basis of the arrhenius integral using the evolutionary algorithm. *Int. J. Thermal Sci.* **130**, 507–517, 8 (2018)
60. Quaranta, G., Formica, G., Machado, J.T., Lacarbonara, W., Masri, S.F.: Understanding covid-19 nonlinear multi-scale dynamic spreading in Italy. *Nonlinear Dyn.* **101**, 1583–1619, 8 (2020)
61. Ranieri, C.M., Pimentel, J.M., Romano, M., Elias, L., Romero, R.A.F., Lones, M.A., Araujo, M.F.P., Vargas, P.A., Moioli, R.C.: A data-driven biophysical computational model of parkinson's disease based on marmoset monkeys. *IEEE Access* **9**, 122548–122567 (2021)
62. Sadowski, K.L., Alderliesten, T., Niatsetski, Y., Van Der Meer, M.C., Thierens, D., Bel, A., Luong, N.H., Van Der Laarse, R., Bosnian, P.A.N.: Exploring trade-offs between target coverage, healthy tissue sparing, and the placement of catheters in HDR brachytherapy for prostate cancer using a novel multi-objective model-based mixed-integer evolutionary algorithm. In: GECCO 2017—Proceedings of the 2017 Genetic and Evolutionary Computation Conference, pp. 1224–1231, 7 (2017)
63. Salgotra, R., Gandomi, M., Gandomi, A.H.: Evolutionary modelling of the covid-19 pandemic in fifteen most affected countries. *Chaos Sol. Fract.* **140**, 110118, 11 (2020)
64. Salgotra, R., Gandomi, M., Gandomi, A.H.: Time series analysis and forecast of the covid-19 pandemic in India using genetic programming. *Chaos Sol. Fract.* **138**, 109945, 9 (2020)
65. Shaban, W.M., Rabie, A.H., Saleh, A.I., Abo-Elsoud, M.A.: A new covid-19 patients detection strategy (cpds) based on hybrid feature selection and enhanced KNN classifier. *Knowl.-Based Syst.* **205**, 106270, 10 (2020)
66. Shamshad, F., Khan, S., Zamir, S.W., Khan, M.H., Hayat, M., Khan, F.S., Fu, H.: Transformers in medical imaging: a survey. *Med. Image Anal.* 102802 (2023)
67. Shindi, O., Kanesan, J., Kendall, G., Ramanathan, A.: The combined effect of optimal control and swarm intelligence on optimization of cancer chemotherapy. *Comput. Methods Program. Biomed.* **189**, 105327, 6 (2020)
68. Shukla, P.K., Sandhu, J.K., Ahirwar, A., Ghai, D., Maheshwary, P., Shukla, P.K.: Multiobjective genetic algorithm and convolutional neural network based covid-19 identification in chest x-ray images. *Math. Probl. Eng.* (2021)
69. Singh, D., Kumar, V., Kaur, M., Jabarulla, M.Y., Lee, H.N.: Screening of covid-19 suspected subjects using multi-crossover genetic algorithm based dense convolutional neural network. *IEEE Access* **9**, 142566–142580 (2021)
70. Smith, S.L., Lones, M.A., Bedder, M., Alty, J.E., Cosgrove, J., Maguire, R., Pownall, M., Ivaniou, D., Lyle, C., Cording, A., Elliott, C.J.H.: Computational approaches for understanding the diagnosis and treatment of parkinson's disease. *IET Syst. Biol.* **9**(6):226–233 (2015)
71. Stillman, N.R., Balaz, I., Tsompanas, M.A., Kovacevic, M., Azimi, S., Lafond, S., Adamatzky, A., Hauert, S.: Evolutionary computational platform for the automatic discovery of nanocarriers for cancer treatment. *NPJ Comput. Mater.* **7**, 1–12, 9 (2021)
72. Su, Y., Li, S., Zheng, C., Zhang, X.: A heuristic algorithm for identifying molecular signatures in cancer. *IEEE Trans. Nanobiosci.* **19**, 132–141, 1 (2020)
73. Tang, L., Li, Y., Bai, D., Liu, T., Coelho, L.C.: Bi-objective optimization for a multi-period covid-19 vaccination planning problem. *Omega* **110**, 102617, 7 (2022)
74. Taou, N., Lones, M.: Optimising boolean synthetic regulatory networks to control cell states. *IEEE/ACM Trans. Comput. Biol. Bioinf.* **18**, 2649–2658 (2021)
75. Tian, Y., Feng, Y., Wang, C., Cao, R., Zhang, X., Pei, X., Tan, K.C., Jin, Y.: A large-scale combinatorial many-objective evolutionary algorithm for intensity-modulated radiotherapy planning. *IEEE Trans. Evol. Comput.* (2022)
76. Tsompanas, M.A., Bull, L., Adamatzky, A., Balaz, I.: In silico optimization of cancer therapies with multiple types of nanoparticles applied at different times. *Comput. Methods Program. Biomed.* **200**, 105886, 3 (2021)
77. Vallejo, M., Cosgrove, J., Alty, J.E., Jamieson, S., Smith, S.L., Corne, D.W., Lones, M.A.: A multi-objective approach to predicting motor and cognitive deficit in parkinson's disease patients. In: Proceedings of Genetic and Evolutionary Computation Conference (GECCO 2016), Workshop on Medical Applications (MedGEC), pp. 1369–1376. ACM (2016)

78. Vieira, P.A., Magalhães, D.M.V., Carvalho-Filho, A.O., Veras, R.M.S., Rabêlo, R.A.L., Silva, R.R.V.: Classification of covid-19 in x-ray images with genetic fine-tuning. *Comput. Electr. Eng.* **96**, 107467, 12 (2021)
79. Walker, J.A., Völk, K., Smith, S.L., Miller, J.F.: Parallel evolution using multi-chromosome cartesian genetic programming. *Gen. Programm. Evol. Mach.* **10**, 417–445, 12 (2009)
80. Wang, P., Liu, R., Jiang, Z., Yao, Y., Shen, Z.: The optimization of combination chemotherapy schedules in the presence of drug resistance. *IEEE Trans. Autom. Sci. Eng.* **16**, 165–179, 1 (2019)
81. Worzel, W.P., Yu, J., Almal, A.A., Chinnaiyan, A.M.: Applications of genetic programming in cancer research. *Int. J. Biochem. Cell Biol.* **41**, 405–413, 2 (2009)
82. Wu, P., Wang, D.: Classification of a DNA microarray for diagnosing cancer using a complex network based method. *IEEE/ACM Trans. Comput. Biol. Bioinf.* **16**, 801–808, 5 (2019)
83. Yarsky, P.: Using a genetic algorithm to fit parameters of a covid-19 SEIR model for us states. *Math. Comput. Simul.* **185**, 687–695, 7 (2021)
84. Yousefpour, A., Jahanshahi, H., Bekiros, S.: Optimal policies for control of the novel coronavirus disease (covid-19) outbreak. *Chaos Solit. Fract.* **136**, 109883, 7 (2020)
85. Zhang, G., Liu, X.: Prediction and control of covid-19 spreading based on a hybrid intelligent model. *PLOS One* **16**, e0246360, 2 (2021)
86. Zhang, M., Li, H., Pan, S., Lyu, J., Ling, S., Su, S.: Convolutional neural networks-based lung nodule classification: a surrogate-assisted evolutionary algorithm for hyperparameter optimization. *IEEE Trans. Evol. Comput.* **25**, 869–882, 10 (2021)

Chapter 21

Evolutionary Machine Learning for Space



Moritz von Looz, Alexander Hadjiivanov, and Emmanuel Blazquez

Abstract The Venn diagram of evolutionary computation, machine learning and space applications shows some intriguing overlaps. As evolutionary algorithms are often resource-intensive, they have not yet been applied *in space*. Nevertheless, it has been decisively demonstrated that evolutionary machine learning (EML) is a valuable tool *for space*, specifically in fields such as trajectory optimisation, optimal control and neuroevolution for robot control, where high-dimensional, discontinuous, sparse and/or non-linear problems abound. In the following chapter, we introduce common problems faced by the space research and application community, together with EML techniques used for generating robust, performant and, sometimes indeed, state-of-the-art solutions. The often complex mathematics behind some problems (especially in trajectory optimisation and optimal control) has been simplified to the minimum necessary to convey the essence of the challenge without encumbering the overview of the relevant EML algorithms. We hope that this chapter provides useful information to both the EML and the space communities in the form of algorithms, benchmarks and standing challenges.

21.1 Introduction

The intersection between evolutionary methods and classical machine learning is extensive and varied [55]. Both approaches have found many applications in the space domain individually: evolutionary methods for interplanetary trajectory optimisation, recurrent neural network architectures for guidance, navigation and control (GNC) tasks, neurocontrollers for in-space robotics or convolutional neural networks for image recognition in earth observation (EO) and astronomy. One of the earliest demonstrations of a real-world application of evolution was an antenna designed by a genetic programming algorithm [34] for NASA's Space Technology 5 mission. The performance of the final design was found to be comparable to hand-crafted

M. von Looz (✉) · A. Hadjiivanov · E. Blazquez
European Space Agency ESA, Keplerlaan 1, 2201 AZ Noordwijk, Netherlands
e-mail: moritz@vlooz.de

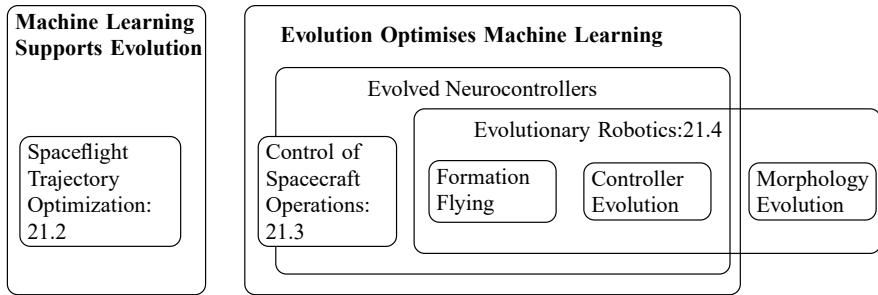


Fig. 21.1 EML for space applications considered in this chapter

antennae. It was evaluated on an actual satellite, and a subsequent study successfully re-evolved the design [3] to comply with changes in the requirements.

An important distinction is that the space domain is not restricted to applications *physically* in space (generally known as *on-board* applications). Traditionally, requirements in terms of robustness and reliability for any physical device deployed in space have been, and remain, notoriously high. This is due to the stringent safety qualification and performance requirements, which in turn stem from the harsh environment and high cost of failures. Furthermore, due to the limited power budget, computational capabilities and support for parallelisation of spaceflight on-board hardware and software, the use of evolutionary algorithms is currently limited to on-ground pre- and post-processing.

Still, just like life on Earth, which flourishes even in the most inhospitable environments thanks to biological evolution, evolutionary machine learning (EML) has found its niche applications in space-related research. Specifically, EML for space applications is an optimisation paradigm that is growing in popularity and can benefit from everything that evolution has to offer, such as tackling large and discontinuous fitness landscapes, as well as robustness to local minima.

The applications of EML methods in the space sector can be broadly grouped into two categories, see Fig. 21.1. The first one is the use of machine learning methods for assisting evolution: Evolutionary methods have been highly successful in interplanetary trajectory design [29], and we will discuss potential applications of EML for trajectories in Sect. 21.2.

The second area of intersection is classical machine learning (most commonly artificial neural networks) assisted by evolutionary methods. This takes the form of neural architecture search, genetic programming or direct optimisation of network weights. Many guidance and path planning tasks in space can be mathematically framed as control problems and addressed with neurocontrollers, as we will discuss in Sect. 21.3. This applies specifically to robotic planetary exploration, see Sect. 21.4.

It is important to note that vision-based object classification is a major task in astronomy, and the field has embraced evolutionary machine learning methods for this purpose [16, 24]. One particular challenge is the identification of objects in the presence of stray light sources and diffusion. For instance, Jones et al. [30] use an

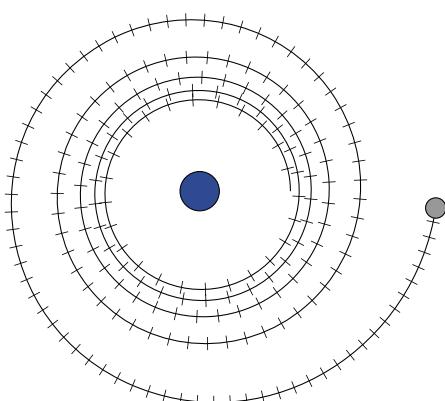
evolutionary approach to neural architecture search, in which they evolve parameters specifying the topology of a convolutional neural network tasked with identifying galaxies in the Zone of Avoidance (the part of the universe that is obscured by the galactic disk of the Milky Way). However, we choose to direct our focus to spaceflight applications in the remainder of this chapter as this matches our area of expertise. Below, we present several categories of such applications, together with examples and future directions in Sect. 21.5.

21.2 Spaceflight Trajectory Optimisation

Among the applications of evolutionary algorithms for spaceflight, the optimization of trajectories is by far the most successful. Numerous winning entries in the series of the Global Trajectory Optimisation Competition (GTOC) have relied primarily on evolutionary algorithms, occasionally augmented with machine learning techniques.

In the absence of other forces, a spacecraft will follow a ballistic, predictable arc. A trajectory is thus characterised by the timing and nature of its *manoeuvres*, either using some propulsion system or gravitational interactions. Propulsion systems can be classified by their *exhaust velocity* and *thrust*. A higher exhaust velocity makes the system more efficient, and larger manoeuvres can be done with the same amount of fuel. A higher thrust allows manoeuvres to be done in shorter time, making their planning and modelling much easier. For classical, *chemical* engines, the thrust is high enough that on the time scales for interplanetary trajectories, manoeuvres can be modelled as instantaneous. In contrast, *low-thrust* engines, for example, ion drives and other sorts of electrical propulsion, are using their fuel much more efficiently, but manoeuvres take weeks or months [59]. Spacecraft with low-thrust engines effectively accelerate continuously on a large part of their trajectory, travelling on long spirals instead of ballistic arcs. Figure 21.2 shows an example of a low-thrust trajectory from an Earth parking orbit to the moon.

Fig. 21.2 An illustration of a low-thrust trajectory, inspired by the SMART-1 technology demonstrator mission to the moon. The continuous thrust results in trajectories with a large number of free variables, illustrated by the ticks. (Not to scale)



The key figure of merit of a trajectory is the total *change of velocity* required to be fulfilled by its propulsive manoeuvres.¹ It is denoted as ΔV , and the fuel requirements of a mission scale exponentially with it

$$m_{\text{fuel}} = (e^{\Delta V / v_{\text{exhaust}}} - 1) \cdot m_{\text{dry}}. \quad (21.1)$$

Equation 21.1 stems from rearranging the famous Tsiolkovsky's rocket equation, with m_{fuel} denoting the fuel mass, v_{exhaust} the exhaust velocity of the propulsion system and m_{dry} the mass of everything else: the payload, but also the weight of the propulsion system and empty tanks. Due to the exponential growth, fuel requirements quickly become prohibitive for higher values of ΔV . Designing efficient trajectories is thus imperative not only to lower the cost of a mission but also to make it possible at all.

A basic building block of efficient trajectory design is the use of *gravity assist* manoeuvres (GAs). In these, a spacecraft flies past a moving massive body, and exchanges momentum due to the gravitational interaction. Multiple flybys can be chained together to increase the saved ΔV , but require precise timing. This technique, denoted as *Multiple Gravity Assists* (MGA), enabled the Grand Tour of Voyager 2 [31] and has become nearly indispensable in modern interplanetary missions. Figure 21.3 shows an example trajectory with four gravity assists to reach Jupiter.

The parameterisation of a trajectory can be conceptually reduced to the epochs, magnitudes and directions of its manoeuvres, including gravity assists. For *impulsive* trajectory designs making use of chemical engines, the arcs between manoeuvres are ballistic, see also Fig. 21.3. For low-thrust trajectories with continuous acceleration, the continuous-time optimisation problem is first discretised and free parameters, such as thrust angles and epochs, are specified for each discretisation node, as shown in Fig. 21.2.

The resulting optimisation problems are high-dimensional, even more so for low-thrust trajectories. As the effect of a gravity assist depends on precise timing, the cost functions have steep gradients and many local optima which makes them challenging to optimise. Analytical gradients are not always available and local optimisation relies heavily on the quality of an initial estimate. Evolutionary algorithms have thus been used extensively and with considerable success to initialise global trajectory optimisation searches [29, 59]. In these, the genotype specifies the trajectory parameters, while the fitness is set to its cumulative ΔV (and thus the logarithm of fuel use) over the entire trajectory.

Multiple approaches have been developed to accelerate the evolutionary trajectory search. Izzo et al. [27] introduce the *archipelago* model to make use of parallel computing resources. In this model, multiple populations of solutions are evolved simultaneously with regular migration. In addition to allowing a higher degree of parallelisation, this approach prevents the evolution from getting stuck in local minima. Some trajectory problems have combinatorial aspects in addition to the continuous

¹ Other considerations include time of flight, radiation load, timing of manoeuvres and targets of opportunity.

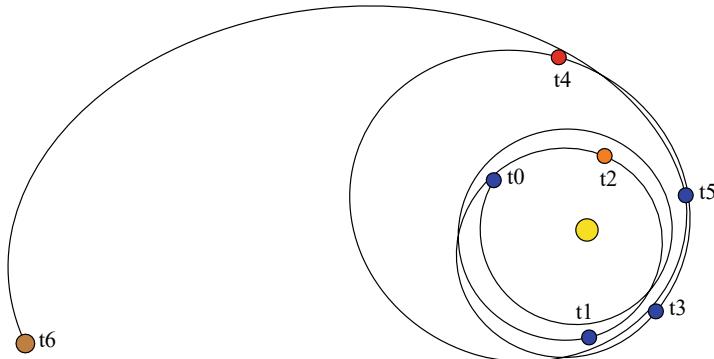


Fig. 21.3 A trajectory using multiple gravity assists on Earth, Mars and Venus on the way to Jupiter. Each coloured dot marks the location of a planet during a gravity assist manoeuvre. The Earth is visited multiple times. The trajectory is inspired by ESA's JUICE mission, launched in 2023 and currently en route to the Jupiter system. In this parametrisation, the free parameters are only the times t_0 – t_6 : the timings of launch, the flybys and the arrival

ones. Examples include the right order of planets to visit for flybys or the creation of tours covering multiple moons or asteroids. Well-known instances include the *Global Trajectory Optimisation Challenge* (GTOC), a regular competition of complex trajectory problems. The seventh GTOC, for example, challenged participants to explore the asteroid belt with multiple craft, while the ninth GTOC posed the problem of selecting, visiting and removing pieces of space debris with a limited fuel budget [25]. More applied mission proposals also face the problem of target selection and visitation orders [47].

21.2.1 Use of Machine Learning Methods

Due to the inherent randomness of evolutionary optimisation and the many local minima prevalent in trajectory problems, it is beneficial to perform multiple restarts of the optimisation process, starting from different seeds to have a higher chance of reaching a global optimum. Due to this process, but also during a single evolutionary run, large amounts of data about the fitness landscape are gathered.

Multiple machine learning approaches have been developed to take advantage of this data to learn the fitness landscape. Cassioli et al. [8] use a Support Vector Machine (SVM) to classify starting points by whether Monotonic Basin Hopping (MBH) will find a good enough solution if starting from that point. They test this approach on a set of trajectory problems from ESA's Advanced Concepts Team (ACT) [56] and report a reduction of total fitness evaluations by about a third. More recently, *Estimation of Distribution Algorithms* (EDAs) [35] have seen some applications to trajectory design, in particular to low-thrust scenarios. EDAs refer to a

class of evolutionary algorithms that makes use of probabilistic models to detect the most promising chromosomes in a population and evolve it. Shirazi et al. [50] applied an enhanced EDA algorithm based on Gaussian distribution learning to solve time-varying Lyapunov control problems for low-thrust transfers in the Earth environment, showing the promise of these techniques in obtaining feasible near-optimal solutions with reasonable performance while offering insight into the probabilistic fitness landscape of the problem at hand. In regards to target selection, Choi et al. [9] target six multiple gravity assist (MGA) trajectory optimisation problems (including the 1st GTOC) as benchmarks for a novel adaptive differential evolution (DE) method, which relies on selecting the most performant strategy from a pool of EAs (DE/rand/1, DE/rand/2, DE/best/2, DE/current-to-best/1, DE/current-to-pbest/1 and DE/current-to-opposition/1) at runtime. Specifically, the adaptive DE method involves recording two extra pieces of information for each individual in the population: the current strategy being used and the time since the last fitness update. If the individual is stagnant, its strategy is updated with a random one from the pool of algorithms, and the best-performing strategies become increasingly likely to be selected as a result of applying the Cauchy distribution method in [10]. Choi et al. [9] report that their DE method matched the currently known best results for two problems and improved on the state of the art for the remaining four problems.

21.2.1.1 Surrogate Models

An alternative approach is to train a model to approximate the fitness function directly, commonly known as *surrogate model*. For instance, Ampatzis et al. [1] describe a surrogate model for Multiple Gravity Assist problems. They use a feed-forward network with two hidden layers of 25 neurons each, with the logistic and tangent activation functions used in the first and second hidden layers, respectively. With this, they find that apart from the strict performance benefit, using a surrogate function also improves the evolutionary process by smoothing out the rugged fitness landscape. The training data for the surrogate model comes from the evolved population itself. In order to address the trade-off between the actual and the surrogate models in terms of the number of fitness evaluations, Ampatzis et al. [1] use a fixed number of generations and alternate evolving the population with the true model and the surrogate model. They use Differential Evolution (DE) [45] for this optimisation and find that on problems such as recreating the Cassini mission trajectory [44], this hybrid approach needs in expectation as many generations as the non-surrogate version, but with much faster function evaluations in generations using the surrogate model.

Stubbig et al. [54] train a three-layer ReLU network as surrogate function for low-thrust trajectories in the hodographic shaping method. They find that feature engineering is relevant, as including more, even redundant information in the state vectors increases the model accuracy.

Hennes et al. [20] develop fast approximators for low-thrust transfers in the main asteroid belt. Low-thrust transfers are expensive to compute and the number of pos-

sible transfers is large in a dense asteroid population, it thus becomes necessary to quickly decide which of them to compute in detail. Approximating low-thrust transfers as ballistic (Lambert) transfers with impulsive manoeuvres is fast but lacks accuracy. The authors find that most common machine learning methods outperform the ballistic approximation, with gradient-boosted decision trees performing best, but all need the right input features. These include the phasing, defining how far along each asteroid is in its orbit. Crucially, the machine learning methods perform best if the ballistic approximation itself is also part of the input features.

Merata et al. [38] extend this approach to Near-Earth Asteroids (NEA), which are closer to the sun than the main belt asteroids and thus have shorter orbital periods. The transfers may thus span multiple orbital periods and the phasing indicators become meaningless. The authors generate a database of 60000 Optimal Control Problem (OCP) descriptions of near-earth asteroid transfers and train an ensemble of machine learning methods. They find that feature selection of the right astrodynamical variables (time of transfer and differences in orbital radius and inclination) is critical for good performance.

21.2.1.2 Parameter Learning

The behaviour of evolutionary algorithms is commonly governed by parameters, whose tuning can have a marked effect on the algorithm's performance. Within evolutionary approaches, Omran et al. [40] use *self-adaptation* to great advantage in differential evolution and Zuo et al. [61] build on that success with a technique they name *case learning*: Keeping a reservoir of mutation and crossover parameters that resulted in improved offspring within differential evolution.

21.3 Optimal Control of Spacecraft Operations

Optimal control refers to the methodology of finding, given a dynamical system and an associated figure of merit defined as the *cost*, a series of control inputs over a period of time that results in the minimisation or maximisation of said cost. A challenge in many control problems, especially for spacecraft applications, is that designing robust and highly performant near-optimal controllers comes at the expense of thorough system identification and computationally expensive open- and closed-loop simulations. Rephrased more generally in the reinforcement learning literature, optimal control is akin to an agent perceiving an environment and selecting actions in order to maximise a reward. Actions are selected according to a *policy*, commonly implemented as a neural network. This can be done by training estimators for the expected value of actions and states, as is done with Q-learning or actor–critic networks, or by optimising the network weights of a policy network directly. The latter strategy is called *direct policy search*. Most neuroevolution methods used for optimal control correspond to gradient-free direct policy search. In recent years, *neurocon-*

trollers have appeared as potential solutions for real-time on-board control of space systems. Neuro-controllers are typically defined as neural network mapping a series of system state inputs to control action outputs with the goal to replace traditional controller design with neural architectures.

For example, Leitner et al. [32] design a neurocontroller for autonomous rendezvous and docking between two spacecraft. It is assumed that only one spacecraft is controllable in translation and attitude with thruster actuators controlled via feed-forward neural networks. In a two-dimensional dynamical system, the controlling network receives position, speed, attitude and angular velocity, for a total of six inputs, and produces two outputs that directly control two thrusters. The overall network is relatively small, consisting of 30 neurons and 103 weights, which are optimised using a simple genetic algorithm from randomly selected starting points. Leitner et al. [32] find that the neurocontroller uses more fuel and time than a numerical optimal control strategy, but can deal better with unexpected changes to the dynamics. They refer to this trade-off as *the price of robustness*.

Dachwald [11] consider very-low-thrust trajectories suitable for solar sails or nuclear electric propulsion, with thrusts in the range of mN/kg . They discretise the trajectory, with the thrust angle and magnitude used as free parameters at each time step. Then, instead of optimising the trajectory directly, they pose the control problem as a reinforcement learning problem and design a neural network (neurocontroller) to take the current state as input and return the thrust as output. They then optimise the network weights via direct policy search.

Willis et al. [57] design a neurocontroller for hovering over a non-spherical asteroid. A particular challenge in navigation around small bodies is their weak gravitational field, leading to a relatively stronger effect of inhomogeneities, solar radiation pressure and other perturbative forces. They investigate a sensor setup without direct distance measurements, instead using optical flow to estimate the ratio of relative velocity to distance from the surface. They then propose a fixed two-layer feed-forward neural network to minimise the offset in each direction independently and optimise the weights with direct policy search using generational particle swarm optimisation (PSO) on an archipelago setup. Multiple populations are evolved independently with regular migration, for a total population of 504 individuals over 1000 generations. They find that, although the lack of absolute distance measurements has a negative effect on the control performance, the evolutionary approach yields significantly better hovering controllers than previous work.

Yang et al. [33] apply a neural guidance scheme using an artificial neural network to control thrust vector steering during low-thrust transfers in the Earth environment, using a Lyapunov control scheme and an improved cooperative evolutionary algorithm to evolve the parameters of the network. The proposed solution shows reasonable accuracy in the satisfaction of the constraints of the corresponding optimal control problem, with the potential of offering an on-board autonomous guidance solution. The robustness of the proposed architecture and its integration within a complete GNC solution still remain open questions.

Marchetti et al. [36] design a hybrid control scheme in which they first find a control law using genetic programming, then optimise it (possibly online) with

artificial neural networks. The coefficients in the evolved control law and the weights of control variables are first optimised with classical approaches (Broyden–Fletcher–Goldfarb–Shanno and Nelder–Mead) with a set of random disturbances. The reference trajectories together with the optimised scalars are used as training data for neural networks. They test this approach on a control task of a single-stage-to-orbit transfer vehicle with random disturbances. Evaluating the neural networks trained on the optimised trajectories yields a success ratio (being within 1% of the reference trajectory) of roughly 65–85%.

Zhang et al. [60] survey different combinations of genetic programming and machine learning approaches in the context of job shop scheduling. Some of them, for example, surrogate functions, are also widely used within evolutionary machine learning for space. In their case, the surrogate models consist of equivalent, but smaller scheduling problems. While genetic programming approaches for space applications are currently a niche application, many of the assistive techniques mentioned by Zhang et al. [60] (for example, feature selection) are likely to be beneficial as well.

21.4 Evolutionary Robotics

In the specific context of space, there are a number of confounding factors that robotics design needs to take into account. Deep space is an exceedingly harsh environment characterised by high vacuum, strong ionising radiation and fluctuating extreme temperatures. Similarly, when considering planetary exploration, there can be a number of complicating factors, such as low or high gravitational pull, broken terrain characteristics and a corrosive, abrasive or pressurised environment. None of these factors are specific to evolutionary robotics (ER)—indeed, they must be dealt with by *any* algorithm used for optimising a robot’s morphology or behaviour. In this section, rather than a thorough overview of the fundamentals of ER, we present approaches that take into consideration at least some of the constraints associated with robotic space exploration. We therefore introduce some insight into morphology evolution (ME) and controller evolution (CE) within a space context, followed by an overview of self-assembly and reconfiguration, which are more niche but prospective, realistic future space applications subject to harsh environmental constraints.

21.4.1 Morphology Evolution

Morphology evolution (ME) focuses on evolving the shape, propulsion mode, materials and other physical aspects of the robot. In ME, the controller is usually fixed, whereas controller evolution (CE) focuses on evolving the robot’s behaviour while keeping its morphology fixed (or at least that the *type* of control is fixed, in other words, if the mode of propulsion is bipedal motion, it is guaranteed that the controller

is not going to have to deal with jet-based propulsion). Morphology/controller co-evolution is a combination of both: it is a manifestation of the philosophy of *embodied intelligence* [21, 22]—the idea that true intelligence can only be achieved with a physical embodiment. CE is conceptually and practically very different from ME—while robot morphology can be evolved in simulation, there is significant overhead associated with the production and testing of the final design, as well as a limitation on the number of components and their types. In contrast, evolved controllers can be readily transferred to actual hardware and tested immediately, leading to a much faster prototyping cycle.

Studies on ME that take into account the fact that the robot will be operating in space are few and far between. In one example, Rommerman et al. [46] apply Covariance Matrix Adaptation—Evolutionary Strategy (CMA-ES) [19] to the morphology evolution of a crawling robot. However, CMA-ES itself does not consider any space-related constraints; instead, it is the base robotic platform (ARAMIES [53]) that makes the results relevant to space applications. Indeed, the ARAMIES robot was designed specifically for moving over rough terrain at steep inclinations, with high robustness, ease of maintenance and long mean time between failures. Other studies have focused on evolving specific subsystems, such as active vision [7, 42, 43] in a simulated rover, or behaviour, such as trajectory optimisation [11, 12] or path planning behaviour [51] for a swarm of rovers. In general, ME for robots designed specifically for space exploration suffers from the issue of *unknown unknowns*—sets of physical parameters that are too poorly understood to be used as constraints or objectives. This is not a limitation of ER *per se*—rather, it is the result of our limited understanding of and inability to reproduce the environments in which the evolved robots might operate, leading to a degree of uncertainty that is currently too high for ME to be applied to real-world optimisation tasks.

21.4.2 Controller Evolution

A relevant example of CE for space applications deals with the feasibility of bio-inspired design specifically targeting deployment on Mars as presented in [14]. The study considers realistic constraints associated with the particular environment of its intended operation, such as communication, temperature range, batteries, landing site and even soil composition. Consequently, Ellery et al. [14] consider the versatility of different propulsion techniques and adopt an insect-like hexapodal structure as well as other insect-like qualities such as perceptual, behavioural and functional. A noteworthy feature of the Mars Walker includes the implementation of *reflex motions*, namely the *searching* reflex and the *elevator* reflex [15], which counteract external perturbations by activating when obstacles or gaps are encountered during regular locomotion. Critically, lessons learned from previous missions to Mars (such as opportunity becoming stuck on relatively flat terrain) as well as the particular operating conditions (e.g., a limited power budget) are translated into requirements for the *controller*, namely, energy efficiency, ability to cover large gaps, robustness

and stability. Therefore, Ellery et al. [14] opt for a model of the stick insect as a natural match for the insect-like body design of the Mars Walker. The weights of the neural networks designed as controllers for each of the hexapod’s legs are optimised using the island model [23], which has emerged as a useful tool for evolutionary optimisation in cases that involve multiple constraints, and in particular in studies on CE for neurocontrollers in the context of designing robots for space exploration. In the case of the Mars Walker, the controller is modelled as a continuous-time recurrent neural network (CTRNN) with weights evolved with coevolutionary distributed GA. Ellery et al. [14] use the Open Dynamics Engine (ODE) simulator [52] to evaluate the robot’s performance. The same approach is also taken by Peniak et al. [42], where the objective is developing an active vision system for obstacle avoidance in unknown environments, targeting a simulated version of a Mars rover. A dedicated study on the island model for multi-agent CE scenarios is given in [7].

Ampatzis et al. [2] consider a robot self-assembly task and investigate the role of isolated populations and migrations for the evolution of neurocontrollers. They use Continuous-Time Recurrent Neural Networks (CTRNNs) of 24 neurons in total and optimise their weights with direct policy search using differential evolution on an archipelago with 10 islands, both without migration and a ring topology. They find that migration has a strong effect on the final fitness: Without migration, only 80% of the maximum fitness is reached, even with a higher number of generations. Peniak et al. [41] use evolutionary search to design neurocontrollers for autonomous planetary rovers. The controllers are implemented as neural networks and receive as input a simple sensory system that can serve as a backup in case 3D vision becomes unavailable. The evolutionary search is a simple genetic algorithm implemented on the archipelago framework, with 9 islands of 10 individuals each. A virtual version of the Mars Science Laboratory (MSL) rover is used in a virtual environment as the base for the simulation. The authors find the island model to be successful for evolving neurocontrollers in this framework.

In a work of de Croon et al. [13], a robot controller is evolved with the objective of odour-based localisation of a simulated methane plume on Mars in the presence of wind with low or high turbulence. They consider a robot equipped with a single chemical sensor and a single wind sensor, and train a CTRNN architecture consisting of 4 input neurons (whose inputs are computed from the two simulated sensors), 10 hidden neurons and 4 output neurons. A population of 30 individuals is used, with 1 elite individual and 6 parents selected on a roulette wheel principle. Crossover is carried out with a probability of 0.1, Gaussian mutation with a probability of 0.03 per gene. They use the simple genetic algorithm implementation from the PyGMO/PAGMO platform [4] with direct policy search, where the objective is to locate and approach the methane source. An interesting approach taken in the study is to interpret the evolved policies as finite-state machines, thus distilling an algorithm from the network behaviour. Arguably, the ability to translate a more or less ‘opaque’ neural model into a highly explainable and convenient symbolic or algorithmic representation for human mission analysts is in fact one crucial step towards the more widespread adoption of evolutionary methods in space-related research involving any form of control.

A key takeaway is that the main differentiator in the design of ME and CE for robots deployed in space is whether the design process considers specific factors of the mission or the environment as either constraints or objectives. The following is a non-exhaustive list of factors to consider:

- **Payload restrictions:** What is the payload mass that can be launched, determined by launch capacity.
- **Extreme conditions:** If the robot is meant to operate in highly inhospitable environments involving as high pressure, radiation, corrosive substances or extreme temperatures (high, low or alternating between the two extremes).
- **Communication delay:** Even at the speed of light, a round-trip to Mars takes over 30 min, and the delay only becomes longer for more distant missions. This translates into a requirement for semi- or even fully autonomous robots.
- **Microgravity:** If the robot is expected to operate in space rather than on (or beneath) the surface of a celestial body, the non-trivial effect of microgravity must be taken into account for everything from propulsion to collision avoidance.

Microgravity is also one of the aspects of space in terms of *environmental conditions* that has no analogue on Earth. For instance, friction is effectively absent in space, and therefore one cannot assume that the robot would simply stop moving when the propulsion is cut off. Microgravity can serve either as an obstacle or as an opportunity depending on the application. Thus, it provides a segue into the next section, which looks at the specific application of *formation flying*, where ER has been applied in a microgravity environment.

21.4.3 Formation Flying and In-Orbit Assembly

In-orbit assembly [6, 58] refers to the process of putting individual components together in-orbit to create a larger structure, and reconfiguration is the process of reshaping or otherwise altering an already assembled structure. Arguably the most successful case of in-orbit assembly is the International Space Station (ISS) [6], which is composed of several modules designed, manufactured and launched by different international actors and space agencies. A unique feature of in-orbit assembly is that it happens in a microgravity environment, which allows certain manoeuvres that are impossible or impractical on the ground.

A noteworthy type of in-orbit assembly is *self-assembly*, where the components self-organise into a larger target structure. An interesting example is given by Shen et al. [49], who study the feasibility of achieving a stable predefined structure from a random starting configuration of *tethered* units (individual robots in a swarm). The tethers feature universal connectors that allow any two connectors to dock together, enabling the robots to form any possible configuration. The study models a system of Intelligent Reconfigurable Components (IRCs) composed of FIMER and CONRO robots that provide tethering connections between pairs of IRCs, systems for position, orientation and wireless communication, and an onboard controller for

topology discovery, planning and communication. As the tethers are flexible, when a connection is established between two IRCs, the tether is reeled in to eliminate slack. The taut link, that is established, then behaves as a rigid beam due to the synchronised coupled orbital motion of the robots at either end. System optimisation relies on a hormone-inspired distributed control algorithm developed specifically for the CONRO robots [48]. Notably, the mechanics of the docking procedure produces non-trivial effects, such as damped oscillatory spinning that results in alternating clockwise/counterclockwise twisting of the tether until an equilibrium point is reached. Nevertheless, Shen et al. [49] show that the method can grow stable tethered structures by allowing the robots to dynamically reconfigure their communication and control strategy by following the simple preferential attachment instructions encoded in the form of artificial hormones.

There are several key motivations for pursuing in-orbit (self-)assembly:

- **Payload management:** Smaller components are easier and cheaper to launch than a single large structure.
- **Maintainability:** Modular structures are easier to repair and upgrade *in situ* (i.e. in orbit) without decommissioning the entire structure.
- **Repurposing:** Reconfigurable structures can be repurposed into different configurations.

In most cases, in-space assembly relies on the fundamental concept of *formation flying* [37], where the objective is to control a fleet of independent satellites in such a way that the respective *relative* distance between each pair of satellites remains fixed. This enables servicing procedures such as rendezvous and docking to take place, while satellites flying in formation can effectively emulate a rigid structure.

From the point of view of ER, formation flying poses an interesting challenge. Orbital dynamics is well-studied and tractable with various optimisation algorithms, while the environment poses some unique constraints (such as working in microgravity) and opportunities (allowing for the creativity of evolution to shine in design and optimisation settings). In the following, we briefly expand on an interesting application of evolutionary algorithms in the context of an inverse-dynamic approach to formation flying named *equilibrium shaping* [26].

21.4.4 Case Study: Equilibrium Shaping of Spacecraft Swarm

Equilibrium shaping (ES) has been developed for the purpose of achieving a stable spacecraft formation from arbitrary initial conditions (i.e. with satellites distributed randomly within a certain volume). It is a behaviour-based path planning algorithm based on a swarm control technique that introduces an artificial potential field [17], where the agents follow the negative gradient of the potential towards unique attractors identified as minima in the global potential landscape. In ES, each agent in the spacecraft swarm follows distinct behaviours that define the overall velocity field

for each agent. The net effect of all behaviours ensures that the agent ends up in one of the equilibrium points designed to coincide with the desired formation. Key properties of ES are the minimisation of inter-agent communication and its limited sensory information requirements.

Formally, the total velocity v_i of each satellite i is distributed as the sum of the velocities defined by three distinct behavioural patterns: *gather*, *dock* and *avoid*:

$$v_i = v_i^{\text{gather}} + v_i^{\text{dock}} + v_i^{\text{avoid}} \quad (21.2)$$

where each of the individual behavioural velocity components can be decomposed as a discrete sum of non-linear functions over state decision vectors, leading to a straightforward *nonlinear programming* formulation. This makes ES quite amenable to evolutionary optimisation techniques. To that end, Izzo et al. [28] apply ES to the problem of formation flying as posed in terms of swarm control [18], where neural controllers are tasked with maintaining the formation in a decentralised manner, without a dedicated control unit. Specifically, the set-up involves the SPHERES robotic platform, consisting of three spherical devices which can manoeuvre independently in six degrees of freedom. The authors design two distinct multi-layer perceptrons (MLPs) networks, one of which translates the relative positions of the satellites into velocities and another one for translating the target formation into angular velocities. They find that the controllers encounter difficulties achieving rotational invariance, i.e. having the controller perform the same action for two states that only differ by a rotated reference frame. For this reason, they use an additional sensor input representing an absolute reference frame. Note that the controllers are identical for each satellite, so every agent is controlled in the same way by the controller and can be swapped freely. Using particle swarm optimisation, the controllers are evolved to achieve sub-micrometre positional accuracy in 99.93% of 25000 simulation runs with the help of the SPHERES simulator [39].

21.4.5 Future Challenge: Fault Recovery

Robots deployed in space would be operating in unknown environments that are often harsh and can cause damage to the structural elements of the robot, its hardware controller (including processor, memory and other components), or both. However, unlike a terrestrial robot, which would be repaired or replaced in the event of damage, a robot in space would need to recover from the damage as fully as possible in order to complete the mission. Such failure recovery should take place at the controller level while still maintaining the strong guarantees for robustness and interpretability normally required for a space-grade controller.

In this regard, the translation of a neural network behaviour into a series of logical steps in de Croon et al. [13] highlights a potential pathway for neuroevolution for robot control in space. Specifically, as evolution is well suited for dealing with

changes in the topology of the search space and the solution itself, it can serve as a valuable tool for controller recovery in the case of irreparable hardware damage. For instance, reconfigurable hardware platforms, such as FPGAs, are becoming increasingly popular and energy efficient, and could ultimately find their way onboard space probes and robots. A controller implemented in FPGA can be reconfigured dynamically in case of a failure; however, since there is no way to estimate with certainty the type of failure, the reconfiguration procedure should be as generic as possible. The recovered controller might need to work with altered inputs (for instance, a blocked wheel or undeployed solar panel) or more subtle changes such as permanently corrupted memory or processor registers. Currently, the place-and-route step is a very computationally intensive part of the FPGA update cycle, which needs to be done with ground support. Further algorithmic improvement would be necessary to limit the recomputations to a small environment around a fault. While to our knowledge there are no dedicated studies on neuroevolution for controller recovery for robots deployed in space, a system such as that in [5] could in principle be used as a fallback system for controller recovery, even if it is initially designed to be used *only* in the case of such irreparable damage. An additional advantage is that robot systems and failures can be simulated with high accuracy and reproducibility, paving the way to fast prototyping cycle. We anticipate that advancing the state of the art in that direction would greatly expand the potential for the application of evolution to robot control for space applications.

21.5 Conclusions

Evolutionary computation and machine learning have been conjointly used for a wide number of space applications, including trajectory and path optimisation, robotics guidance and control and morphology evolution.

In some applications, combining evolutionary and machine learning methods helps in further reducing the need for human design choices. For instance, the design of neural networks, including the number and width of layers or the choice of activation functions, can be partially automated using evolutionary neural architecture search (ENAS). Evolutionary algorithms are used extensively in the prospect of automating offline trajectory design, and machine learning methods is used in shaping the search space, reducing expensive function evaluations and complex system modeling.

So far, the combined use of machine learning and evolutionary optimisation remains exploratory in nature. No evolutionary machine learning technique has yet become the standard used to solve a particular challenge in space engineering, despite the promise shown by the prospective studies showcased in this review. A dynamic subfield within EML for space applications is optimal control problems solved with artificial neural networks, which are in turn optimised with evolutionary methods. This, together with learning the search space in evolutionary trajectory design seems to be the key promising areas for future developments.

References

1. Ampatzis, C., Izzo, D.: Machine learning techniques for approximation of objective functions in trajectory optimisation. In: Proceedings of the IJCAI-09 Workshop on Artificial Intelligence in Space, pp. 1–6 (2009)
2. Ampatzis, C., Izzo, D., Ruciński, M., Biscani, F.: Alife in the galapagos: migration effects on neuro-controller design. In: Advances in Artificial Life. Darwin Meets von Neumann: 10th European Conference, ECAL 2009, Budapest, Hungary, 13–16 Sept. 2009, Revised Selected Papers, Part I 10, pp. 197–204. Springer (2011)
3. Basak, A., Lohn, J.D.: A comparison of evolutionary algorithms on a set of antenna design benchmarks. In: 2013 IEEE Congress on Evolutionary Computation, pp. 598–604. IEEE (2013)
4. Biscani, F., Izzo, D.: A parallel global multiobjective framework for optimization: pagmo. *J. Open Sour. Softw.* **5**(53), 2338 (2020)
5. Borrett, F., Beckerleg, M.: A comparison of an evolvable hardware controller with an artificial neural network used for evolving the gait of a hexapod robot. *Gen. Programm. Evol. Mach.* **24**(1), 5 (2023)
6. Boyd, I.D., Buenconsejo, R.S., Piskorz, D., Lal, B., Crane, K.W., De La Rosa, Elena, B.: On-Orbit Manufacturing and Assembly of Spacecraft. Technical report, Institute for Defense Analyses (2017)
7. Cangelosi, A., Marocco, D., Peniak, M., Bentley, B., Ampatzis, C., Izzo, D.: Evolution in Robotic Islands. Technical Report Ariadna ID: 09-8301, ESA (2010)
8. Cassioli, A., Di Lorenzo, D., Locatelli, M., Schoen, F., Sciandrone, M.: Machine learning for global optimization. *Comput. Optim. Appl.* **51**, 279–303 (2012)
9. Choi, J.H., Lee, J., Park, C.: Deep-space trajectory optimizations using differential evolution with self-learning. *Acta Astronautica* **191**, 258–269 (2022)
10. Choi, T.J., Togelius, J., Cheong, Y.-G.: Advanced cauchy mutation for differential evolution in numerical optimization. *IEEE Access* **8**, 8720–8734 (2020)
11. Dachwald, B.: Optimal solar sail trajectories for missions to the outer solar system. *J. Guid. Control Dyn.* **28**(6), 1187–1193 (2005)
12. Dachwald, B.: Optimization of very-low-thrust trajectories using evolutionary neurocontrol. *Acta Astronautica* **57**(2–8), 175–185 (2005)
13. de Croon, G., O'connor, L.M., Nicol, C., Izzo, D.: Evolutionary robotics approach to odor source localization. *Neurocomputing* **121**, 481–497 (2013)
14. Ellery, A., Scott, G.P., Gao, Y., Husbands, P., Vaughan, E., Eckersley, S.: Mars Walker. Technical Report AO/1-4469/03/NL/SFe, ESA (2005)
15. Espenschied, K.S., Quinn, R.D., Beer, R.D., Chiel, H.J.: Biologically based distributed control and local reflexes improve rough terrain locomotion in a hexapod robot. *Robot. Auton. Syst.* **18**(1–2), 59–64 (1996)
16. Fluke, C.J., Jacobs, C.: Surveying the reach and maturity of machine learning and artificial intelligence in astronomy. *Wiley Interdiscip. Rev.: Data Mining Knowl. Disc.* **10**(2), e1349 (2020)
17. Gazi, V.: Swarm aggregations using artificial potentials and sliding-mode control. *IEEE Trans. Robot.* **21**(6), 1208–1214 (2005)
18. Gazi, V., Fidan, B., Marques, L., Ordonez, R.: Robot swarms: dynamics and control. In: Kececi, E.F., Ceccarelli, M. (eds.), *Mobile Robots for Dynamic Environments*, pp. 79–126. ASME Press (2015)
19. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. *Evol. Comput.* **9**(2), 159–195 (2001)
20. Hennes, D., Izzo, D., Landau, D.: Fast approximators for optimal low-thrust hops between main belt asteroids. In: 2016 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 1–7. IEEE (2016)
21. Howard, D., Eiben, A.E., Kennedy, D.F., Mouret, J.-B., Valencia, P., Winkler, D.: Evolving embodied intelligence from materials to machines. *Nat. Mach. Intell.* **1**(1), 12–19 (2019)

22. Howard, D., Glette, K., Cheney, N.: Editorial: evolving robotic morphologies. *Front. Robot. AI* **9**, 874853 (2022)
23. Husbands, P.: Distributed coevolutionary genetic algorithms for multi-criteria and multi-constraint optimisation. In: Fogarty, T.C. (ed.) *Evolutionary Computing. Lecture Notes in Computer Science*, vol. 865, pp. 150–165. Springer, Berlin, Heidelberg (1994)
24. Ivezic, Z., Connolly, A.J., VanderPlas, J.T., Gray, A.: *Statistics, Data Mining, and Machine Learning in astronomy*. In: *Statistics, Data Mining, and Machine Learning in Astronomy*. Princeton University Press (2014)
25. Izzo, D.: Problem description for the 9th global trajectory optimisation competition. *Acta Futura* **11**, 49–55 (2017)
26. Izzo, D., Pettazzi, L.: Autonomous and distributed motion planning for satellite swarm. *J. Guid. Control Dyn.* **30**(2), 449–459 (2007)
27. Izzo, D., Ruciński, M., Biscani, F.: The generalized Island model. *Parallel Arch. Bioinspired Algorim.* 151–169 (2012)
28. Izzo, D., Simões, L.F., Croon, G.C.H.E.: An evolutionary robotics approach for the distributed control of satellite formations. *Evol. Intell.* **7**(2), 107–118 (2014)
29. Izzo, D., Sprague, C.I., Tailor, D.V.: Machine learning and evolutionary techniques in inter-planetary trajectory design. In: *Modeling and Optimization in Space Engineering: State of the Art and New Challenges*, pp. 191–210 (2019)
30. Jones, D., Schroeder, A., Nitschke, G.: Evolutionary deep learning to identify galaxies in the zone of avoidance (2019). [arXiv:1903.07461](https://arxiv.org/abs/1903.07461)
31. Kohlhase, C.E., Penzo, P.A.: Voyager mission description. *Space Sci. Rev.* **21**(2), 77–101 (1977)
32. Leitner, J., Ampatzis, C., Izzo, D.: Evolving ams for spacecraft rendezvous and docking. In: *Proceedings of the 10th International Symposium on Artificial Intelligence, Robotics and Automation in Space, i-SAIRAS 2010*, pp. 386–393. European Space Agency (ESA) (2010)
33. Yang, D.L., Xu, B., Zhang, L.: Optimal low-thrust spiral trajectories using lyapunov-based guidance. *Acta Astronautica* **126**, 275–285 (2016)
34. Lohn, J.D., Hornby, G.S., Linden, D.S.: An evolved antenna for deployment on Nasa’s space technology 5 mission. In: O’Reilly, U.-M., Yu, T., Riolo, R., Worzel, B. (eds.), *Genetic Programming Theory and Practice II*, volume 8 of *Genetic Programming*, pp. 301–315. Springer (2005)
35. Lozano, J.A., Larrañaga, P., Inza, I., Bengoetxea, E. (eds.): *Towards a New Evolutionary Computation*. Springer, Berlin, Heidelberg (2006)
36. Marchetti, F., Minisci, E.: A hybrid neural network-genetic programming intelligent control approach. In: *Bioinspired Optimization Methods and Their Applications: 9th International Conference, BIOMA 2020, Brussels, Belgium, 19–20 Nov. 2020, Proceedings*, vol 9, pp. 240–254. Springer (2020)
37. Mathavaraj, S., Padhi, R.: *Satellite Formation Flying: High Precision Guidance Using Optimal and Adaptive Control Techniques*. Springer Singapore (2021)
38. Mereta, A., Izzo, D., Wittig, A.: Machine learning of optimal low-thrust transfers between near-earth objects. In: *Hybrid Artificial Intelligent Systems: 12th International Conference, HAIS 2017, La Rioja, Spain, June 21–23, 2017, Proceedings*, pp. 543–553. Springer (2017)
39. Miller, D., Saenz-Otero, A., Wertz, J., Chen, A., Berkowski, G., Brodel, C., Carlson, S., Carpenter, D., Chen, S., Cheng, S., Feller, D., Jackson, S., Pitts, B., Perez, F., Szuminski, J., Sell, S.: SPHERES: a testbed for long duration satellite formation flying in micro-gravity conditions. *Adv. Astronautical Sci.* **105** (2000)
40. Omran, MG.H., Salman, A., Engelbrecht, A.P.: Self-adaptive differential evolution. In: Hao, Y., Liu, J., Wang, Y., Cheung, Y.-M., Yin, H., Jiao, L., Ma, J., Jiao, Y.-C. (eds.), *Computational Intelligence and Security*, Berlin, Heidelberg, pp. 192–199. Springer, Berlin, Heidelberg (2005)
41. Peniak, M., Bentley, B., Marocco, D., Cangelosi, A., Ampatzis, C., Izzo, D., Biscani, F.: An evolutionary approach to designing autonomous planetary rovers. *TAROS 2010*, pp. 198 (2010)
42. Peniak, M., Bentley, B., Marocco, D., Cangelosi, A., Ampatzis, C., Izzo, D., Biscani, F.: An Island-model framework for evolving neuro-controllers for planetary rover control. In: *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. IEEE (2010)

43. Peniak, M., Marocco, D., Ramirez-Contla, S., Cangelosi, A.: Active vision for navigating unknown environments: an evolutionary robotics approach for space research. In: Lacoste, H. (ed.), *ESA Special Publication*, volume 673 of *ESA Special Publication*, p. 7 (2009)
44. Peralta, F., Flanagan, S.: Cassini interplanetary trajectory design. *Control Eng. Pract.* **3**(11), 1603–1610 (1995)
45. Price, K.V.: Differential evolution. *Handbook of Optimization: From Classical to Modern Approach*, pp. 187–214 (2013)
46. Rommerman, M., Kuhn, D., Kirchner, F.: Robot design for space missions using evolutionary computation. In: 2009 IEEE Congress on Evolutionary Computation, pp. 2098–2105. IEEE (2009)
47. Cuartielles, J.P., Gibbings, A., Snodgrass, C., Green, S., Bowles, N.: Asteroid belt multiple flyby options for m-class missions. In: 67th International Astronautical Congress, p. IAC-16.C1.5.7x33119. International Astronautical Federation (2016)
48. Shen, W.-M., Lu, Y., Will, P.: Hormone-based control for self-reconfigurable robots. In: Proceedings of the Fourth International Conference on Autonomous Agents, AGENTS '00, pp. 1–8. Association for Computing Machinery (2000)
49. Shen, W.-M., Will, P.M., Khoshnevis, B.: Self-assembly in space via self-reconfigurable robots. In: 2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422), **2**, 2516–2521 (2003)
50. Shirazi, A., Holt, H., Armellin, R., Baresi, N.: Time-varying lyapunov control laws with enhanced estimation of distribution algorithm for low-thrust trajectory design. In: Modeling and Optimization in Space Engineering: New Concepts and Approaches, pp. 377–399. Springer (2023)
51. Simões, L.F., Cruz, C., Ribeiro, R.A., Correia, L., Seidl, T., Ampatzis, C., Izzo, D.: Path Planning Strategies Inspired By Swarm Behaviour of Plant Root Apexes. Technical Report Ariadna ID: 09/6401, ESA (2011)
52. Smith, R.: Open Dynamics Engine (2008)
53. Spenneberg, D., Albrecht, M., Backhaus, T., Hilljegerdes, J., Kirchner, F., Zschenker, H.: ARAMIES: A four-legged climbing and walking robot. In: Proceedings of the 8th International Symposium on Artificial Intelligence, Robotics and Automation in Space, vol. 603 (2005)
54. Stubbig, L.J., Cowan, K.J.: Improving the evolutionary optimization of interplanetary low-thrust trajectories using a neural network surrogate model. *Adv. Astronaut. Sci.* **175** (2021)
55. Telikani, A., Tahmassebi, A., Banzhaf, W., Gandomi, A.H.: Evolutionary machine learning: a survey. *ACM Comput. Surv. (CSUR)* **54**(8), 1–35 (2021)
56. Vinkó, T., Izzo, D.: Global optimisation heuristics and test problems for preliminary spacecraft trajectory design. Advanced Concepts Team, ESATR ACT-TNT-MAD-GOHTPPSTD (2008)
57. Willis, S., Izzo, D., Hennes, D.: Reinforcement learning for spacecraft maneuvering near small bodies. *AAS/AIAA Space Flight Mech. Meet.* **158**, 1351–1368 (2016)
58. Xue, Z., Liu, J., Chenchen, W., Tong, Y.: Review of in-space assembly technologies. *Chinese J. Aeronaut.* **34**(11), 21–47 (2021)
59. Yam, C.H., Lorenzo, D.D., Izzo, D.: Low-thrust trajectory design as a constrained global optimization problem. *Proc. Inst. Mech. Eng. Part G: J. Aerosp. Eng.* **225**(11), 1243–1251 (2011)
60. Zhang, F., Mei, Y., Nguyen, S., Zhang, M.: Survey on genetic programming and machine learning techniques for heuristic design in job shop scheduling. *IEEE Trans. Evol. Comput.* (2023)
61. Zuo, M., Dai, G., Peng, L., Wang, M., Liu, Z., Chen, C.: A case learning-based differential evolution algorithm for global optimization of interplanetary trajectory design. *Appl. Soft Comput.* **94**, 106451 (2020)

Chapter 22

Evolutionary Machine Learning in Control



Guy Y. Cornejo Maceda and Bernd R. Noack

Abstract This chapter aims to give an overview of recent applications of Evolutionary Machine Learning (EML) to control including opportunities and challenges. Control is at the heart of engineering applications. Examples include regulation, stabilization, reference tracking, synchronization, and coordination. Yet, control design of complex systems may be challenged by high dimensionality, nonlinearities, and delayed responses. A new path for control design is to reformulate the control problem as a regression problem to leverage powerful Machine Learning (ML) methods. In particular, bio-inspired ML methods are well adapted for solving control tasks thanks to easy deployment, interpretability, and little/no prior knowledge of the system to control needed. Hence, since the 50s, EML methods have been successful in optimizing intelligent controllers to solve many control tasks, including adaptive, multi-objective, robust control for robotics, electric engineering, and fluid mechanics, to cite a few examples.

22.1 Introduction

22.1.1 Control Theory

Control is the engineering field that aims to bring a dynamic system to a desired state or to behave in a desired way. We restrain this study to autonomous systems, i.e., systems that are invariant in time. In this framework, the dynamical system, also called plant (P), includes one or several actuators and sensors. The sensors provide at least partial information about the system state and characterize the control's performance. The sensor signals are collected in a vectorial quantity noted by s ; they are also referred to as the plant's outputs. The actuators manipulate the plant to achieve a control objective, and are driven by the actuation command, noted by

G. Y. Cornejo Maceda · B. R. Noack (✉)

School of Mechanical Engineering and Automation, Harbin Institute of Technology (Shenzhen), University Town, Xili, Shenzhen 518055, PR China

e-mail: bernd.noack@hit.edu.cn

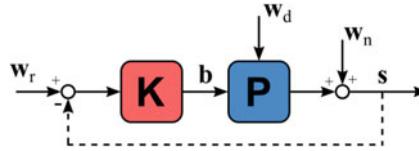


Fig. 22.1 Control loop diagram. The controller (K) sends an actuation command b to the plant (P). The plant gives back a sensor signal s that is information on the plant state. The plant is subject to external disturbances w_d and measurement noise w_n . For open-loop control, the actuation command is typically a function of a reference signal w_r . For closed loop, the actuation command is also a function of the sensor signal s . The dashed line indicates state feedback for closed-loop control only

b ; the latter is also referred to as the plant's input. A control law determines the actuation command. The controller is the control unit, while the control law is a function from the space of sensor signals to the space of all possible actions, i.e., the actuation space; In sensor-based feedback, the control law is a mapping from the sensor signals to the actuation command. In the following, the controller is denoted K , and the control law \mathbf{K} . The control objective is to derive the optimal control law that best fulfills the control objective. Following the control problem, the objective may be to control the system such that its outputs match a reference signal, noted w_r . There are two types of controls.

- **Open-loop control.** The actuation command b is a signal independent of the state of the system. It can be a function of a reference signal w_r or other signals.
- **Closed-loop control.** The actuation command b is a function of the system state.

The design of a controller K needs to reduce the impact of external disturbances w_d and sensor noise w_n .

Figure 22.1 summarizes the elements of the control loop.

22.1.2 Challenges and Current Methods for Control Design

Control design for real systems is challenged by several factors:

- High-dimensionality of the state space and the actuation space;
- Nonlinearities of the plant;
- Time-delays between the action and the response;
- Uncertainties including model uncertainties, parameter uncertainties, disturbances, and noise;
- Multiple, sometimes conflicting control objectives;
- Partial observation of the system state;
- Distributed actuators and sensors;

Control theory based essentially on linear control has been successful for a large range of systems; methods such as Model Predictive Control (MPC) and

Linear-Quadratic-Gaussian control (LQG) have been largely employed in many fields such as automotive, chemical processing, and power generation, to name a few examples. The stability of closed-loop behavior has been proven for optimal control. On the other hand, robustness against changes in operating conditions is not always guaranteed. We refer in particular to the famous paper by J. C. Doyle on the absence of guaranteed margins for LQG regulators [43]. For complex systems, reduced-order models can be built and linearization around a state of interest allow to employ linear control methods. Yet, those models may lack robustness as they have a limited domain of validity and generally don't take into account the effect of control. Robust controllers can be built with adaptive control methods, i.e., with online parameter estimation and control update. Nevertheless, the robustness of such methods is far from guaranteed.

22.1.3 Why Evolutionary Machine Learning (EML)?

Machine learning opens a new avenue for the challenges of nonlinear dynamics. Machine learning may build “intelligent” controllers that are able to leverage the nonlinearities of the system for improved performances and to perform well in a larger range of uncertainties than adaptive control [44]. In this framework, the control problem is reformulated as a regression problem for the control law. This optimization problem is in general non-convex and may feature many minima and plateaus. Machine-learned controllers can be optimized directly in the plant in a model-free manner following an iterative and stochastic process. Figure 22.2 summarizes the two main control design frameworks.

EML methods are particularly fitted to solve regression problems for control. The general idea of EML is the gradual optimization of candidate solutions to the

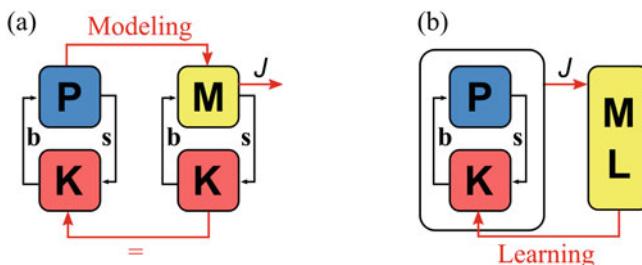


Fig. 22.2 The two main control design frameworks. For simplicity, the reference signal and disturbances are not displayed. **a** Model-based control. A reduced-order model (M) of the plant (P) is built and used to optimize a controller (K) minimizing a cost function J . The controller is then used as such to control the plant. **b** Machine learning control. The controller is updated thanks to a machine learning algorithm, depicted by the yellow block (ML). The optimized controller is obtained by solving a regression problem

regression problem (controllers in this context) following the Darwinian evolution. First of all, one or several candidate solutions also referred to as a *population of individuals*, are randomly generated. All individuals are evaluated, or tested, to assess their performance or *fitness*. Based on their performance, the individuals are selected to generate the new *generation* of individuals. New individuals are generated by the stochastic modification of one individual (*mutation*) and/or recombination (crossover) of two individuals. Mutation and crossover are referred to as *genetic operators*. The choice of the genetic operator to generate a new individual is parameterized by the mutation and crossover probabilities. A new generation of individuals is produced until a stopping criterion is reached such as a performance threshold or total evaluation budget. Compared to other optimization solvers, evolutionary algorithms benefit from:

- Noise insensitivity. These algorithms are gradient-free, i.e., they do not require the optimization problem to be differentiable;
- A balance between exploration and exploitation enabling optimization in non-convex spaces including many minima, plateaus, and valleys;
- Model-free optimization, the plant is considered as a black box and mappings between the outputs and the inputs of the system are directly learned. No information on the dynamics of the plant is needed but can be included.
- Interpretability. Symbolic regression solvers like genetic programming optimize function expressions that help human understanding of control mechanisms.
- Easy to incorporate domain-specific knowledge. Constraints can be enforced in the structure of the solutions for example, and known solutions can be seeded in the population.

However, evolutionary methods may suffer from premature convergence, from slow learning, and come with no guarantee of the solution's stability. EML parameters (population size, generations, selection method, mutation, and crossover probabilities) often require fine-tuning to achieve the best performances.

22.1.4 Taxonomy of EML Methods for Control

Following [18], Machine learning (ML) is a sub-field of Artificial Intelligence (AI) regrouping that process and extract information from data. ML is subdivided into three categories:

- Supervised: learning is achieved from labeled data with expert knowledge. The goal is to build a mapping between the data and the labels.
- Semisupervised: learning is achieved from partially labeled data or by interaction with the environment. Semisupervised ML includes Reinforcement Learning (RL) and generative models.
- Unsupervised: learning is achieved from unlabeled data. The goal is to derive the underlying structure of the data.

We locate evolutionary computation between supervised and unsupervised learning as the methods relies on the interaction with an environment/plant to solve a

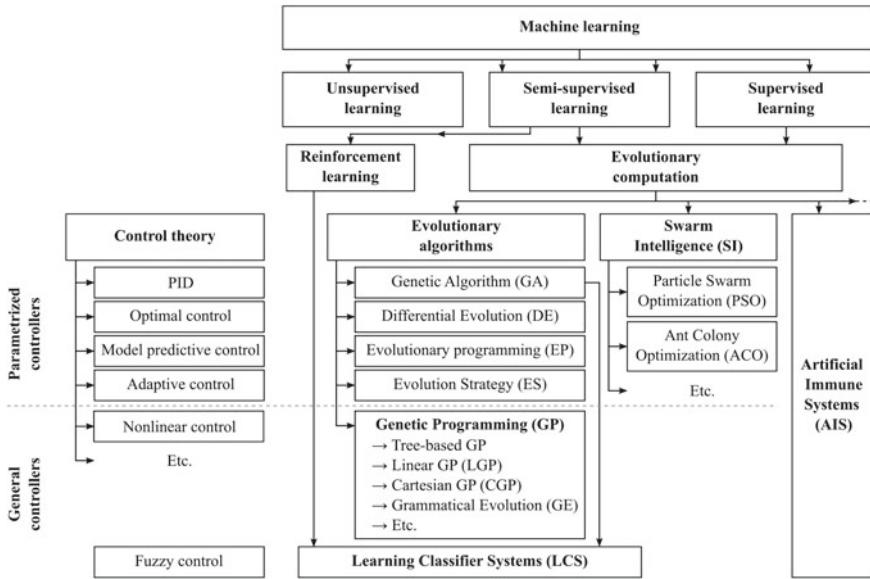


Fig. 22.3 Taxonomy of the main Evolutionary Machine Learning (EML) methods employed for control (right). The methods are divided into two groups: methods for optimization of parametrized controllers (top) and methods for optimization of general controllers (bottom). Classical methods of control theory and fuzzy control (left). Studies combining methods of evolutionary computation and control theory are reported in this review

regression problem. This chapter focuses on the main methods of evolutionary computation employed for control, i.e., evolutionary algorithms, Learning Classifier Systems (LCS), Swarm Intelligence (SI), and Artificial Immune Systems (AIS). Figure 22.3 gives the taxonomy of these EML methods in parallel with control theory methods that solve similar control problems.

Review papers such as [44, 48, 80, 131] detail many success stories of EML to solve control tasks. Thus, the automation control systems category is the 5th most popular area of applications of EML with 10977 papers registered in the Web of Science database in 2020 [131]. Note that this number only includes studies using Genetic Algorithm (GA), Genetic Programming (GP), Differential Evolution (DE), Evolution Strategy (ES), evolutionary programming, and their variants. Interestingly, among the 10977 papers the vast majority of 10540, i.e., 96%, employ algorithms to solve parametric optimization tasks.

In their review, [44] and [48] propose different classifications of EML methods based on the type of controller or on the type of optimization. [44] makes the distinction between “pure” and “hybrid” controllers. Pure controllers are controllers learned from scratch with an evolutionary method, while hybrid controllers are known controllers (such as neural networks and fuzzy controllers) optimized or augmented with evolutionary methods. See Fig. 22.3 for an overview of the main methods of control theory. On the other hand, [48] distinguish offline design of controllers and online

optimization, i.e., the controller is updated in real-time. Eventually, both reviews classify the methods into two groups:

- Methods for parameterized control law optimization. A fixed structure of the control law is given and its parameters are tuned with EML.
- Methods for general control law optimization. The control structure and parameters are learned simultaneously.

This study follows the latter classification.

22.1.5 Content of the Chapter

This book chapter is organized into six parts. First, the control problem is reformulated as a regression problem to be solved with EML (Sect. 22.2). Then, we review pioneering and among the most significant and recent applications of EML to parametric optimization (Sect. 22.3) and control law optimization (Sect. 22.4) for control. In Sect. 22.5.1, EML for learning control-oriented reduced-order models is reviewed. Control stability and robustness of EML solutions are discussed in Sect. 22.5.2. Finally, Sect. 22.6 concludes this chapter and envisions the future of EML for control.

22.2 Problem Formulation

In this section, we describe how the control problem can be reformulated as a regression problem to be solved with EML.

22.2.1 Control Objective and Cost Function

To employ evolutionary methods, the performance of a given controller needs to be translated into a fitness function to be maximized. In the following, we chose to employ the term *cost J* from the control community. Contrary to fitness, the cost function is a quantity to minimize. In general, the control goal includes several non-commensurable and conflicting objectives, such as distance to the optimal solution, stability, convergence rate, noise sensitivity, robustness, actuation power, etc. A common way to design the cost function is then to take a weighted sum of several cost functions J_i .

$$J = \sum_i \gamma_i J_i \quad (22.1)$$

The choice of the weights (γ_i) is strongly dependent on the applications.

There are also Pareto-based techniques that consider a cost vector and avoid choosing subjective weights. The goal is then to look for Pareto optimal solutions also referred to as non-dominated solutions, i.e., solutions in which one performance criterion cannot be improved except by degrading another one. Multi-objective genetic algorithms and multi-objective genetic programming methods have been developed to learn those Pareto fronts. However, the comprehensive exploration of the Pareto front for control problems with several objectives may require a large number of plant evaluations.

22.2.2 *Control Optimization as Regression Problem*

Once the cost function is defined, the control problem can be reformulated as a regression problem where the goal is to derive the best controller that minimizes the cost function J . In sensor feedback, the sensor signals s are the control law input. Often, the sensor signals s are lifted to a causal feature a as control law input. The feature a is causal, which means it is a function of the sensor signals and, potentially their history. The feature may be an estimated state from a dynamic observer or dynamical model [95], obtained by filtering into different components [88], or the time-delay coordinates [59].

We distinguish two categories of optimization problems.

- **Optimization of parameterized control laws.** Here, the control law has a given structure with free parameters p , e.g. the gain matrix of linear control. A general expression is:

$$\mathbf{b} = \mathbf{K}_p(\mathbf{a}) \quad (22.2)$$

where \mathbf{b} is the actuation command, \mathbf{K} is the control law, p are the free control parameters to be optimized, and \mathbf{a} is the sensor-based feature. The optimization problem is then to derive the optimal parameter vector p^* that minimizes the cost function. The regression problem to solve is

$$\mathbf{p}^* = \arg \min_{p \in E} J(p). \quad (22.3)$$

where E is the space of admissible parameters. In control, EML has been largely employed for tuning PID (Proportional-Integral-Derivative) controllers or optimizing the weighting matrices in LQG or H-infinity controllers. Typical EML algorithms to solve such problems are Genetic Algorithms (GA), Evolutionary Strategies (ES, CMA-ES, etc.), Differential Evolution (DE), and Swarm Intelligence (SI) methods (PSO, ACO, etc.). The solutions are often represented by binary or n-ary strings [62], but there are also other representations closer to the problem such as gray coding and float point representation [106].

- **Optimization of general control laws.** The structure of the control law is not imposed. In this case, a general expression is

$$\mathbf{b} = \mathbf{K}(\mathbf{a}) \quad (22.4)$$

where \mathbf{b} is the actuation command, \mathbf{K} is the control law, and \mathbf{a} are the plant features. EML optimizes both the structure and its parameters. Thus, control laws can directly be learned with function optimization solvers such as Genetic Programming (GP) [79]. Functions can be represented by trees (like in gene expression programming) or matrices like in Linear Genetic Programming [16, LGP] for example. Functions in the frequency domain, or s-domain can also be learned with an optimization of block diagrams directly on Simulink [51]. The regression problem to solve is then: Find the optimal controller \mathbf{K}^* that minimizes the cost function J .

$$\mathbf{K}^* = \arg \min_{\mathbf{K} \in \mathcal{K}} J(\mathbf{K}) \quad (22.5)$$

with \mathcal{K} being the space of all admissible control laws.

One of the benefits of EML in controller optimization is that the type of control is not imposed. Open-loop and closed-loop strategies can be learned indifferently as well as complex controllers leveraging noise to excite the plant. In practice, EML re-discovers open-loop and closed-loop control mechanisms and combines them to achieve better performances; This is especially true in fluid mechanics.

EML methods also benefit from a straightforward implementation of control constraints and domain-specific knowledge. Constraints can directly be enforced in the internal representation of the solutions to render some inaccessible. Pre-tests can also be done for each individual to avoid costly evaluations of unwanted solutions. The constraints can also be relaxed and included as a penalization term in the cost function.

Great care must be taken when designing the regression problem to ensure that it reflects the original control problem. The choice of the EML algorithm depends on the nature of the control laws, control inputs, control outputs, and genetic operators. Specific EML methods are often developed to take into account the specificities of the problems. In the following, we review pioneering recent applications of EML to control.

22.3 Optimization of Parameterized Control Laws

EML for parametric optimization has been pioneered by [62] with GA. Since then many variants have been developed as well as other parameter optimization techniques based on evolutionary principles such as evolution strategies, differential evolution, and evolutionary programming. In this section, we review some of the most significant and recent control tasks achieved with parametric optimization using EML.

22.3.1 Genetic Algorithm, Differential Evolution, and Evolutionary Strategies

Flow control problems are notoriously hard to solve due to the inherent nonlinearities of the Navier-Stokes equations, the multi-scale feature of turbulence, and significant time delays between actuation and sensing. Nevertheless, [8] show that GA is able to learn the optimal parameters (voltage amplitude, burst frequency, and duty cycle) of a plasma actuator to control a turbulent separated flow in experiments. Two control tasks are achieved, the minimization of the reattaching distance and the maximization of the wall pressure fluctuations. After only a few generations of 120 individuals, GA learns the optimal solutions, i.e., forcing at the shear layer frequency and half the shear layer frequency, respectively. Thereafter, more challenging flow control problems have been tackled with EML and especially GP, see the next section for more details.

Dracopoulos [44] reports various works combining GA and other artificial intelligence and control methods already since the '90s. In particular, the author gives examples of combinations with neural networks, reinforcement learning, fuzzy logic, PID controllers, pole placement adaptive controller, and traditional optimization theory. Since then, [90] propose an improved genetic algorithm optimization fuzzy controller for the control of throttle valve in managed pressure drilling. The controller is developed to tackle the strong nonlinearities and time variability of the throttle valve control. The authors propose an improved initial population generation, an adaptive genetic operator probability selection, and genetic operators. The learned controller presents great advantages in terms of speed, stability, and robustness. More synergies between evolutionary algorithms and fuzzy controllers are described in [27–29, 61, 68, 134], and between GA and neuro-fuzzy controllers in various fields such as electrochemical systems [104], underwater vehicles [69], manipulation robots [122], and Unmanned Aerial Vehicle (UAV) control [14].

Other EML methods such as Differential Evolution (DE) and Evolutionary Strategies (ES) have also been employed for control. We refer to the review paper by [10] for a comprehensive review of DE over the two last decades. The authors report, in particular, the optimization of fuzzy controllers, multivariable PI controllers, and optimal control design. ES, CMA-ES, and other variants have been developed, such as the General Learning Evolutionary Algorithm and Method (GLEAM) for planning and control [13], and employed for the optimization of trajectory planning and a sliding mode tracking controller [31], the optimization of large-scale UAV cluster confrontation game in response to the shortcomings of optimal control and reinforcement learning [92], weight optimization of Artificial Neural Networks (ANNs) for reinforcement learning with an ensemble of mutation strategies outperforming classical ES [2], automatic berthing with optimal control theory [100, 101], low and high-dimensional control optimization of reservoir management processes [4], and to optimize an adaptive predictive control of the hydrocracking process [89].

22.3.2 *Swarm Intelligence Methods*

Swarm Intelligence (SI) techniques are also well adapted for control design. PSO, in particular, has been able to solve large-scale nonlinear optimization problems when analytical methods failed to converge. Compared to GA, PSO benefits from an easier implementation with fewer parameters to tune, a more effective memory capability, and more efficient preservation of the population diversity [39]. In their review of PSO, [39] give examples of applications to power systems including PID controller tuning, weight optimization for neurocontrollers, and fuzzy logic-based controller optimization with hybrid GA-PSO methods. We refer to the review paper for more details on those applications. [132] reports that up to 38 SI methods have been developed between 1992 and 2017, and the two most popular SI methods being Particle Swarm Optimization (PSO) and Ant Colony Optimization (ACO). The authors also review SI applications to control design, such as PSO for control of renewable energy systems, or design of robust controllers, and ACO for optimal control of pumping water distribution networks.

Lately, PSO has been employed for efficient coverage control for wireless sensor networks [139], disturbance rejection for position control of the magnetic levitation system [64], PID controller optimization for a quadrotor with a virtual sensor [112] and for efficient control of a nonlinear double-pendulum overhead crane with sensorless payload motion [67]. ACO has also been employed in several domains such as traffic control thanks to the Internet of Vehicles (IoV) [83], controller optimization for humanoid guidance in cluttered environments [127], optimization of a PID controller for vibration control of a wind turbine tower under different types of loads [120], and stability control for a magnetic levitation system [110].

22.3.3 *Hybrid Variants*

Efficiency of EML methods has been improved by merging them with other methods. [96] combine GA and the downhill simplex method to accelerate the learning of a chiller configuration for a heat source plant. GA acts as an exploration step, its role is to find new basins of attractions in the search space and to avoid getting stuck in a local minimum. On the other hand, downhill simplex linearly combines the best individuals for a fast descent in the minimum; it acts as an exploitation step. The combination of the two methods is enabled by a straightforward conversion between the gene and vector representation. Such a method cures the poor exploitation of gradient information by GAs. The hybrid method requires 280 iterations to find the global optimum while classic GA needs 1320 iterations, i.e., a 79% evaluation cut.

In addition, PSO and AOC have been combined with several other methods in many applications, for example, [135] combine PSO and MPC for path tracking of four-wheel steering and four-wheel drive vehicles, [140] combine PSO and central pattern generator for control and optimization of a bionic robotic fish, [15]

combine PSO and Gravitational Search Algorithm (GSA) to optimize a fuzzy sliding mode controller for Doubly Fed Induction Generators (DFIG) wind turbine, [12] combine ACO and Nelder-Mead's downhill simplex to tune a PID controller for the automatic voltage regulator system, [84] combine ACO and sine-cosine algorithms for optimal path search and control of a mobile robot, and [26] combine ACO and GA to maximize the maximum power point of a photovoltaic module under varying temperatures and sunlight irradiance.

22.3.4 Multi-objective Optimization

As mentioned earlier, EML has also been employed to optimize parameters based on multiple and often conflicting objectives. Recently, multi-objective GA has been employed for hybrid electric vehicle parameter optimization [19] energy optimization in wireless sensor networks [70], response mitigation of a wind-excited tall building [77], PID controller design [53], optimal distributions of actuators and sensors in structures [24, 25], energy optimization and thermal comfort in building design [144], solving complex multi-UAV mission planning problems [121], greenhouse environment control [94], and optimization of HVAC system energy consumption [111].

Multiple-objective problems have been solved with PSO, such as optimization of power management in AC/DC micro-grid [66], congestion control in wireless sensor networks with a hybrid multi-objective PSO-GSA algorithm [130], optimization of a robust fuzzy controller for a four-degree-of-freedom quadrotor [97], global path planning and path control for unmanned surface vehicle [54], path planning in rough terrain for rotary unmanned aerial vehicles [145], and PI control of DFIG wind turbine under electrical fault conditions [7]. We refer to the detailed review by [126] for more information on multi-objective meta-heuristics for controller tuning. ES has also been employed for multi-objective control of the IFAC 1993 benchmark control problem [11].

22.4 Optimization of General Control Laws

In this section, we review recent applications of EML methods to derive general control laws, i.e., problems where the structure of the control law is not imposed.

22.4.1 Tree-based Genetic Programming and Linear Genetic Programming

The first applications of Genetic Programming (GP) to control are the examples provided by [79]. Control laws are represented by tree expressions. GP is employed to solve the broom balancing problem where the goal is to balance a broom on a translating cart in minimal time. To assure the robustness of the solution, 10 initial conditions have been tested for each individual. For the learning process, 500 individuals evolved through 50 generations. The learned solution performed slightly better than a “pseudo-optimal” strategy derived from a linearization of the problem. Another example is the truck backer upper problem where the goal is to steer a tractor-trailer truck so as to back it up to a loading dock. For robustness, 8 initial conditions have been tested and 1000 individuals evolved through 50 generations. After 25 generations, GP derives a solution for the eight cases. It is worth noting that for the examples, there are no known mathematically exact solutions.

In [114], the authors propose the first implementation of online GP for real robot control. A compiled GP system is employed, i.e., a binary code is generated without any interpreting steps. The genetic operators are then directly applied to the binary code. Such a method improved the performance by a factor of 2000 compared to an interpreted-language method. GP is able to learn an efficient control law with noise input and real-time constraints for obstacle avoidance. Interestingly, the performance of the control is retained for different environments.

In [80], the author presents 77 human-competitive results produced by GP. We note in particular the synthesis of a time-optimal robot controller and the design of an analog electrical circuit that implements a near-optimal strategy [81]. 72 test cases were evaluated for each individual. A population of 40000 individuals evolves through 70 generations for the robot controller and a population of 640000 individuals evolves through 31 generations for the electrical circuit. Among the human-competitive results related to control, we note the reproduction of patented results such as the PID controller (1939 U.S. patent 2,175,985 by Albert Callender and Allan Stevenson), the PID-D2 (second derivative) type of controller (1942 U.S. patent 2,282,726 by Harry Jones) and also the production of innovative results that have been patented. We refer to the review paper for more information on those successes.

Recently, [42] developed a new GP method (complete binary genetic programming, CBGP) to synthesize an optimal control for a group of robots. The advantage of CBGP is that all functions are represented by a complete binary tree of the same depth. Moreover, the arguments and functions with one and two arguments are at specific positions in the tree. The definition of genetic operators becomes then easier to define. We refer to [41] for a review of machine learning techniques, especially GP methods, employed for control synthesis, optimal control, and model identification.

Regarding multi-objective GP algorithms for control, they have been employed in autonomous controller design for UAVs [6, 115], crowd control [63], and performance optimization of home energy-management systems [147].

Fluid mechanics presents a particular challenge to control as unsteady flows include complex, nonlinear, and multi-scale dynamics that are not easy to address even for popular machine learning algorithms [18]. Moreover, contrary to other engineering fields, fluids often require expensive tests. Experiments are often difficult to repeat and automate and have generally a limited evaluation budget to reduce the impact of drifts. Numerical simulations need also supercomputers to do parallel computations. In addition, flow properties require long evaluations (often more than 10 times the natural period of the phenomenon to control) to have converged statistics. The definition of the flow control problem is not straightforward as decisions on the type and number of sensors and actuators, their position, the cost function definition, and flow features are required. There is no general method to define the flow control problem and they often rely on engineering wisdom. Hence, classical control methods have been replaced with model-free machine learning methods such as GP. Recently, Machine Learning Control (MLC) based on GP [45] and LGP [34] has been successful in dozens of flow control tasks in numerical simulations and experiments often exploiting nonlinear mechanisms [113, 123]. Due to the stochastic nature of EML, the learned control laws are rarely reproducible. However, in most of the flow control applications, the same control mechanisms are consistently rediscovered in different runs.

The straightforward implementation of GP in experiments enables an easy interface with complex measurement methods. Thus, in [49], GP is employed to mitigate the flow separation over a backward-facing step in a water experiment with real-time feedback of the velocity field for the control.

GP is also able to learn robust control laws. The vortex-induced vibration of a cylinder has been reduced by 94.2% with GP and the learned control law is robust and efficient for different conditions (Reynolds number ranging from $Re=100$ to 400) [124]. In order to enforce robustness, [5] and [129] learn control laws in varying conditions for stall suppression with increasing angle of attack and drag reduction of a truck model with varying yaw angle, respectively.

Recently, GP has been benchmarked against reinforcement learning for the drag reduction problem of cylinder flow. [23] show that for this particular problem, deep reinforcement learning shows higher robustness with respect to the initial conditions and noise, while GP identifies compact and interpretable control laws. Moreover, in this study, like in many others in fluid mechanics, GP acts as a sensor optimizer, i.e., only a few sensors are selected to achieve effective control. On the other hand, in [119], GP achieved the best performance but needed a larger number of evaluations and had the highest learning variance.

LGP and downhill simplex are also combined to exploit the local gradient information in a subspace spanned by the most performing individuals. Better performances than GP and with fewer evaluations (1/5) are achieved for the stabilization of the fluidic pinball in numerical simulations [33] and the open cavity experiment [36] with gradient-enriched Machine Learning Control (gMLC). The learning rate is estimated to be improved by a factor of 10. Contrary to [96] (see Sect. 22.3), an additional reconstruction step is required to build back a matrix representation for

the linear-combined control laws. The reconstruction is in essence a function-fitting problem that the authors solve with LGP.

22.4.2 Cartesian Genetic Programming and Grammatical Evolution

Cartesian GP (CGP) employs graph representation to encode the control laws. The fixed two-dimensional grid of nodes prevents bloating, i.e., the control law does not become larger with each new generation without any performance improvement; Moreover, CGP can represent solutions with an arbitrary number of outputs [58, 107] Applications of CGP to control include the optimization of a controller to guide a robot out of a maze [58] the optimization of ANN for nonlinear control exemplified on the simple and double pole stabilization problem [98, 99], and the design of continuous-time controllers for hydraulic turbine power control [71].

In Grammar-based GP or Grammatical Evolution (GE), the control laws are evolved according to a specified grammar. Thus, GE benefits from flexible means of search space restriction, and homologous operators [102] which are valuable to encode control constraints and reduce the complexity of the control laws. In [22], the authors highlight the flexibility of GE by being able to learn controllers in different paradigms (behavioral or connectionist), different structures (rule sets, machine code, or complex polynomials), and different coordination scales (within a single robot or multiple robots). We refer to their publication for references on each application. [136] employs a grammar to define a nonrandom mutation operator for the gait control of a snake-like robot. The authors achieved better results than classical GP for unconstrained, constrained environments and with partially damaged snakes. Interestingly, when trained in an environment with obstacles, the snake robot learns solutions adopted by real snakes such as body elevation or compact side winding. In [21, 22], the authors reformulate the control problem as a vector-valued function to be estimated with GE for autonomous robot control. [143] employ GE for designing a control program for a vehicle robot.

Table 22.1 summarizes the key successes of GP for flow control tasks. Note that in most of those applications, convergence has been reached in approximately half of the total number of evaluations.

22.4.3 Artificial Immune Systems and LCS

Following the words of [137]: “The immune system is highly distributed, highly adaptive, self-organizing in nature, maintains a memory of past encounters and has the ability to continually learn about new encounters”. Hence, immune systems have inspired methods based on Artificial Immune Systems (AIS) to solve different

Table 22.1 Summary of recent control problems in flow control solved with GP

Paper	Plant	Type	# Plant inputs	# Plant outputs	Main objective	Algorithm	Pop. Size × #Gen
Parezanovic et al. [116, 117]	Mixing layer	Exp.	96 (in unison)	7	Energetization	Tree-based GP	20 × 25
Duriez et al. [46], Devbien et al. [37]	TBL over a Ramp	Exp.	54 (in unison)	2	Separation control	Tree-based GP	100 × (5, 10, 5)
Kadlic et al. [71]	Hydraulic turbine	Num.	4	1	Reference tracking	CGP	30-50 × 4000
Gautier et al. [49]	Backward-facing step	Exp.	1	1	Reirc. zone red.	Tree-based GP	500 × 12
Chovet et al. [32]	Backward-facing step	Exp.	20 (in unison)	3	Mixing increase	Tree-based GP	100 × 12
Li et al. [87, 88]	Ahmed body	Exp.	4 (in unison)	12	Drag reduction	LGP	50 × 5
El Sayed et al. [47]	High-lift config.	Exp.	33	8	Lift increase	Tree-based GP	150 × 8
Asai et al. [5]	Airfoil	Exp.	1	1	Stall suppression	Tree-based GP	100 × 15
Cornejo Maceda et al. [35]	Fluidic pinball	Num.	3	9	Drag reduction	LGP	100 × 10
Ren et al. [124]	Cylinder	Num.	2 (in unison)	1	Stabilization	Tree-based GP	50 × 25
Li et al. [86]	2D Mixing layer	Num.	1	0	Stabilization	LGP	100 × 6
Li et al. [86]	2D Mixing layer	Num.	1	0	Energetization	LGP	100 × 6
Wu et al. [142]	Jet	Exp.	1	0	Mixing	LGP	100 × 6
Zhou et al. [146]	Jet	Exp.	6	0	Mixing	LGP	100 × 30
Kane [73]	Wind turbine	Num.	3	31	Energy production	Tree-based GP	100 × 100
Cornejo Maceda et al. [33]	Fluidic pinball	Num.	3	9	Stabilization	LGP+	1000 eval.
Passagia et al. [118]	Airfoil	Exp.	82 (in unison)	8	Lift-to-drag ratio max.	LGP	60 × 6
Castellanos et al. [23]	Cylinder	Num.	2 (in unison)	(5,11)	Drag reduction	LGP	100 × 15
Semanan et al. [129]	Yawed truck model	Exp.	5	18	Drag reduction	LGP+	1200 eval.
Cornejo Maceda et al. [36]	Cavity flow	Exp.	1	1	Stabilization	LGP+	1000 eval.
Pino et al. [119]	Cylinder	Num.	4	5	Drag reduction	Tree-based GP	3000 eval.

types of problems. In [137], the authors report applications of AIS to the field of robotics, e.g., the control of large populations of robots to generate self-organization [85, 108], and the design of an autonomous robot—the immunoid—that mimics the immune system for displacement and collecting tasks and where the antibodies act as potential behaviors [78, 141]. [9] proposes an adaptive control methodology based on the immune network, the resulting methodology is close to Q-learning. [76] tune a PID controller for nonlinear processes with an immune network algorithm based on fuzzy sets. [72] take inspiration from the immune system to build increasing levels of intelligent control (robust feedback control, adaptive control, optimal control, planning control) for autonomous aircraft control problems. More recently, AIS has been employed for the coordination and control of a wheeled mobile manipulator [38].

AIS is part of rule-based machine learning alongside Learning Classifier Systems (LCS). LCS are methods that combine discovery, usually based on genetic algorithm, and learning, based on supervised learning, or reinforcement learning [138]. The main idea is to break down the complex solution space into simpler smaller parts. Such an approach is well-suited to derive adaptive controllers. Thus, [65] introduces a variant of LCS, Temporal Classifier System (TCS), to learn a real robot controller. [20] develop a distributed adaptive control for road traffic junction signals with LCS. [125] employs LCS to improve the design of Organic Computing, a field aiming to design and control complex systems that are able to adapt. [74] employs a variant of LCS, eXtended Classifier System (XCS), to evolve a set of control rules to control networks that model real-world systems. [133] employ XCS for self-adaptive and self-organizing agents; compared to (deep) reinforcement learning the advantages of XCS are the interpretability and continual evolution of knowledge.

22.5 More Control Problems

22.5.1 Model Identification for Control

EML is also employed to build models of the plant and subsequently apply methods from classical control theory. One of the first studies on this approach is [82], where the authors use GA to identify both discrete and continuous-time systems. The poles and zeros are then estimated with GA and used to build an adaptive controller. Note that the control design method performs well even in the presence of unmodeled dynamics. Nonetheless, the authors report a huge computational load and an increase of the CPU time by a factor of 50 compared to a classical model identification technique. [51] employ GP for the modeling of a laboratory scale process involving a coupled water tank system and for the identification of a helicopter rotor speed controller and engine from flight test data.

Recently, [109] use a multi-objective GA to estimate the parameters of a nonlinear dynamic model for a hydraulic robot manipulator. The authors show in particular

that the multi-objective GA achieves better performance than single-objective GA. [1] and [52] employ a GA and a multi-objective GA, respectively, to tune the parameters of a fuzzy modeling method and thus identify a nonlinear system. In [40], multi-objective GA is employed to optimize the parameters of an automatic control system (a Simulink model). Multi-objective GP is employed in combination with a NARMAX method to build a model for a chaotic system [56].

The development of control-oriented digital twins that mirror the plant is key to accelerating the learning of controllers. Expensive evaluations can be replaced by fast and cheap simulations. Moreover, digital twins can be learned during the controller optimization process by leveraging the data generated by the evaluation of each individual. Lately, evolutionary methods have also been employed to build digital twins. In [91], the authors developed Evolutionary Digital Twin (EDT) to address the lack of flexibility and adaptability in traditional methods. [55] propose a GA approach to building digital twins for photovoltaic power simulation. [93] give an example of trajectory optimization with GA based on a digital twin.

22.5.2 *Control Stability and Robustness*

Following [44], EML controllers can be classified into two categories: pure controllers and hybrid controllers. Hybrid controllers can benefit from stability and robustness results from control theory. Thus, [30] propose a multi-objective evolutionary algorithm approach combined with H_∞ loop-shaping design procedure for robust design of control systems. On the other hand, the stability of pure controllers needs a case-by-case analysis usually from scratch. Moreover, as EML solutions are rarely identically reproducible, the stability and robustness analysis of the controllers need to be repeated for each new run.

For linear systems, the stability depends on the value of the transfer function's poles. The poles need to reside in the stability region of the complex plane, i.e., in the open left plane for continuous time and inside the unit circle for discrete times. Such analysis can be automated for most EML solutions. For control laws learned with GP, a simplification step is generally required. However, the simplification is not straightforward as some operators are protected (\div , log, etc.), i.e., their definition is extended to all real numbers. The protection allows to close the space of controllers but calls for special treatment for singularity points. For nonlinear systems, stability is studied through the Input-To-State Stability (ISS) paradigm, and the EML solutions need to be integrated into the control system.

The robustness of EML solutions to out-of-design conditions and uncertainties still need to be addressed in future work. A study on the robustness of evolutionary fuzzy control systems is given in [103]. [50] discuss the robustness of GA-tuned controllers when applied in the real world. Here are a few rules of thumb for robustness from past experience with flow control experiments. First, if the control mechanisms rely on large-scale dynamics, a controller learned in one condition is likely to perform well for a range of conditions. Second, learning control laws based on features

that scale with the operating conditions may be more relevant. This is for example the case in [49] where the Strouhal number, i.e., the non-dimensionalized frequency, is chosen instead of the frequency in Hertz unrelated to the velocity change. Third, robustness to different operating conditions can be included in the learning process. For instance, each individual can be evaluated in all operating conditions sequentially or with a transient [5, 129]. The downside is, of course, an increased evaluation time for each individual. [57] present a method for handling uncertainties in the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) for the reduction of thermoacoustic instabilities of gas turbine combustors. Due to the stochastic nature of the combustion process, the operating conditions are modified and the uncertainties are introduced in the cost function. The method allows online optimization of robust feedback controllers and achieves good performances even under highly unsteady operating conditions. The algorithm requires only a few additional evaluations per generation and is well-suited for online optimization. Robustness to hardware faults needs also to be addressed as the control may significantly vary if an actuator or sensor is damaged. An example of fault tolerant control based on CGP is proposed by [60].

22.5.3 Summary of Current Major Challenges for EML in Control

Following the no-free lunch theorem, no algorithm can solve all optimization problems and is on average more efficient than other algorithms. Thus, EML methods cannot be expected to outperform methods of control theory built for linear or linearizable problems. They are, however, efficient in deriving solutions for nonlinear, non-convex, non-differentiable, multimodal, noisy search spaces [128]. Nonetheless, there are major opportunities for improvement.

Although current studies work on the development of these topics, the need for control guarantees is fundamental to demonstrate the methods in industrially relevant environments and thus increase the technology readiness level. For systems with linear dynamics, optimality, stability, and robustness results are well-known. Whereas for nonlinear dynamics, optimized solutions are obtained for given operating conditions. Stability and robustness can be improved by including different conditions and transients in the learning process. Thus, transfer learning presents an opportunity to generalize and improve the performance of optimized control laws to out-of-design conditions. EML methods oriented for transfer learning purposes are a promising path toward learning more efficient controllers.

Another challenge is related to the human interpretability problem of machine-learned solutions. Contrary to reinforcement learning methods, for example, GP has the advantage of proposing solutions in the shape of mathematical expressions. However, these expressions are not directly interpretable as they are often bloated

expressions whose simplification is limited by the protected operations. A method for analysis, interpretation, and simplification of EML solutions is still needed.

A way to limit the bloated expressions is to learn control laws based on relevant features of the plant. Current methods consist of scaling the plant outputs, including past measurements, and are often based on engineering wisdom. So far, good results are achieved with such approaches but better performances can be expected with control-oriented features. Although control-oriented feature design is not specific to EML, it will certainly accelerate learning and enable better solutions. More generally, the efficiency of EML methods for control needs to integrate domain-specific knowledge for improved efficiency.

The learning process of EML methods tends to generate lots of data. The exploitation of this large data set to orient the learning process is still an untapped opportunity for EML. Thus, building a surrogate model of the plant based on the evaluated controllers may help to predict the next controller to evaluate. Moreover, the analysis of the data during the learning process enables online adjustment of the algorithms. Genetic operator's probabilities can then be tuned during the optimization to promote exploration or exploitation following the situation. Likewise, different genetic operators can be selected based on the learning stage. For control, the online analysis of the best and worst control laws may help to reduce the search space by selecting only the relevant control inputs, or by activating only the beneficial actuators. The generated data can also help to characterize the search space for selecting adequate optimization algorithms. In this manner, Exploratory Landscape Analysis [105, ELA] and its R implementation FLACCO [75] propose an automatic and quick extraction of low-level features for automatic algorithm selection.

22.6 Conclusions and Outlook

In this chapter, we reviewed Evolutionary Machine Learning (EML) to solve control tasks. Control presents the most difficult challenges in regression problems such as nonlinearities, high dimensionality, time delays, and external noises. Model-free methods such as the ones of EML are particularly fitted to tackle those challenges. EML has been successful in deriving or optimizing controllers in many fields like engineering, robotics, fluid mechanics, and industrial processes to cite a few examples. EML is employed at all stages of control fault detection, robustness analysis, and design of reliable systems [48]. EML has also been employed to design the plant, prior to its solving. Thus, evolutionary methods solve complex combinatorial problems such as sensor and actuator placement. Although EML algorithms are easy to develop and code, their implementation requires domain-specific knowledge to define rich enough controller spaces. Indeed, the design of the plant features is a critical step in turbulence control [17]. Again, evolutionary algorithms are used for feature construction [3]. Thereby, control is slightly drifting toward fully automated EML tasks, including model identification, feature design, control, stability, and

robustness analysis. Yet, any new task solved with evolutionary algorithms requires additional evaluations thus the need for accelerated learning.

Accelerated learning is pursued with the combination of EML with other optimization methods [23, 119]. In [96] and [33], GA and GP have both benefited from intermediate downhill simplex steps to exploit the local gradients in the search space and converge faster toward the minimum of the basin of attraction. Another approach to accelerate the learning is the use of digital twins learned during the optimization process. The idea is to leverage the huge amount of data generated during the learning to build a surrogate model of the plant. The number of evaluations can virtually be augmented by testing new controllers on the model. [48] already envision evolutionary adaptive control in the '00s with an online adaptation of the controller with a fast evaluation of models. Nowadays, with advances in artificial intelligence and deep neural networks, this is all the more relevant, in particular, since Moore's law is still valid. The learning acceleration opens the door to control with distributed-input distributed-put control. Indeed, as sensors and actuators become cheaper, more powerful, and reliable, large-scale multiple input and multiple output control become an attractive opportunity. So far, EML has been employed to learn controllers with $O(10)$ actuators and sensors, and new methods need to be developed for plants including $O(100)$ inputs and outputs. New challenges arise such as the explosion of the combinatorial complexity and improving the learning rate of EML is one of the keys to tackling it.

One of the most challenging issues to be tackled by EML in control is still the stability and robustness of controllers. In order to implement EML controllers in transport vehicles, energy systems, or production, control guarantees need to be certified. So far, there is no general methodology to analyze these machine-learned solutions. Especially, in the case of structure-free algorithms like GP, the solutions are generally complex and contain many redundancies. Human interpretability must be reintroduced afterward via simplification, analysis, and extraction of key terms that explain the controlled dynamics.

The fast growth of machine learning and artificial intelligence capabilities leads to a paradigm shift in control where controllers are learned directly from the plant or from data. Domain-specific knowledge is required afterward to interpret and generalize the solutions. The latter is an essential step for the widespread integration of EML results in all types of industries.

Acknowledgements This work is supported by the National Natural Science Foundation of China under grants 12302293, 12172109, and 12172111, by the Guangdong Basic and Applied Basic Research Foundation under grant 2022A1515011492, and by the Shenzhen Science and Technology Program under grant JCYJ20220531095605012.

References

1. Adánez, J.M., Al-Hadithi, B.M., Jiménez, A.: Multidimensional membership functions in T-S fuzzy models for modelling and identification of nonlinear multivariable systems using genetic algorithms. *Appl. Soft Comput.* **75**, 607–615 (2019)
2. Ajani, O.S., Mallipeddi, R.: Adaptive evolution strategy with ensemble of mutations for reinforcement learning. *Knowl. Based Syst.* **245**, 108624 (2022)
3. Al-Sahaf, H., Bi, Y., Chen, Q., Lensen, A., Mei, Y., Sun, Y., Tran, B., Xue, B., Zhang, M.: A survey on evolutionary machine learning. *J. R. Soc. N. Z.* **49**(2), 205–228 (2019)
4. Alrashdi, Z., Sayyafzadeh, M.: $(\mu + \lambda)$ Evolution strategy algorithm in well placement, trajectory, control and joint optimisation. *J. Pet. Sci. Eng.* **177**, 1042–1058 (2019)
5. Asai, S., Yamato, H., Sunada, Y., Rinoie, K.: Designing machine learning control law of dynamic bubble burst control plate for stall suppression. In: 2019 AIAA SciTech Forum, San Diego, CA. Paper 1899 (2021)
6. Barlow, G.J., Oh, C.K., Grant, E.: Incremental evolution of autonomous controllers for unmanned aerial vehicles using multi-objective genetic programming. In: IEEE Conference on Cybernetics and Intelligent Systems, 2004, vol. 2, pp. 689–694 (2004)
7. Barrios Aguilar, M.E., Vinicius Coury, D., Reginatto, R., Machado Monaro, R.: Multi-objective PSO applied to PI control of DFIG wind turbine under electrical fault conditions. *Electr. Power Syst. Res.* **180**, 106081 (2020)
8. Benard, N., Pons-Prats, J., Periaux, J., Bugeda, G., Braud, P., Bonnet, J.P., Moreau, E.: Turbulent separated shear flow control by surface plasma actuator: experimental optimization by genetic algorithm approach. *Exp. Fluids* **57**(2):22, 1–17 (2016)
9. Bersini, H.: Immune network and adaptive control. In: Proceedings of the 1st European conference on artificial life (ECAL), pp. 217–226. MIT Press (1991)
10. Bilal Pant, M., Zaheer, H., Garcia-Hernandez, L., Abraham, A.: Differential evolution: a review of more than two decades of research. *Eng. Appl. Artif. Intell.* **90**, 103479 (2020)
11. Binh, T.T., Korn, U.: An evolution strategy for the multiobjective optimization. In: Proceedings of the 2nd International Conference on Genetic Algorithms, pp. 23–28 (1996)
12. Blondin, M.J., Sanchis, J., Sicard, P., Herrero, J.M.: New optimal controller tuning method for an AVR system using a simplified ant colony optimization with a new constrained Nelder-Mead algorithm. *Appl. Soft Comput.* **62**, 216–229 (2018)
13. Blume, C., Jakob, W.: GLEAM—an evolutionary algorithm for planning and control based on evolution strategy. In: GECCO Late Breaking Papers, pp. 31–38 (2002)
14. Boumediene, S., Chouraqui, S., Belkacem, S.: A genetic algorithm-based neuro-fuzzy controller for unmanned aerial vehicle control. *Int. J. Appl. Metaheuristic Comput.* **13**(1), 1–23 (2022)
15. Bounar, N., Labdai, S., Boulkroune, A.: PSO-GSA based fuzzy sliding mode controller for DFIG-based wind turbine. *ISA Trans.* **85**, 177–188 (2019)
16. Brameier, M., Banzhaf, W.: Linear Genetic Programming. Springer Science & Business Media (2006)
17. Brunton, S.L., Noack, B.R.: Closed-loop turbulence control: progress and challenges. *Appl. Mech. Rev.* **67**(5):050801, 01–48 (2015)
18. Brunton, S.L., Noack, B.R., Koumoutsakos, P.: Machine learning for fluid mechanics. *Ann. Rev. Fluid Mech.* **52**, 477–508 (2020)
19. Bufu, B., Zhancheng, W., Yangsheng, X.: Multi-objective genetic algorithm for hybrid electric vehicle parameter optimization. In: 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 5177–5182 (2006)
20. Bull, L., Sha'Aban, J., Tomlinson, A., Addison, J.D., Heydecker, B.G.: Towards distributed adaptive control for road traffic junction signals using learning classifier systems. In: Bull, L. (eds.) Applications of Learning Classifier Systems. Studies in Fuzziness and Soft Computing, vol. 150. Springer, Berlin, Heidelberg (2004)

21. Burbidge, R., Walker, J.H., Wilson, M.S.: Grammatical evolution of a robot controller. In: 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 357–362 (2009)
22. Burbidge, R., Wilson, M.S.: Vector-valued function estimation by grammatical evolution for autonomous robot control. *Inf. Sci.* **258**, 182–199 (2014)
23. Castellanos, R., Cornejo Maceda, G.Y., de la Fuente, I., Noack, B.R., Ianiro, A., Discetti, S.: Machine learning flow control with few sensor feedback and measurement noise. *Phys. Fluids*. **34**(4):047118, 1–17 (2022)
24. Cha, Y.-J., Agrawal, A.K., Kim, Y., Raich, A.M.: Multi-objective genetic algorithms for cost-effective distributions of actuators and sensors in large structures. *Expert. Syst. Appl.* **39**(9), 7822–7833 (2012)
25. Cha, Y.-J., Raich, A.M., Barroso, L., Agrawal, A.: Optimal placement of active control devices and sensors in frame structures using multi-objective genetic algorithms. *Struct. Control. Health Monit.* **20**(1), 16–44 (2013)
26. Chao, K.-H., Rizal, M.N.: A hybrid MPPT controller based on the genetic algorithm and ant colony optimization for photovoltaic systems under partially shaded conditions. *Energy*. **14**(10) (2021)
27. Chen, C., Li, M., Sui, J., Wei, K., Pei, Q.: A genetic algorithm-optimized fuzzy logic controller to avoid rear-end collisions. *J. Adv. Transp.* **50**(8), 1735–1753 (2016)
28. Cheong, F., Lai, R.: Constraining the optimization of a fuzzy logic controller using an enhanced genetic algorithm. *IEEE Trans. Syst. Man Cybern. Part B (Cybern.)* **30**(1), 31–46 (2000)
29. Cheong, F., Lai, R.: Simplifying the automatic design of a fuzzy logic controller using evolutionary programming. *Soft Comput.* **11**(9), 839–846 (2007)
30. Chipperfield, A.J., Dakev, N.V., Fleming, P.J., Whidborne, J.F.: Multiobjective robust control using evolutionary algorithms. In: Proceedings of the IEEE International Conference on Industrial Technology (ICIT'96), pp. 269–273 (1996)
31. Choi, Y.-K., Park, J.-H., Kim, H.-S., Kim, J.H.: Optimal trajectory planning and sliding mode control for robots using evolution strategy. *Robot.* **18**(4), 423–428 (2000)
32. Chovet, C., Keirsbulck, L., Noack, B.R., Lippert, M., Foucaut, J.-M.: Machine learning control for experimental shear flows targeting the reduction of a recirculation bubble. In: The 20th World Congress of the International Federation of Automatic Control (IFAC), Toulouse, France, pp. 1–4 (2017)
33. Cornejo Maceda, G.Y., Li, Y., Lusseyran, F., Morzyński, M., Noack, B.R.: Stabilization of the fluidic pinball with gradient-enriched machine learning control. *J. Fluid Mech.* **917**:A42, 1–43 (2021)
34. Cornejo Maceda, G.Y., Lusseyran, F., Noack, B.R.: xMLC—A Toolkit for Machine Learning Control, vol. 2. Machine learning tools in fluid mechanics. Technische Universität Braunschweig, Braunschweig, first edition (2022)
35. Cornejo Maceda, G.Y., Noack B.R., Lusseyran, F., Deng, N., Pastur, L., Morzyński, M.: Artificial intelligence control applied to drag reduction of the fluidic pinball. *Proc. Appl. Math. Mech.* **19**(1):e201900268, 1–2 (2019)
36. Cornejo Maceda, G.Y., Varon, E., Lusseyran, F., Noack, B.R.: Stabilization of a multi-frequency open cavity flow with gradient-enriched machine learning control. *J. Fluid Mech.* **955**, A20 (2023)
37. Debien, A., von Krbek, K.A.F.F., Mazellier, N., Duriez, T., Cordier, L., Noack, B.R., Abel, M.W., Kourta, A.: Closed-loop separation control over a sharp-edge ramp using genetic programming. *Exp. Fluids*. **57**(3):40, 1–19 (2016)
38. Deepak, B.B.V.L., Parhi, D.R.: Control of an automated mobile manipulator using artificial immune system. *J. Exp. & Theor. Artif. Intell.* **28**(1–2), 417–439 (2016)
39. del Valle, Y., Venayagamoorthy, G.K., Mohagheghi, S., Hernandez, J.-C., Harley, R.G.: Particle swarm optimization: basic concepts, variants and applications in power systems. *IEEE Trans. Evol. Comput.* **12**(2), 171–195 (2008)
40. Denisova, L.A., Meshcheryakov, V.A.: Control system synthesis based on multicriteria optimization using genetic algorithm. In: 2017 Dynamics of Systems, Mechanisms and Machines (Dynamics), pp. 1–5 (2017)

41. Diveev, A., Shmalko, E.: Machine Learning Control by Symbolic Regression. Springer, Cham (2021)
42. Diveev, A., Sofronova, E., Prisca, D.M.C.: Synthesised optimal control for a robotic group by complete binary genetic programming. In: 2021 IEEE 16th Conference on Industrial Electronics and Applications (ICIEA), pp. 100–105 (2021)
43. Doyle, J.C.: Guaranteed margins for LQG regulators. *IEEE Trans. Autom. Control.* **23**(4), 756–757 (1978)
44. Dracopoulos, D.C.: Evolutionary Learning Algorithms for Neural Adaptive Control. Springer-Verlag (1997)
45. Duriez, T., Brunton, S.L., Noack, B.R.: Machine Learning Control—Taming Nonlinear Dynamics and Turbulence. Fluid Mechanics and its Applications, vol. 116. Springer-Verlag (2017)
46. Duriez, T., Parezanović, V., Laurentie, J.-C., Fourment, C., Delville, J., Bonnet, J.-P., Cordier, L., Noack, B.R., Segond, M., Abel, M.W., Gautier, N., Aider, J.-L., Raibaud, C., Cuvier, C., Stanislas, M., Brunton, S.: Closed-loop control of experimental shear layers using machine learning (invited). In: 7th AIAA Flow Control Conference, pp. 1–16, Atlanta, Georgia, USA (2014)
47. El-Sayed, Y., Oswald, P., Sattler, S., Pradeep, K., Radespiel, R., Behr, C., Sinapius, M., Petersen, J., Wierach, P., Quade, M., Abel, M., Noack, B.R., Semaan, R.: Open-and closed-loop control investigations of unsteady Coanda actuation on a high-lift configuration. In: AIAA Aviation, pp. 1–13, Atlanta, Georgia, USA. AIAA 2018-3684 (2019)
48. Fleming, P.J., Purshouse, R.C.: Evolutionary algorithms in control systems engineering: a survey. *Control. Eng. Pract.* **10**(11), 1223–1241 (2002)
49. Gautier, N., Aider, J.-L., Duriez, T., Noack, B.R., Segond, M., Abel, M.W.: Closed-loop separation control using machine learning. *J. Fluid Mech.* **770**, 424–441 (2015)
50. Gongora, M.A., Passow, B.N., Hopgood, A.A.: Robustness analysis of evolutionary controller tuning using real systems. In: 2009 IEEE Congress on Evolutionary Computation, pp. 606–613 (2009)
51. Gray, G.J., Murray-Smith, D.J., Li, Y., Sharman, K.C., Weinbrenner, T.: Nonlinear model structure identification using genetic programming. *Control. Eng. Pract.* **6**(11), 1341–1352 (1998)
52. Guenounou, O., Belmehdi, A., Dahhou, B.: Multi-objective optimization of TSK fuzzy models. *Expert. Syst. Appl.* **36**(4), 7416–7423 (2009)
53. Guenounou, O., Dahhou, B., Athmani, B.: Optimal design of PID controller by multi-objective genetic algorithms. In: International Conference on Computer Related Knowledge (ICCRK' 2012), p. 6p, Sousse, Tunisia (2012)
54. Guo, X., Ji, M., Zhao, Z., Wen, D., Zhang, W.: Global path planning and multi-objective path control for unmanned surface vehicle based on modified particle swarm optimization (PSO) algorithm. *Ocean. Eng.* **216**, 107693 (2020)
55. Guzman Razo, D.E., Müller, B., Madsen, H., Wittwer, C.: A genetic algorithm approach as a self-learning and optimization tool for PV power simulation and digital twinning. *Energy*. **13**(24) (2020)
56. Han, P., Zhou, S., Wang, D.: A multi-objective genetic programming/ NARMAX approach to chaotic systems identification. In: 2006 6th World Congress on Intelligent Control and Automation, vol. 1, pp. 1735–1739 (2006)
57. Hansen, N., Niederberger, A.S.P., Guzzella, L., Koumoutsakos, P.: A method for handling uncertainty in evolutionary optimization with an application to feedback control of combustion. *IEEE Trans. Evol. Comput.* **13**(1), 180–197 (2009)
58. Harding, S., Miller, J.F.: Evolution of robot controller using Cartesian genetic programming. In: Keijzer, M., Tettamanzi, A., Collet, P., van Hemert, J., Tomassini, M. (eds.) *Genetic Programming*, pp. 62–73. Springer, Berlin Heidelberg (2005)
59. Hervé, A., Sipp, D., Schmid, P.J., Samuelides, M.: A physics-based approach to flow control using system identification. *J. Fluid Mech.* **702**, 26–58 (2012)

60. Hirayama, Y., Clarke, T., Miller, J.F.: Fault tolerant control using Cartesian genetic programming. In: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, New York, NY, USA, pp. 1523–1530. Association for Computing Machinery (2008)
61. Hoffmann, F.: Evolutionary algorithms for fuzzy control system design. Proc. IEEE. **89**(9), 1318–1333 (2001)
62. Holland, J.H.: Adaptation in Natural and Artificial Systems. The University of Michigan Press, Ann Arbor (1975)
63. Hu, N., Zhong, J., Zhou, J.T., Zhou, S., Cai, W., Monterola, C.: Guide them through: an automatic crowd control framework using multi-objective genetic programming. Appl. Soft Comput. **66**, 90–103 (2018)
64. Humaidi, A.J., Badr, H.M., Hameed, A.H.: PSO-based active disturbance rejection control for position control of magnetic levitation system. In: 2018 5th International Conference on Control, Decision and Information Technologies (CoDIT), pp. 922–928 (2018)
65. Hurst, J., Bull, L., Melhuish, C.: TCS learning classifier system controller on a real robot. In: Guervós, J.J.M., Adamidis, P., Beyer, H.-G., Schwefel, H.-P., Fernández-Villacañas, J.-L. (eds.) Parallel Problem Solving from Nature—PPSN VII, pp. 588–597. Springer, Berlin, Heidelberg (2002)
66. Indragandhi, V., Logesh, R., Subramaniyaswamy, V., Vijayakumar, V., Siarry, P., Uden, L.: Multi-objective optimization and energy management in renewable based AC/DC microgrid. Comput. & Electr. Eng. **70**, 179–198 (2018)
67. Jaafar, H.I., Mohamed, Z., Mohd Subha, N.A., Husain, A.R., Ismail, F.S., Ramli, L., Tokhi, M.O., Shamsudin, M.A.: Efficient control of a nonlinear double-pendulum overhead crane with sensorless payload motion using an improved PSO-tuned PID controller. J. Vib. Control. **25**(4), 907–921 (2019)
68. Jahedi, G., Ardehali, M.M.: Genetic algorithm-based fuzzy-PID control methodologies for enhancement of energy efficiency of a dynamic energy system. Energy Convers. Manag. **52**(1), 725–732 (2011)
69. Javadi-Moghaddam, J., Bagheri, A.: An adaptive neuro-fuzzy sliding mode based genetic algorithm control system for under water remotely operated vehicle. Expert. Syst. Appl. **37**(1), 647–660 (2010)
70. Jia, J., Chen, J., Chang, G., Tan, Z.: Energy efficient coverage control in wireless sensor networks based on multi-objective genetic algorithm. Comput. & Math. Appl. **57**(11), 1756–1766 (2009)
71. Kadlic, B., Sekaj, I., Pernecký, D.: Design of continuous-time controllers using Cartesian genetic programming. IFAC Proc. Vol. **47**(3), 6982–6987 (2014)
72. Kalmanje, K.K., Neidhoefer, J.: Immunized Adaptive Critic for an Autonomous Aircraft Control Application. Springer, Berlin, Heidelberg (1999)
73. Kane, M.B.: Machine learning control for floating offshore wind turbine individual blade pitch control. In: 2020 American Control Conference (ACC), pp. 237–241 (2020)
74. Karlsen, M.R., Moschoyannis, S.: Evolution of control with learning classifier systems. Appl. Netw. Sci. **3**(1), 30 (2018)
75. Kerschke, P., Trautmann, H.: The R-package FLACCO for exploratory landscape analysis with applications to multi-objective optimization problems. In: 2016 IEEE Congress on Evolutionary Computation (CEC), pp. 5262–5269 (2016)
76. Kim, D.H.: Tuning of a PID controller using immune network model and fuzzy set. In: ISIE 2001. 2001 IEEE International Symposium on Industrial Electronics Proceedings (Cat. No.01TH8570), vol. 3, pp. 1656–1661 (2001)
77. Kim, H.-S., Kang, J.W.: Semi-active fuzzy control of a wind-excited tall building using multi-objective genetic algorithm. Eng. Struct. **41**, 242–257 (2012)
78. Kondo, T., Ishiguro, A., Watanabe, Y., Shirai, Y., Uchikawa, Y.: Evolutionary construction of an immune network-based behavior arbitration mechanism for autonomous mobile robots. Electr. Eng. Jpn. **123**, 1–10 (1998)

79. Koza, J.R.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press, Boston (1992)
80. Koza, J.R.: Human-competitive results produced by genetic programming. *Genet. Program. Evolvable Mach.* **11**(3), 251–284 (2010)
81. Koza, J.R., Bennett, F.H., Keane, M.A., Andre, D.: Automatic programming of a time-optimal robot controller and an analog electrical circuit to implement the robot controller by means of genetic programming. In: *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97. ‘Towards New Computational Principles for Robotics and Automation’*, pp. 340–346 (1997)
82. Kristinsson, K., Dumont, G.A.: System identification and control using genetic algorithms. *IEEE Trans. Syst. Man Cybern.* **22**(5), 1033–1046 (1992)
83. Kumar, P.M., Devi G, U., Manogaran, G., Sundarasekar, R., Chilamkurti, N., Varatharajan, R.: Ant colony optimization algorithm with internet of vehicles for intelligent traffic control system. *Comput. Netw.* **144**, 154–162 (2018)
84. Kumar, S., Parhi, D.R., Muni, M.K., Pandey, K.K.: Optimal path search and control of mobile robot using hybridized sine-cosine algorithm and ant colony optimization technique. *Ind. Robot.* **47**, 535–545 (2020)
85. Lau, H.Y.K., Wong, V.W.K., Lee, I.S.K.: Immunity-based autonomous guided vehicles control. *Appl. Soft Comput.* **7**(1), 41–57 (2007)
86. Li, H., Tan, J., Gao, Z., Noack, B.R.: Machine learning open-loop control of a mixing layer. *Phys. Fluids.* **32**(111701), 1–7 (2020)
87. Li, R., Noack, B.R., Cordier, L., Borée, J., Harambat, F.: Drag reduction of a car model by linear genetic programming control. *Exp. Fluids.* **58**(103), 1–20 (2017)
88. Li, R., Noack, B.R., Cordier, L., Borée, J., Kaiser, E., Harambat, F.: Linear genetic programming control for strongly nonlinear dynamics with frequency crosstalk. *Arch. Mech.* **70**(6), 505–534 (2018)
89. Li, Z., Wang, X., Du, W., Yang, M., Li, Z., Liao, P.: Data-driven adaptive predictive control of hydrocracking process using a covariance matrix adaption evolution strategy. *Control. Eng. Pract.* **125**, 105222 (2022)
90. Liang, H., Zou, J., Zuo, K., Khan, M.J.: An improved genetic algorithm optimization fuzzy controller applied to the wellhead back pressure control system. *Mech. Syst. Signal Process.* **142**, 106708 (2020)
91. Lin, T.Y., Jia, Z., Yang, C., Xiao, Y., Lan, S., Shi, G., Zeng, B., Li, H.: Evolutionary digital twin: a new approach for intelligent industrial product development. *Adv. Eng. Inform.* **47**, 101209 (2021)
92. Liu, H., Wu, K., Huang, K., Cheng, G., Wang, R., Liu, G.: Optimization of large-scale UAV cluster confrontation game based on integrated evolution strategy. *Clust. Comput.* (2023)
93. Liu, X., Jiang, D., Tao, B., Jiang, G., Sun, Y., Kong, J., Tong, X., Zhao, G., Chen, B.: Genetic algorithm-based trajectory optimization for digital twin robots. *Front. Bioeng. Biotechnol.* **9** (2022)
94. Llera, J.R., Deb, K., Runkle, E., Xu, L., Goodman, E.: Evolving and comparing greenhouse control strategies using model-based multi-objective optimization. In: *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 1929–1936 (2018)
95. Loiseau, J.-C., Noack, B.R., Brunton, S.L.: Sparse reduced-order modeling: Sensor-based dynamics to full-state estimation. *J. Fluid Mech.* **844**, 459–490 (2018)
96. Maehara, N., Shimoda, Y.: Application of the genetic algorithm and downhill simplex methods (Nelder-Mead methods) in the search for the optimum chiller configuration. *Appl. Therm. Eng.* **61**(2), 433–442 (2013)
97. Mahmoodabadi, M.J., Babak, N.R.: Robust fuzzy linear quadratic regulator control optimized by multi-objective high exploration particle swarm optimization for a 4 degree-of-freedom quadrotor. *Aerosp. Sci. Technol.* **97**, 105598 (2020)
98. Khan, M.M., Ahmad, A.M., Khan, G.M., Miller, J.F.: Fast learning neural networks using Cartesian genetic programming. *Neurocomputing* **121**, 274–289 (2013)

99. Khan, M.M., Khan, G.M., Miller, J.F.: Evolution of optimal ANNs for non-linear control problems using Cartesian genetic programming. In: International Conference on Artificial Intelligence (2010)
100. Maki, A., Akimoto, Y., Umeda, N.: Application of optimal control theory based on the evolution strategy (CMA-ES) to automatic berthing (part: 2). *J. Mar. Sci. Technol.* **26**, 835–845 (2021)
101. Maki, A., Sakamoto, N., Akimoto, Y., Nishikawa, H., Umeda, N.: Application of optimal control theory based on the evolution strategy (CMA-ES) to automatic berthing. *J. Mar. Sci. Technol.* **25**, 221–233 (2020)
102. McKay, R.I., Hoai, N.X., Whigham, P.A., Shan, Y., O'Neill, M.: Grammar-based genetic programming: a survey. *Genet. Program. Evolvable Mach.* **11**, 365–396 (2010)
103. Meghdadi, A.H.: On robustness of evolutionary fuzzy control systems. In: IEEE Annual Meeting of the Fuzzy Information, 2004. Processing NAFIPS '04, vol. 1, pp. 254–258 (2004)
104. Melin, P., Castillo, O.: Intelligent control of complex electrochemical systems with a neuro-fuzzy-genetic approach. *IEEE Trans. Ind. Electron.* **48**(5), 951–955 (2001)
105. Mersmann, O., Bischi, B., Trautmann, H., Preuss, M., Weihns, C., Rudolph, G.: Exploratory landscape analysis. In: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO '11, New York, NY, USA, pp. 829–836. Association for Computing Machinery (2011)
106. Michalewicz, Z.: Binary or Float?, pp. 97–106. Springer, Berlin, Heidelberg (1996)
107. Miller, J.F.: Cartesian Genetic Programming, pp. 17–34. Springer, Berlin, Heidelberg (2011)
108. Mitsumoto, N., Fukuda, T., Arai, F., Tadashi, H., Idogaki, T.: Self-organizing multiple robotic system (a population control through biologically inspired immune network architecture). In: Proceedings of IEEE International Conference on Robotics and Automation, vol. 2, pp. 1614–1619 (1996)
109. Montazeri, A., West, C., Monk, S.D., Taylor, C.J.: Dynamic modelling and parameter estimation of a hydraulic robot manipulator using a multi-objective genetic algorithm. *Int. J. Control.* **90**(4), 661–683 (2017)
110. Mughees, A., Mohsin, S.A.: Design and control of magnetic levitation system by optimizing fractional order PID controller using ant colony optimization algorithm. *IEEE Access.* **8**, 116704–116723 (2020)
111. Nasruddin, S., Satrio, P., Mahlia, T.M.I., Giannetti, N., Saito, K.: Optimization of hvac system energy consumption in a building using artificial neural network and multi-objective genetic algorithm. *Sustain. Energy Technol. Assess.* **35**, 48–57 (2019)
112. Nazaruddin, Y.Y., Andritini, A.D., Anditio, B.: PSO based PID controller for quadrotor with virtual sensor. *IFAC-PapersOnLine.* **51**(4), 358–363 (2018)
113. Noack, B.R.: Closed-loop turbulence control—from human to machine learning (and retour). In: Zhou, Y., Kimura, M., Peng, G., Lucey, A.D., Hung, L. (eds.) Fluid-Structure-Sound Interactions and Control. In: Proceedings of the 4th Symposium on Fluid-Structure-Sound Interactions and Control, pp. 23–32. Springer (2019)
114. Nordin, P., Banzhaf, W.: An on-line method to evolve behavior and to control a miniature robot in real time with genetic programming. *Adapt. Behav.* **5**(2), 107–140 (1997)
115. Oh, C.K., Barlow, G.J.: Autonomous controller design for unmanned aerial vehicles using multi-objective genetic programming. In: Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753), vol. 2, pp. 1538–1545 (2004)
116. Parezanović, V., Cordier, L., Spohn, A., Duriez, T., Noack, B.R., Bonnet, J.-P., Segond, M., Abel, M., Brunton, S.L.: Frequency selection by feedback control in a turbulent shear flow. *J. Fluid Mech.* **797**, 247–283 (2016)
117. Parezanović, V., Laurentie, J.C., Fourment, C., Cordier, L., Noack, B.R., Shaqarin, T.: Modification of global properties of a mixing layer by open/closed loop actuation. In: Proceedings of the 8th International Symposium On Turbulent and Shear Flow Phenomena (2013)
118. Passaggia, P.-Y., Quansah, A., Mazellier, N., Cornejo Maceda G.Y., Kourta, A.: Real-time feedback stall control of an airfoil at large Reynolds numbers using linear genetic programming. *Phys. Fluids.* **34**(4), 045108 (2022)

119. Pino, F., Schena, L., Rabault, J., Mendez, M.A.: Comparative analysis of machine learning methods for active flow control. *J. Fluid Mech.* **958**, A39 (2023)
120. Rahman, M., Ong, Z.C., Chong, W.T., Julai, S., Ng, X.W.: Wind turbine tower modeling and vibration control under different types of loads using ant colony optimized PID controller. *Arab. J. Sci. Eng.* **44**, 707–720 (2019)
121. Ramirez-Atencia, C., Bello-Orgaz, G., R-Moreno, M.D., Camacho, D.: Solving complex multi-UAV mission planning problems using multi-objective genetic algorithms. *Soft Comput.* **21**(17), 4883–4900 (2017)
122. Refoufi, S., Benmohammed, K.: Control of a manipulator robot by neuro-fuzzy subsets form approach control optimized by the genetic algorithms. *ISA Trans.* **77**, 133–145 (2018)
123. Ren, F., Hu, H.-B., Tang, H.: Active flow control using machine learning: a brief review. *J. Hydodyn.* **32**(2), 247–253 (2020)
124. Ren, F., Wang, C., Tang, H.: Active control of vortex-induced vibration of a circular cylinder using machine learning. *Phys. Fluids.* **31**(9), 093601 (2019)
125. Richter, U.M.: Controlled self-organisation using learning classifier systems. KIT Scientific Publishing (2009)
126. Rodríguez-Molina, A., Mezura-Montes, E., Villarreal-Cervantes, M.G., Aldape-Pérez, M.: Multi-objective meta-heuristic optimization in intelligent control: a survey on the controller tuning problem. *Appl. Soft Comput.* **93**, 106342 (2020)
127. Sahu, C., Parhi, D.R., Kumar, P.B.: An approach to optimize the path of humanoids using adaptive ant colony optimization. *J. Bionic Eng.* **15**, 623–635 (2018)
128. Schwefel, H.-P.: Advantages (and disadvantages) of evolutionary computation over other approaches. In: *Evolutionary Computation 1*, pp. 58–60. CRC Press (2018)
129. Semaan, R., Oswald, P., Cornejo Maceda, G.Y., Noack, B.R.: Aerodynamic optimization of a generic light truck under unsteady conditions using gradient-enriched machine learning control. *Exp. Fluids.* **64**(3), 59 (2023)
130. Singh, K., Singh, K., Son, L.H., Aziz, A.: Congestion control in wireless sensor networks by hybrid multi-objective optimization algorithm. *Comput. Netw.* **138**, 90–107 (2018)
131. Slowik, A., Kwasnicka, H.: Evolutionary algorithms and their applications to engineering problems. *Neural Comput. Appl.* **32**(16), 12363–12379 (2020)
132. Slowik, A., Kwasnicka, H.: Nature inspired methods and their industry applications-Swarm intelligence algorithms. *IEEE Trans. Ind. Inform.* **14**(3), 1004–1015 (2018)
133. Stein, A., Tomforde, S.: Reflective learning classifier systems for self-adaptive and self-organising agents. In: *2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*, pp. 139–145 (2021)
134. Syed, M.K., Ram, B.V.S.: A genetic algorithm optimized fuzzy logic controller for shunt active power filter. In: *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, pp. 1892–1896 (2016)
135. Tan, Q., Dai, P., Zhang, Z., Katupitiya, J.: MPC and PSO based control methodology for path tracking of 4WS4WD vehicles. *Appl. Sci.* **8**(6) (2018)
136. Taney, I.: Genetic programming incorporating biased mutation for evolution and adaptation of Snakebot. *Genet. Program. Evolvable Mach.* **8**, 39–59 (2007)
137. Timmis, J., Knight, T., de Castro, L.N., Hart, E.: An overview of artificial immune systems. In: *Computation in Cells and Tissues: Perspectives and Tools of Thought*, pp. 51–91 (2004)
138. Urbanowicz, R.J., Moore, J.H.: Learning classifier systems: a complete introduction, review, and roadmap. *J. Artif. Evol. Appl.* **2009**, 25 (2009)
139. Wang, J., Ju, C., Gao, Y., Sangaiah, A.K., Kim, G.-J.: A PSO based energy efficient coverage control algorithm for wireless sensor networks. *Comput. Mater. Contin.* **56**(3), 433–446 (2018)
140. Wang, M., Dong, H., Li, X., Zhang, Y., Yu, J.: Control and optimization of a bionic robotic fish through a combination of CPG model and PSO. *Neurocomputing.* **337**, 144–152 (2019)
141. Watanabe, Y., Ishiguro, A., Uchikawa, Y.: Decentralized Behavior Arbitration Mechanism for Autonomous Mobile Robot Using Immune Network, pp. 187–209. Springer, Berlin, Heidelberg (1999)

142. Wu, Z., Fan, D., Li, R., Noack, B.R.: Jet mixing optimization using machine learning control. *Exp. Fluids.* **59**(8), 131 (2018)
143. Yamada, S., Sato, R., Tamaki, T., Kita, E.: Control program design of autonomous vehicle robot using grammatical evolution. In: 2021 6th International Conference on Robotics and Automation Engineering (ICRAE), pp. 308–312 (2021)
144. Yu, W., Li, B., Jia, H., Zhang, M., Wang, D.: Application of multi-objective genetic algorithm to optimize energy efficiency and thermal comfort in building design. *Energy Build.* **88**, 135–143 (2015)
145. Zhen, X., Enze, Z., Qingwei, C.: Rotary unmanned aerial vehicles path planning in rough terrain based on multi-objective particle swarm optimization. *J. Syst. Eng. Electron.* **31**(1), 130–141 (2020)
146. Zhou, Y., Fan, D., Zhang, B., Li, R., Noack, B.R.: Artificial intelligence control of a turbulent jet. *J. Fluid Mech.* **897**:A27, 1–46 (2020)
147. Zupančič, J., Filipič, B., Gams, M.: Genetic-programming-based multi-objective optimization of strategies for home energy-management systems. *Energy.* **203**, 117769 (2020)

Chapter 23

Evolutionary Machine Learning in Robotics



Eric Medvet, Giorgia Nadizar, Federico Pigozzi, and Erica Salvato

Abstract In this chapter, we survey the most significant applications of EML to robotics. We first highlight the salient characteristics of the field in terms of what can be optimized and with what aims and constraints. Then we survey the large literature concerning the optimization, by the means of evolutionary computation, of artificial neural networks, traditionally considered a form of machine learning, used for controlling the robots: for easing the comprehension, we categorize the various approaches along different axes, as, e.g., the robotic task, the representation of the solutions, the evolutionary algorithm being employed. We then survey the many usages of evolutionary computation for optimizing the morphology of the robots, including those that tackle the challenging task of optimizing the morphology and the controller at the same time. Finally, we discuss the reality gap problem that consists in a potential mismatch between the quality of solutions found in simulations and their quality observed in reality.

23.1 Robot Optimization and its Peculiarities

The field of robotics involves the design, construction, and operation of *robots*. In this chapter, we define as robot any agent, be it real or simulated, which can interact with an *environment*. We require said interaction to be bidirectional, meaning that the agent can affect the environment with its actions, and is in turn affected by the environment, either through sensory perceptions or simple mechanical effects.

A robot is defined by its body and its controller, which are deeply interconnected. The *controller* (often called brain) is responsible for making decisions concerning the actions to be made. Oftentimes, such decisions consist of computing control values which are sent to the body, which then actuates them. The *body* of the agent involves all the physical aspects of the robot, such as its external aspect, its

E. Medvet (✉) · G. Nadizar · F. Pigozzi · E. Salvato
University of Trieste, Trieste, Italy
e-mail: emedvet@units.it

constituting materials, or its modules and connecting parts (joints). In addition, the body is responsible for actuating actions, and also for proprioception and environmental awareness.

In general, the actions of a robot are driven toward the achievement of a goal, which we define as the *task* of the agent. Within this paradigm, it is possible, and often desirable, to *optimize* the agent for a task, that is, achieving a design that enables the robot to most successfully accomplish its goal. However, due to the complexity of the design process, given the usually extremely vast search space, handcrafting satisfying solutions is practically unfeasible. Therefore, automatic optimization techniques play a fundamental role in assisting designers in the achievement of successful robots. Among those, evolutionary algorithms (EAs) stand out for their effectiveness and efficiency in exploring the solution space, yielding to extremely successful results with relatively little human effort. In particular, in most cases, they only require to define a suitable measure of quality (i.e., the *fitness*) of the solution under optimization (i.e., the robot): while defining such fitness is not always easy [34], it is in general much easier than attempting to manually design the solution.

The application of evolutionary computation (EC) to robotics constitutes the field of *evolutionary robotics* (ER) [36, 129]: in a broad sense, ER consists in optimizing the robot (or some of its parts) for a given task, using EC. In this chapter, we deal with EML in robotics, which can be considered the sub-field of ER where ML is somehow involved. As a matter of fact, the vast majority of EML applications to robotics deal with the optimization of a robot controller which is based on an artificial neural network (ANN): indeed, ANNs are traditionally considered an artifact belonging to the field of machine learning (ML). As a consequence, the largest part of this chapter, namely Sect. 23.2, is devoted to surveying the body of literature dealing with evolutionary optimization of ANN-based controllers for robotic agents. There are, however, other artifacts that can be employed as controllers and can be optimized with EC: for example, behavioral trees can be evolved using genetic programming. From the point of view of the definition given in this book, they can be considered at the boundary of EML. We survey some of these approaches in Sect. 23.2.2.

From a broader point of view, robotics is a field that exhibits a few peculiarities that are relevant to the usage of EC as a form of optimization.

First, optimization can be performed at various levels, targeting different features of a robot. Namely, most applications of EML in robotics are either aimed at optimizing the controller, the body of the agent, or both, but there are also some more peculiar examples of EML in robotics, as we will see in Sect. 23.4. It is worth to note that, even when just the controller is subjected to optimization, the body still plays a relevant role, because it actively takes part in the way the robot interacts with the environment by processing perception to decide actions to be performed; in other words, the body is capable of performing some morphological computation [63]. This phenomenon is captured by the embodied cognition paradigm [140].

Second, robots are not static artifacts, but change over time. In the simplest case, the only thing that changes is their spatial configuration, i.e., their position within the environment or the relative position of their components. In more complex scenarios, they can suffer malfunctions [96, 105] or even grow over time [94, 120]. What is of

key importance, however, is that a robot existence (or life) is not instantaneous: hence, the evolution time-scale may interact with the life time-scale and adaptation can occur at both levels. This opportunity has been exploited by researchers for combining EC with learning, morphological development, or other forms of adaptation in order to obtain robots that are eventually better in performing their task.

Third, robots, and the environment they are immersed in, are usually complex and sometimes dangerous. The straightforward application of the main EC steps, i.e., evaluation, selection, and variation, is often not feasible using the real robot in its real environment, because the evaluation stage is too costly, not scalable, or even dangerous (for the robot itself, the environment, or other human operators involved in the optimization process). For this reason, the optimization is often carried out in simulation, possibly exploiting computing machinery that allows fast computation and well fits the inherent feature of EAs to be massively parallelizable. However, the simulation is rarely capable of capturing each subtle aspect of the reality and this can eventually result in an optimized robot whose behavior in reality is different to the one observed in simulation. This problem is known as the reality gap problem: we discuss it in Sect. 23.5.

23.2 EML for Controller Optimization

The controller of a robot is responsible for deciding which actions need to be performed. In many practical cases, this boils down to computing the control values that are sent to the body to guide actuation. Since robots are *embodied* agents, it is often convenient to equip them with sensors of various kinds, whose readings can be used as feedback by the controller to effectively guide movements (closed-loop controllers). For these controllers, the control values are, broadly speaking, a function of the sensor readings and possibly of past experience, in case there is some form of memory.

In the field of ML, ANNs are everywhere being deployed to approximate functions for solving a very diverse variety of tasks. The robotics domain is no exception, and ANNs are among the most used tool within the agent controller to compute its next actions. Since engineering ANNs is not an easy task, requiring a lot of domain expert knowledge, experience, and trial and error, EAs often come in handy for obtaining well-optimized ANNs that can successfully control artificial agents. More in details, neuroevolution (NE) has been beneficially applied both for neural architecture search (NAS) and for training various flavors of ANNs (see Sect. 23.2.1). In addition, EAs have also been combined with Reinforcement Learning (RL) to effectively train ANNs to solve robotics tasks.

Aside from ANNs, several other approaches have been proposed merging ML techniques with evolutionary optimization for robot control, as we will detail in Sect. 23.2.2. Among them, we can include behavior trees and more classical control theory approaches.

23.2.1 Neuroevolution

Neuroevolution (NE) is the sub-field of evolutionary computation that deals with ANNs. Here, NE has the objective of searching for a good robot controller, i.e., a good robot brain, meaning that it has to effectively search the space of ANNs, eventually finding the most suitable one for guiding an agent toward the accomplishment of its task. Several works exist in which NE has been applied to optimize robot controllers. Even though it is not feasible to analyze each of them in detail, we aim at providing an overview of the features of some relevant studies, together with a characterization along the following axes:

- task to be solved by the robot
- ANN model and architecture
- EA used for the optimization
- presence of another adaptation time-scale (e.g., learning)

Finally, we discuss the case where NE has been used for evolving one or more controllers that are shared by many robots that interact, more or less tightly, to achieve a common goal.

23.2.1.1 Tasks Considered

For allowing NE to truly shine, the tasks which have been mostly taken into consideration in the examined literature are those which require advanced controllers, either because of the task difficulty itself or because of the complexity of the robot involved. Among the first ones, we can mention *navigation* [17, 18, 47, 59, 159, 160, 163] (see Fig. 23.1), predator-prey tasks [30], or *locomotion* tasks in complex environments [155], for which the sensor-actuator mapping becomes non-trivial, even considering a simple agent with a restricted set of actions.

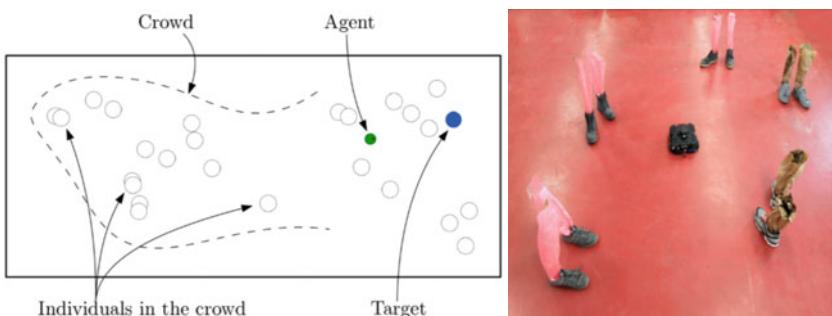


Fig. 23.1 A schematic view of a navigation task for a differential drive wheeled robot (left) and its realization with a few human mock-ups (right); both images are taken from [163]. The task has been solved by [163] with NEAT

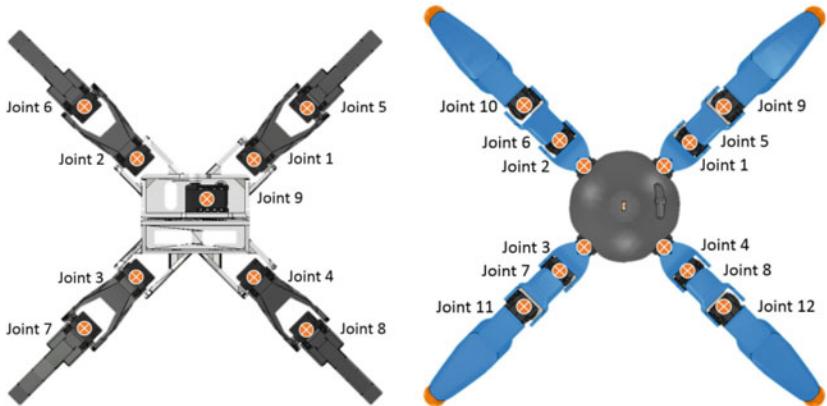


Fig. 23.2 Top view of two legged robots, each with four legs, for which the gait has been optimized by [150] using a few variants of Hyper-NEAT. Image taken from [150]

Among the latter ones, locomotion is usually the go-to task, as for legged robots (see Fig. 23.2), for instance, it already requires the control of several robotic parts [2, 54, 150]. In the case of legged robots, the task of obtaining a policy governing the movements of the legs is often referred to as *gait generation*.

Neuroevolution has been employed also for solving tasks related to industrial robots, among which the task of manipulating an object using a robotic arm with a suitable effector is a prominent example [68, 179] (see Fig. 23.3).

Last, when dealing with modular robots or swarm robotics, self-assembly [85] or reconfiguration [194] tasks are also noteworthy test beds for evolutionary optimization: the use of ANNs for solving these tasks, and hence of NE for optimizing them, is still an open front.

23.2.1.2 ANN Model and Architecture

Focusing more on the NE aspect, the first features we concentrate on are the neuron model and the neural architecture employed. The McCulloch and Pitts neuron model (*perceptron*) is one of the most commonly used [106] for computing neurons outputs, thanks to its simplicity and computational efficiency [141, 174]. However, quite a few works have considered more biologically plausible models of neurons, such as the bio-inspired spiking neuron model [72, 176] (see Fig. 23.4).

Concerning the architecture, again most of the works aim at simplicity and computational efficiency, choosing a fully-connected or a sparse feed-forward ANN. The latter ones, in particular, can either result from a process of NAS (as we will see in the following paragraph) or can be obtained by pruning [119, 122], that is the removal of some neural structures (e.g., synapses) during the lifetime of the agent.

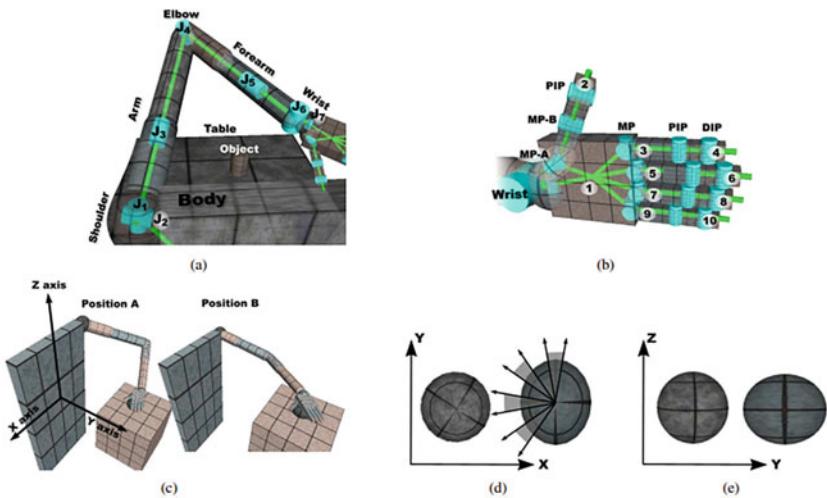


Fig. 23.3 An overview of the task tackled with neuroevolution by [179]: the goal is to make the anthropomorphic robotic arm (equipped with a human-like hand as effector) able to discriminate different kinds of objects using perception. The robot is controlled with an ANN with a single hidden layer of few nodes: the input layer is fed with readings coming from arm and hand proprio-sensors and tactile sensors; the output layer provides the arm and hand actuators values and the category of the object being manipulated as estimated by the robot. Image taken from [179]

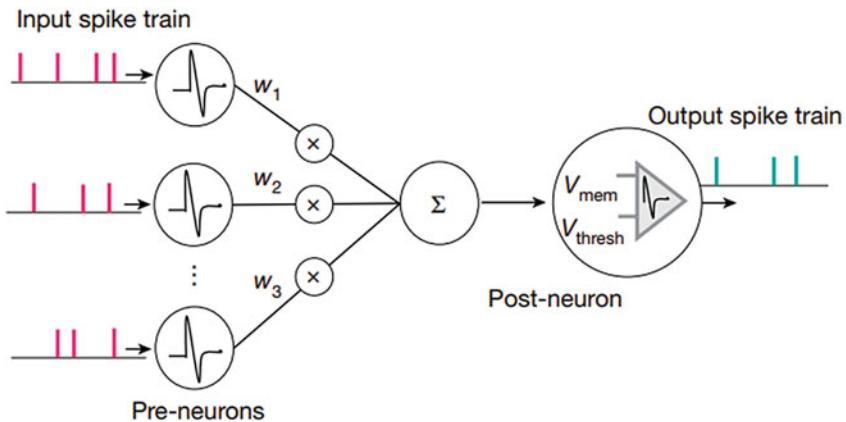


Fig. 23.4 A schematic view of a neuron model based on spikes, instead of continuous values; image taken from [154]. [121] optimized a controller for modular soft robots based on this model

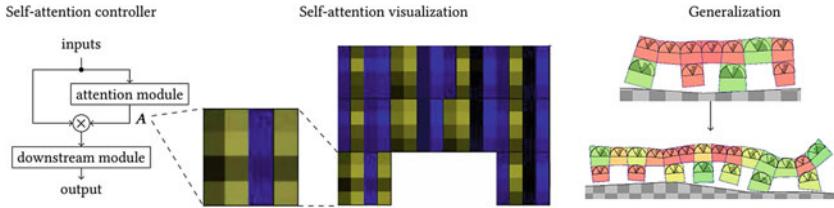


Fig. 23.5 A neural controller incorporating an attention module, optimized by [143] for controlling a modular soft robot; image taken from [143]. Self-attention is a mechanism that allows the ANN to enhance some of the inputs with respect to other ones, resulting, in practice, in a form of auto-adaptation. Thanks to the attention module, the authors were able to make modules (colored squares in the images on the right) not needing any form of communication among them, still obtaining a collective behavior highly effective for the considered task (locomotion): specifically, the ANN in each module becomes more attentive to sensor readings that are more useful locally, hence performing a sort of specialization

Some more sophisticated architectures of ANNs have also been used, namely, Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM), in order to endow the controller with a form of memory, which is particularly useful for the accomplishment of tasks that benefit from tracking previous perceptions and/or actions [1, 196] such as navigation. In addition, recent advances in the field of deep learning (DL) have ignited the experimentation with deeper and more complex ANNs as robotic controllers, involving DL inspired elements such as self-attention modules [143, 175] (see Fig. 23.5).

Last, a slightly separate role is played by Central Pattern Generators (CPGs), which are biological neural circuits that produce rhythmic outputs in the absence of rhythmic input [15], that have been successfully employed in those tasks (e.g., locomotion) where a periodic behavior is useful for achieving stability and high performance [78, 87, 102, 177] (see Fig. 23.6).

23.2.1.3 Evolutionary Algorithm

Another axis of categorization regards the evolutionary aspect, i.e., what is being evolved and what EA is being used to this extent. As seen in the previous paragraph, many works have relied on fixed neural architectures, be them feed-forward, recurrent, or more refined. In those cases, the most common approach to training involves evolving the parameters, i.e., the synaptic weights, of the ANN with EAs that are suitable for fixed-length numeric genotypes, such as evolutionary strategies (ES). Not only have ES been proved to achieve state-of-the-art performance on a wide set of tasks [155], but, as opposed to gradient-based optimization methods (e.g., backpropagation) they need not enforce any particular constraint on the outputs of neurons. As such, ES have been profitably used to overcome training issues like the

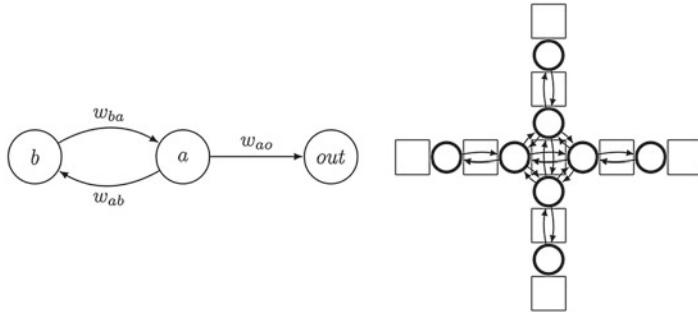


Fig. 23.6 The main component of a CPG (a differential oscillator, on left) and its usage within a specific morphology of a modular robot; both images are taken from [78]. In brief, a CPG constitutes a dynamical system whose evolution over time (i.e., the dynamics) is determined by a few numerical parameters. [78] used CPGs and CPPNs (see Sect. 23.3.1.2) for evolving both the morphology and the controller of simulated modular robots for the task of locomotion. Specifically, they used a compound of CPGs as an open-loop controller, i.e., one in which the CPGs are not fed with sensory feedback, since gait learning on a flat surface can be solved without such feedback. Moreover, the authors employed a Lamarckian approach, i.e., one in which some traits are developed after birth and are then inherited by the offspring

non-differentiability in Spiking Neural Networks (SNNs) [31, 33, 47, 59, 121, 147, 159, 160]. In addition, within the domain of fixed architectures, quality-diversity (QD) approaches have also been explored, in order to avoid getting stuck in local optima [27, 46] (see Fig. 23.7).

On the other hand, many studies have encompassed non-fixed neural architectures, relying on the evolutionary process for obtaining the most suitable ANN structure for the task at hand. In fact, as recently shown by [50], often the architecture of the ANN plays such an important role, that even random weights could be used, as long as the architecture stays untouched. Along this line, many have resorted to EC applied to NAS, mainly applying the NeuroEvolution of Augmenting Topologies (NEAT) algorithm [171, 172] for obtaining well performing and robust robot controllers [17, 18, 163, 188].

A step further has been taken with Hyper-NEAT [170], a generative encoding which evolves large scale ANNs with the principles of NEAT. This approach has led to outstanding results for the control of robots with high symmetry, such as legged ones [2, 25, 30, 58, 150], since Hyper-NEAT can automatically identify and effectively exploit problem regularities (see Fig. 23.8), yielding to emergent controller modularity and high coordination. As a side note, some older works have also focused on the importance of ANN modularity for control tasks [19], yet handcrafted solutions which tried to take advantage of symmetry and repetitions [54, 181] were in general less fortunate than those obtained through a generative encoding.

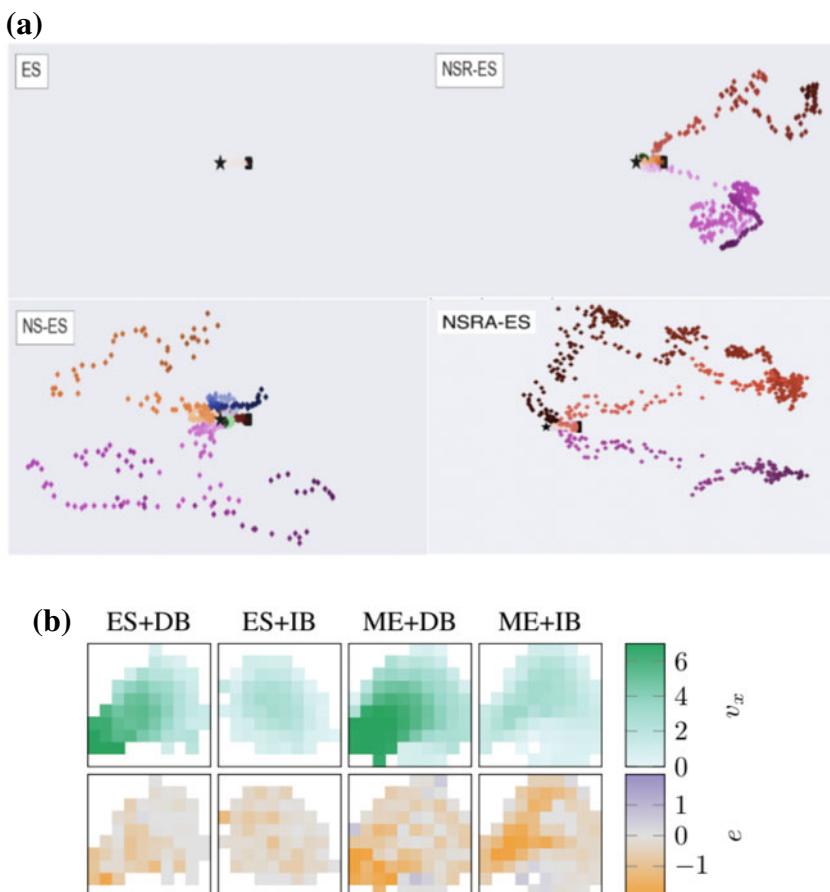


Fig. 23.7 Two examples of usage of QD EAs for optimizing the ANN-based controller of robotic agents. **a** Four variants of ES, three of which employing a form of QD; image from [27]. [27] used ES for optimizing the neural network controlling a simulated humanoid for the task of locomotion. The plots show the behaviors of different agents in terms of position (top view); QD is clearly beneficial, as it allows the robot to overcome an obstacle being placed in front of it at its initial position (the star). **b** Overview of the outcomes of several optimizations performed with ES or Map-Elites (ME, a form of QD evolution [118]) in terms of fitness (above) and evolvability (below), in the case of controller optimization for modular soft robots; image taken from [46]. *x* and *y* axes of each plot represent two behavioral descriptors. [46] compared different representations and EAs in terms of their ability to favor evolvability and exploring the space of solutions

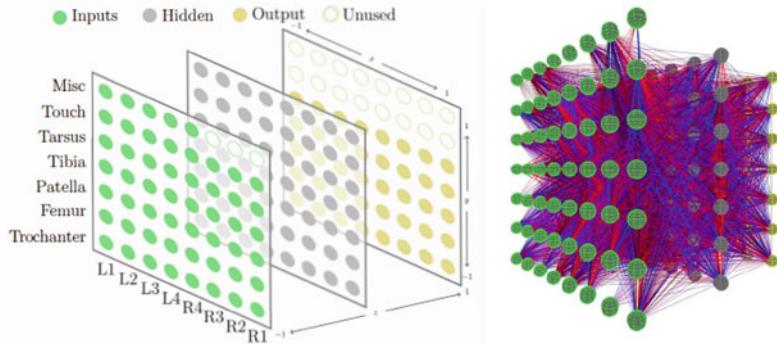


Fig. 23.8 Schematic view of the usage of Hyper-NEAT for optimizing the controller of a robotic-like eight-legged character; on the right, the mapping between inputs and output that exploits the physical structure of the robot; on the left, a representation of the best ANN obtained for the task of locomotion. Both images are taken from [2]

23.2.1.4 Adaptation Time Scale

Another significant aspect to take into consideration when analyzing literature on ANNs for robot control regards the adaptation aspect. More precisely, most works focus on evolutionary adaptation, namely relying on the time-scale of different generations for improving the ANNs of the agents. However, some researchers have experimented with a shorter adaptation time-scale, that is life time learning, with the goal of fostering generalization to unforeseen circumstances, as for biological creatures [48, 130, 131]. In this context, the most fruitful results have been achieved with unsupervised Darwinian learning, i.e., learned traits are not transferred to the offspring, in the form of neural plasticity, i.e., Hebbian learning. These ideas have been ignited by [123] which have proposed to evolve the synapse-specific Hebbian learning rules instead of the synaptic weights, and have been productively applied for controlling different types of robots to enhance their resilience to body alterations and/or environmental changes [44, 138].

23.2.1.5 Modular Robots and Swarm Robotics

The last view-point considers the amount of “independent” modules or robots involved. In the first case, we consider modular robots, for which the modules may be able to detach or at least to control themselves independently of the others (see Fig. 23.9), whereas for the latter case we fall into the category of swarm robotics. In this context, we can put forward the concepts of monomorphic and polymorphic systems, where the first indicates modular agents or swarms were all components are alike, while the second refers to heterogeneous compounds. Some works have found monomorphism to be preferable [136, 142] for both modular robots with

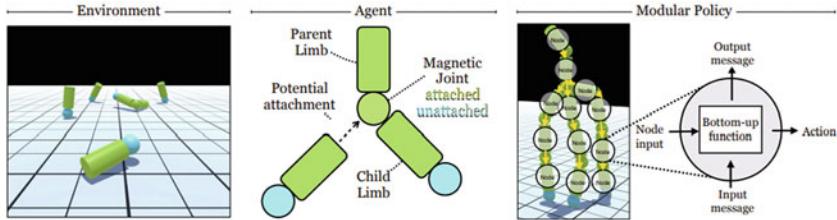


Fig. 23.9 An overview of an evolutionary approach for optimizing the controller of simple robotic modules that have the capacity of joining and hence forming complex morphologies; image taken from [136]. Single modules are governed by ANNs that determine the actions to be performed (i.e., how to control the actuator) and the messages to be sent to neighbor modules. Despite the approach is described by [136] as a form of co-evolution of morphology and controller, the optimization is actually a form of RL

truly embodied controllers [108] and robot swarms [14, 85]. Intuitively, optimizing monomorphic components (i.e., modules or robots) is easier because the search space is smaller; moreover, the interaction between several independent components is facilitated when they are similar to each other. In addition, it has been observed that despite the independence of the agents, coordination can still emerge [14, 58, 121], even in the absence of explicit inter-agent communication [143].

23.2.2 Other Combinations of ML and EC for Controller Optimization

Besides ANNs, several other approaches to robot controller optimization have involved the application of EC. In this section, we try to cover some relevant works within this broad area.

23.2.2.1 Combining Evolution with Reinforcement Learning

Reinforcement Learning (RL) is the field of ML in which an agent learns the desired behavior through a trial-and-error process of interaction with an environment [173]. Due to its formulation, RL is naturally suited for the robotics context, in which it is applied to learn a controller, called *policy* in the RL context, which maximizes a certain reward signal during a given time span, e.g., during the life time of the agent. The core of RL is hence the agent policy, which can be represented in several manners, and needs to be optimized through a policy search process [164] that can be conducted in various ways, depending mostly on the representation chosen and on the related search space.

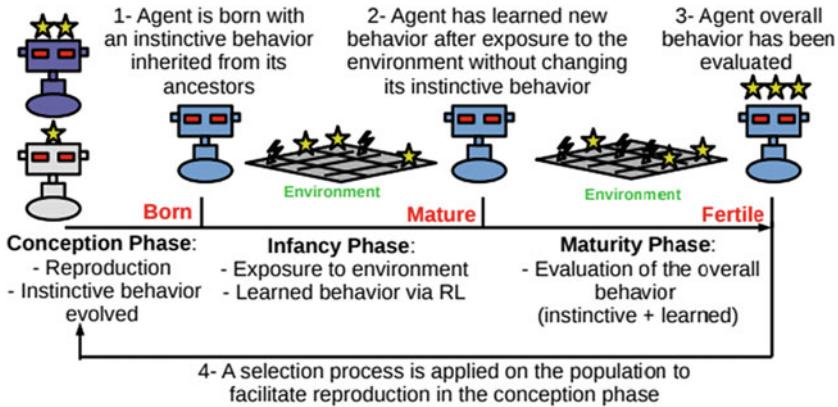


Fig. 23.10 The scheme principled by [60] for combining RL and evolutionary computation for ER; image taken from [60]. During the evolution, robotic agents are initially developed with an instinctive behavior dictated by a policy inherited from parents; their policy can change during their life based on their interactions with the environment, according to an RL approach; the selection phase of the evolutionary process takes finally into account the fitness resulting from the learned behavior, rather than just the instinctive one. The authors assessed their approach on a few (simulated) robotic tasks and found promising results

For instance, it is possible to rely only on EC for policy search with different EAs according to the specific representation chosen [27, 145, 155, 171, 190]. In this context, the dependency between the policy and the effectiveness of the corresponding behavior is not explicitly modeled; instead of trying to estimate the model, the search process only aims at moving solutions toward points in the search space which maximize the final objective, i.e., yield to a higher cumulative reward. Even though EC for policy search suffers from lower sample efficiency as compared to classical RL algorithms, there have been some attempts to tackle this issue [89, 145]. In addition, it has been shown how EC can be a scalable alternative to classical RL [155], achieving state-of-the-art performance on a benchmark of control tasks.

Other examples in which EC has been applied to RL aim at leveraging the advantages of the two approaches. In particular, two mainstream ideas suggest to make use of EC to improve the outcome of RL techniques or to allow EC and RL to act as two concurrent forms of adaptation occurring along different time-scales, as in [60] (see Fig. 23.10). Concerning the former idea, EC has been used for reward shaping, to discover a reward signal that allows the agent to efficiently learn [128].

Some have tried to achieve explainable policies, which are of primary importance in critical settings, relying on tree-based representations and genetic programming (GP) in combination with RL [29, 64, 193]. Last, EC and RL have often been combined, in and out of the robotic context for algorithm acceleration [38, 103, 104, 116, 137], using EC as bootstrapping for RL, or for hyper-parameters tuning [73].

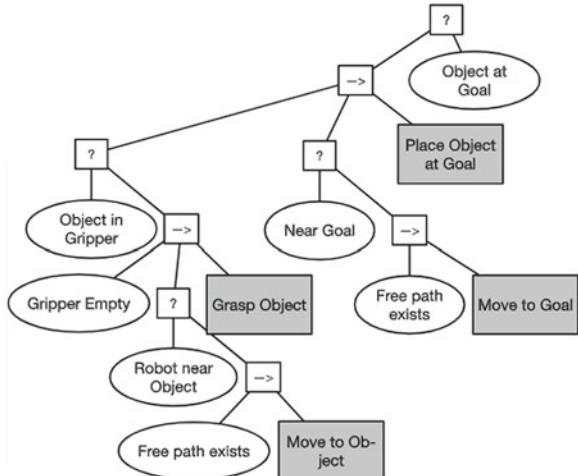
23.2.2.2 Behavior Trees

Behavior Trees (BTs) describe the controller of an agent by the means of a tree, which is meant to be traversed until reaching a terminal state (see Fig. 23.11). The leaves in BTs correspond to executable behaviors, whereas internal nodes represent control flows. This representation was first invented to enable modularity in artificial intelligence for computer games, but has recently received an increasing amount of attention in the robotics community [70]. The main advantage of BTs lies in their interpretability, which comes in particularly handy when the considered task is critical or there is the need to adjust the learned behavior by hand to address reality gap issues [161] (see Sect. 23.5). Given their tree-based structure, GP is naturally suited for optimizing BTs [71], and it can also be boosted with grammatical evolution (GE) to enhance the final performance [61, 79]. BTs have also been profitably applied in various multi-agent contexts, to ease the understanding and prediction of the emergent swarm behavior [81, 125, 126].

23.2.2.3 Evolution in Control Systems

Despite not belonging fully to the ML context, several classical control theory approaches have benefited from EC techniques in robotics, so we briefly report some relevant works in the field here. For instance, EC has been used in combination with forward kinematics equations for trajectory planning in robotics [186], in order to reduce the impact of noisy environments [68]. In addition, GP has often been exploited to address the problem of inverse kinematics in manipulators (see Fig. 23.12), aiming at obtaining precise yet interpretable results [20, 35, 86, 149].

Fig. 23.11 An example of a BT for a mobile manipulator, i.e., a robot composed of a manipulator mounted on a wheeled platform; image taken from [70]



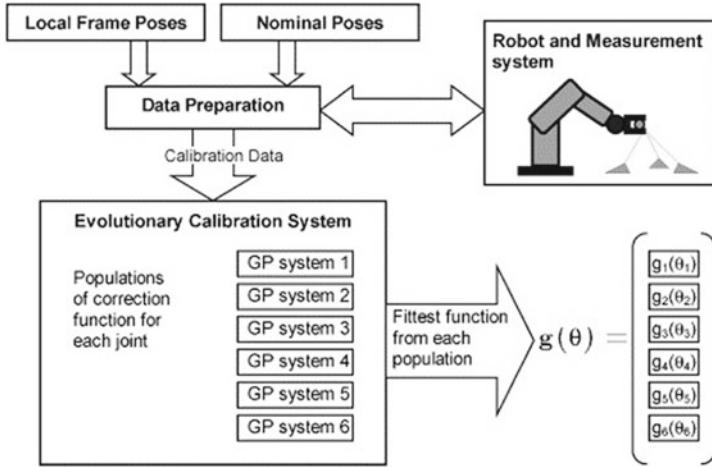


Fig. 23.12 An overview of the approach of [35] for performing the inverse static kinematic calibration of industrial robots using GP; image taken from [35]. The authors used the capability of GP of optimizing both the structure and parameters of a regression model for building concurrently six models for the six joints of the robot

Moreover, EC has been often applied in combination with control algorithms for parameter fine-tuning in robots like Unmanned Aerial Vehicles (UAVs) [10] or walkers [66].

Last, we quickly mention some applications of EC on fuzzy control systems [42, 127, 146], that are those control systems based on fuzzy logic. In this context, EC has mostly been used for tuning the parameters of the fuzzy controller, in several robotic scenarios. In particular, autonomous vehicles have been a widely used test-bed for deploying fuzzy controllers with feedback [55]. Various works have encompassed wheeled robots in navigation tasks [24, 82], and have used EC to achieve more interpretable results [83, 84].

23.3 EML for Body and Body-and-Brain Optimization

The *embodied cognition* paradigm [140] posits that the intelligence of an agent (natural or artificial it may be) emerges from the complex interactions between the body, the brain, and the environment. Intelligence is not only rational, but is also embedded in the body. The paradigm stands in contrast with the traditional view inside the computer science community of intelligence being abstract, purely Platonic “reason” [191]. Indeed, bodies co-evolved with brains and shaped them, inasmuch as the body defines the actions that the brain can “afford” [51]. As a matter of fact, there exist organisms capable of performing complex computations by the means of their bodies only. Individuals of the genus *Planaria* regrow amputated limbs by



Fig. 23.13 One of the *Strandbeests* realized by the dutch artist Theo Jansen, named by the creator Animaris Percipiere; image taken from [77]

virtue of biological processes localized in the severed portion of their bodies [98]. Individuals of the species *Trichoplax adhaerens* self-organize their cilia to locomote in the absence of any nervous system [16]. As a notable instance in the artificial domain, passive walkers locomote by the means of body dynamics only [107] and are already a mature field of study within robotics [26], to the point that artist Theo Jansen applied EC to design the *Strandbeests* (see Fig. 23.13), passive walkers that rely on wind energy only, to raise environmental awareness [77].

To no surprise, researchers delved into the issue of agent body optimization; such topic is relevant for EML as it is deals with problems of RL, itself a part of ML. With respect to controller optimization, the body places an additional challenge: most agent bodies are not differentiable and thus cannot be optimized by the means of gradient-based optimization methods. [57] explored policy search to jointly optimize the control and body parameters of simulated bipedal walkers, but restricted the search to different configurations of the same morphology (e.g., length of the pedals). To fully unleash its potential, body optimization must be capable of open-ended complexity. EAs fit this goal, since they can evolve any artifact given an appropriate representation and a fitness function.

Starting from the seminal work of [166]—the “father” of ER—who first unveiled the power of evolution for optimizing the bodies of virtual creatures, researchers have increasingly relied on EC also for body optimization and body-brain optimization. Both of them pose unique challenges when compared to controller optimization, as we shall see in the following.

23.3.1 EML for Body Optimization

As representing a body is non-trivial, body optimization propelled researchers to put effort into conceiving representations that are (a) scalable, in terms of the solution complexity, and (b) effective, in terms of the quality of the solutions. The literature usually groups body representations into two categories: *direct* and *indirect* (sometimes also called *generative*). Direct representations provide a one-to-one mapping between genotype and solution, whereas indirect representations, on the other hand, evolve a data structure that we can map to a solution in a non-trivial procedure. In a certain sense, indirect representations evolve the blueprint that can generate a solution by decoding the instructions contained in the genotype (hence the name “*generative*”).

23.3.1.1 Direct Body Representations

Direct representations provide a one-to-one mapping between genotype and solution. They are, in general, easy to craft, but potentially less scalable than indirect ones, as the complexity of the genotype directly depends on the complexity of the solution. Examples include graphs and numeric vectors.

Numeric vectors are the most direct of the representations. In that case, every gene (i.e., scalar) encodes a specific trait of the solution. For example, [142] evolved the morphologies of 2D voxel-based soft robots encoding each voxel (given a maximum enclosing grid) with a real number in a vector (see Fig. 23.14). Albeit simple, this representation proved effective to evolve a variety of shapes and behaviors, also using a speciation mechanism [172].

Sims [166] first evolved virtual creatures by encoding their morphology as a directed graph, with nodes being the rigid parts and the edges being the connections among those. The resulting creatures not only evolved life-like patterns for locomo-

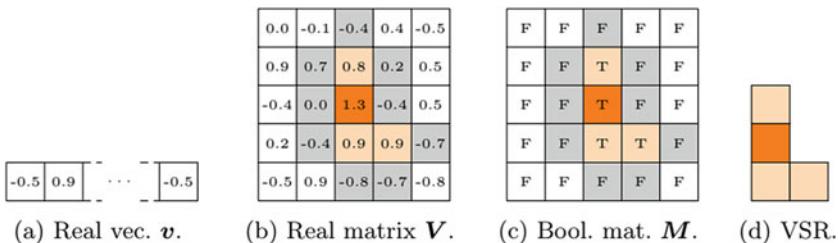


Fig. 23.14 A schematic view of the grid-based, fixed-length direct representation for evolving the body of 2D voxel-based soft robots used by [120]; image taken from [120]. A fixed-length real vector is first reshaped in a matrix; then, starting from the cell with the largest values, subsequent voxels are added in correspondence with cell with next largest values that are adjacent to already visited cells. Similar representations have been used, e.g., in [46, 142, 143]

tion, but also for competitive co-evolution, where individuals belonging to different populations (forming species) evolve to win a one-on-one fight for the control of a resource (a cube in the arena where the simulated creatures fight) [165]. As testified by adoption of open-source platforms, graphs (including trees [69]) have proved an intuitive representation for modular robots [3, 195].

23.3.1.2 Indirect Body Representations

Indirect representations evolve a data structure that we can map to a solution in a non-trivial procedure.

Indirect representations attracted the most interest in the community. In fact, they bear similarities to what happens in biological organisms, whose complexity and diversity are the result of an astonishing data compression feat, the “genomic bottleneck” [151]: just a handful of genes in the DNA encode the instructions to build a complex organism (humans, with 30 trillion cells and all their cerebral complexity, just have 30,000 genes in their genome [189]). As a result, indirect representations might provide a bridge toward more complex robotic systems.

Examples of indirect representations include

- Compositional Pattern Producing Networks (CPPNs);
- sequences mapped to strings belonging to languages defined by grammars;
- Gene Regulatory Networks (GRNs).

Indirect representations allow great freedom to the designer and are suitable to evolve properties we know are beneficial a priori. As a matter of fact, CPPNs [169]—univariate function networks resulting in multivariate functions expressing repetitions and symmetries—evolved to express multi-material 3-D morphologies with life-like patterns, e.g., symmetry [4] (see Fig. 23.15). On the same research line evolved CPPNs expressed effective morphologies for 3-D voxel-based soft robots for squeezing through tight spaces [21], underwater locomotion [28], and recovering from damages [96]. Most notably, evolved CPPNs expressed—*in silico*—the morphologies for organisms to be assembled—*in vivo*—from cells of *Xenopus laevis* (a frog); such organisms (the *xenobots*) can locomote by themselves [95] as well as self-replicate [93] and are, to date, one of the most impressive simulation-to-reality feats within the robotics EML community.

Similarly to CPPNs, grammars allow the designer to bias the search space, but in a different way: grammars specify the building blocks of a solution and what relationships are admissible among those blocks. There exists a rich literature on the evolution of solutions defined in languages described by grammars, starting from Grammatical Evolution (GE), a form of GP [92], and its more recent variants, as Probabilistic Structured GE [109] or Weighted Hierarchical GE [7]. Lindenmayer systems (L-systems) [99]—first developed to model the biological development of multi-cellular organisms—consist of an alphabet of symbols to be replaced with replacement rules (see Fig. 23.16): symbols can be robot modules and instructions

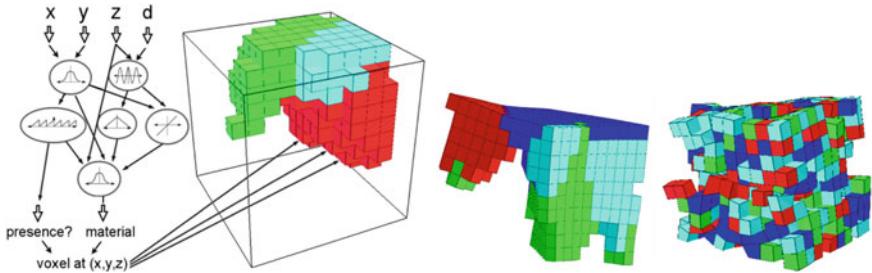
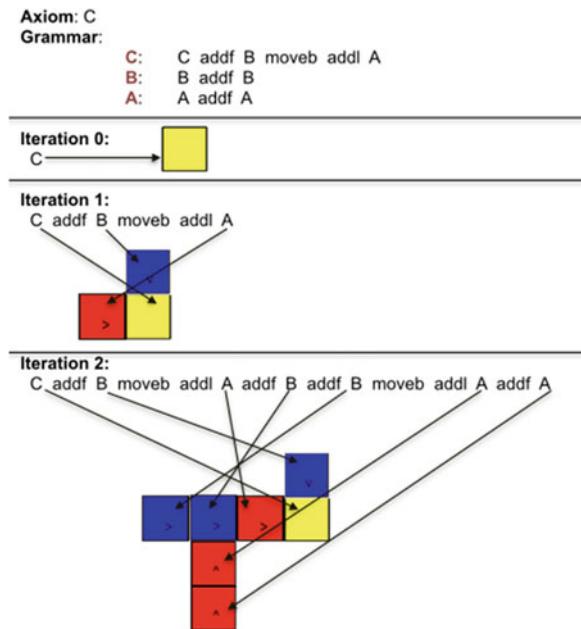


Fig. 23.15 Schematic view (on the left) of an indirect representation based on CPPNs for optimizing the body of 3D voxel-based soft robots; in the middle, an example of an evolved body that is effective in locomotion and exhibits life-like patterns as repetitions and symmetries; on the right, a robot evolved with a direct representation (conceptually similar to the one of Fig. 23.14)—all the three images are taken from [23]. [23] used NEAT as the EA driving the evolution of the CPPNs describing the robot bodies. The controller here simply consists in different phases of the expansion-contraction cycle of the voxel composing the body, here visually encoded with different colors

Fig. 23.16 A schematic overview of an indirect body representation for modular robots based on L-systems; image taken from [115]. There is a symbol in the grammar for each available module and a few symbols for operating on the current expression, by replacing symbols, and for changing the reference point. [115] used this representation for evolving (simulated) modular robots composed of three possible modules: a controller, a joint actuated by a servo-motor, and a passive structural module



on how to assemble them [65]. As a result, L-systems have succeeded at evolving modular robots [113, 114] on the Revolve framework [69].

When evolving embodied agents, the effect of mutations is non-trivial and hampers evolvability [41]. To this end, [13] conceived GRNs as linear genotypes that, through differential gene expression and the diffusion of gene products, transform a single structural unit into a complete robot morphology via a development process:

mutations expressed earlier in development tend to have a more variable effect than mutations expressed later, lending to the evolvability of the representation.

23.3.1.3 Comparisons Among Representations

Considering how important the choice of the representation is, some works did consider comparing different representations; most of them considered the scenario of modular robots, because they allow for great expressive power in describing the bodies, due to their inherent nature of systems composed of many components (see Fig. 23.17).

Extending the work of [4], [23] showed that evolution of CPPNs is more effective than a direct representation at expressing morphologies for multi-material voxel-based agents, as it produces regular patterns as seen in nature (see Fig. 23.15). Also using modular robots, [185] argued that indirect representations are more suitable for evolving small-sized robots, a conclusion that we find counter-intuitive. [32] found an indirect L-system representation to underperform a tree-based direct one in terms of locality of the representation [152], since many genes needed to align to create a positive effect on the solution (epistatic effects).

In the context of voxel-based soft robots, both [120] and [46] compared different representations (a direct and indirect ones) for the body of the robots. In the former study, the authors considered the case of morphological development of robots, i.e., the addition of new voxels to the body during the life of the individual, as dictated by the genotype according to the chosen representation. In [46] (see Fig. 23.17c), the aim was to compare the representations in terms of their ability to favor the evolvability, i.e., the possibility to produce fit offspring, of found solutions.

Despite the efforts devoted by the authors of the studies surveyed above, a comprehensive comparison among representations is arduous, given the abundance of them; even indirect ones can differ wildly in terms of properties of the representation [153]. Moreover, the works mentioned so far consider different robot types and frameworks. Thus, the community still misses a full understanding of the trade-offs among representations.

23.3.2 EML for Body-and-Brain Optimization

Body-and-brain optimization has historically been difficult [80], due to the deep entanglement between the brain and the body. The seminal work of [100] suggested the reason to be the ruggedness of the fitness landscape: they proved that evolving the morphology of voxel-based agents for a fixed controller results in more premature convergence to local minima than evolving the controller for a fixed morphology. Controllers need to adjust to their morphology [39], as the body modulates the communication (motor and sensory) between the brain and the environment.

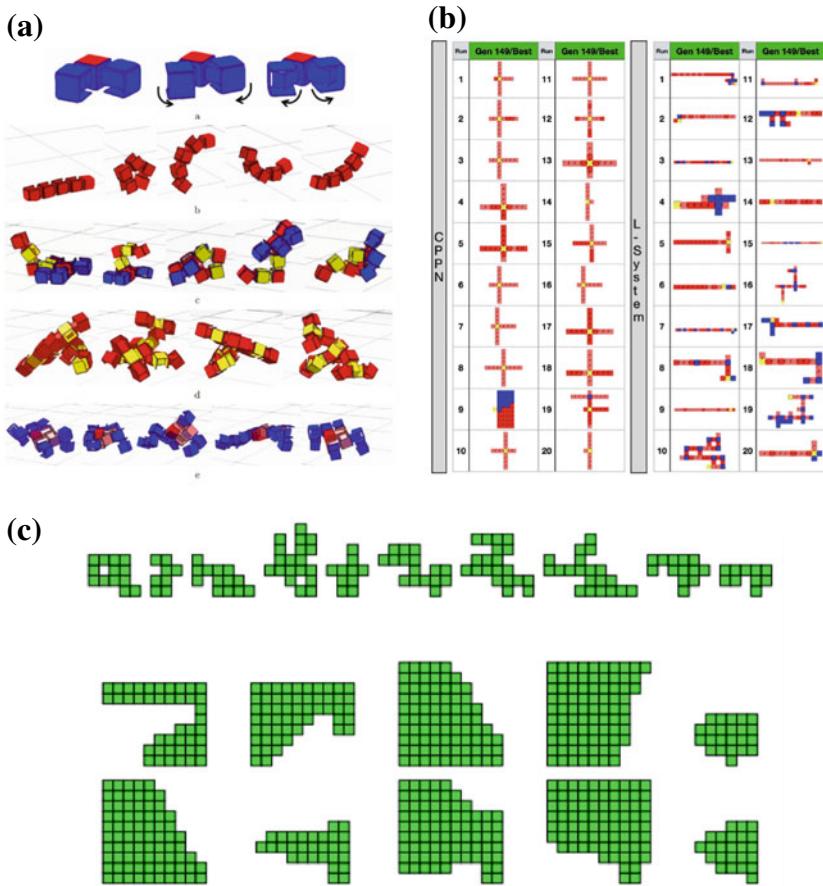


Fig. 23.17 A visual summary of the outcome of three studies comparing different representations for the body of modular robots. In all the cases, the figure shows a few individuals obtained by the means of evolutionary optimization using one or two representations. **a** Veenstra et al. [185] considered modular rigid robots composed of up to 20 modules and found that, in particular at the initial stage of the evolution, a generative representation is better than a direct representation. Modules are of two kinds: a fixed cube and a cube-like one with a face that can be displaced with respect to the others by actuating a rotational joint. The generative representation is based on L-systems. Image taken from [185]. **b** Miras [111] compared a representation based on CPPN and one based on L-systems for evolving the body of modular rigid robots composed of four kinds of modules. The authors considered not only the impact on the effectiveness, i.e., the ability of the robot to move, but analyzed how the two representation bias and constrain the search space. They found that CPPN gives slower but more stable gaits. Image taken from [111]. **c** Ferigo et al. [46] compared two representations, a grid-based direct one and an indirect based on Gaussian-mixtures, and two EAs in terms of their ability to foster the evolvability of found solutions, in the context of 2D voxel-based soft robots. The authors found that the evolvability is mostly influenced by the EA, while the representation largely affects the fitness. They also found an evident bias toward larger robots with the indirect representation (bottom in this figure). Image taken from [46]

Researchers have since then tackled body-and-brain optimization either by *joining* the body and the brain in the same optimization or *decoupling* the two (e.g., with two nested optimization loops). Both approaches have benefits and costs: the former simplifies the optimization, while the latter explicitly takes into account how brains need to adapt to their bodies.

23.3.2.1 Joining Body-and-Brain Optimization

Joining body-and-brain optimization allows to solve two (entangled) optimization problems as one single optimization problem.

When the representation is direct (as in Sect. 23.3.1.1) that amounts to joining the two representations (for brain and body) into one single representation [142]. For instance, [133] successfully evolved real four-legged robots (see Fig. 23.18) using one single numerical genotype for both morphological (leg lengths) and control (gait) parameters and showed adaptation as physical conditions changed. Going back, [166] evolved the neural network controllers of his virtual creatures as graphs embedded inside the nodes of the morphology graph (see Sect. 23.3.1.1), with mutations applying to both types of graphs.

Indirect representations (as in Sect. 23.3.1.2), on the other side, can potentially encode body and brain with the same data structure. [22] evolved voxel-based agents with two CPPNs, one to express the morphology and one to express the distributed open-loop controller, and reducing selection pressure on individuals with recent morphological mutations: evolution was free to mutate one CPPN or the other, while

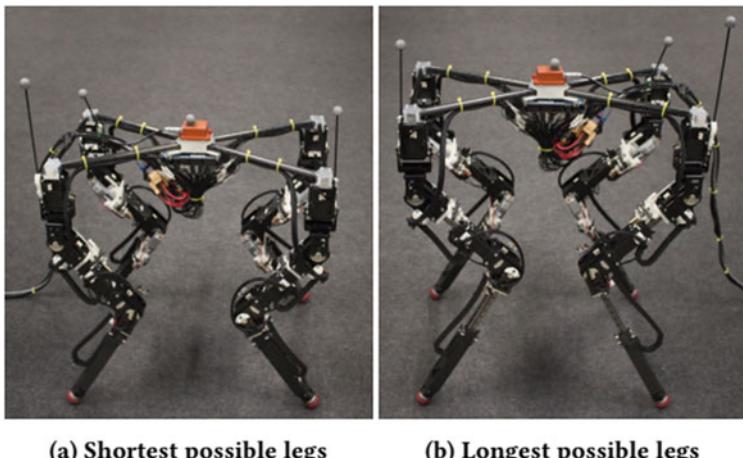


Fig. 23.18 The four-legged robot used by [133] for the joint evolution of morphology and control; image taken from [133]. The robot can (slowly) change the length of the legs: since the time-scale of this variation is much longer than that of the gait, the change is made offline, i.e., between subsequent fitness evaluations, and is hence considered a form of morphological evolution

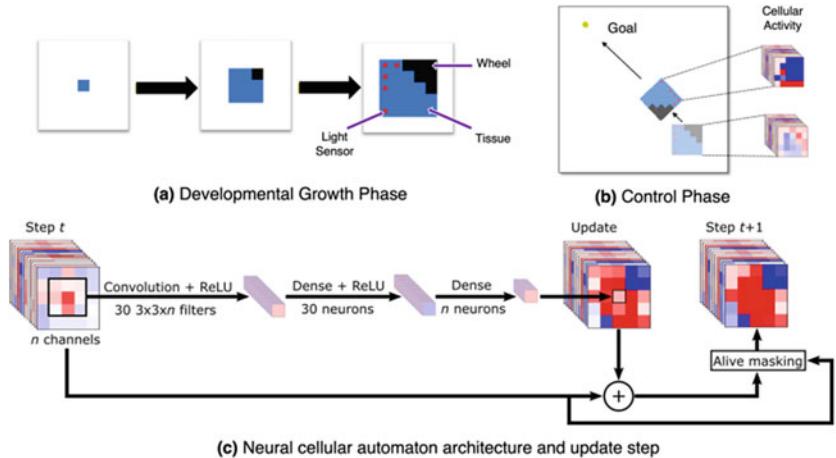


Fig. 23.19 A schematic overview of the approach proposed by [144] for the concurrent evolution of the body and the brain of a simulated robotic mobile agent composed of a few modules with different roles. The key contribution of this work is in the neural cellular automaton used for determining both the way the body develops, in an initial instantaneous stage at the beginning of the agent life, and how it behaves during its life: in the latter stage, actuatable modules (i.e., those equipped with wheels) wheels are controlled based on the state of the automaton. Image taken from [144]

having time to “readapt” to new morphologies. Their approach showed potential to avoid local optima. Finally, [144] evolved one single neural cellular automaton for developing the body and functioning as the brain for multi-material car racers and is a promising line of research (see Fig. 23.19).

23.3.2.2 Decoupling Body-and-Brain Optimization

Decoupling approaches usually cast the body-and-brain optimization problem as a nested optimization problem. An outer evolutionary loop searches in the space of bodies, while an inner optimization loop searches—for each body—in the space of brains; intuitively, that is how nature shaped animal life on Earth. Approaches then differ according to the algorithm employed at the inner loop. Interestingly, both evolution and learning (the slowest and the fastest time-scales of adaptation, respectively [167]) appear.

Some works adopted RL to learn the brains, as RL is a sample-efficient optimization over the lifetime of a robot. Most notably, [56] evolved bodies with a genetic algorithm and learned their brains with RL to master a wide gamut of tasks for tree-based simulated robots, verifying that such environmental complexity fosters the ability of a body to facilitate learning of novel tasks. Additionally, bodies that are more physically stable and energy efficient facilitate learning the most.

On the other side, the inner loop need not consist of learning, but may be evolutionary just like the outer loop. In particular, every body comes with a population of brains that evolve and whose best individual determines the fitness to the body-brain pair. Using this approach, [78] compared a Darwinian and Lamarckian approach for the inner loop and found that the latter considerably reduces the time to learn a good solution. Similarly, [97] showed that initiating learning from a brain inherited from an archive of learned brains, rather than from a randomly initialized one, both the speed and magnitude of learning increased over time. Subsequently, [112] found that indeed the inner evolutionary loop produces robots that perform better on a given task and [102] quantified the *learning delta*—the performance difference between inherited and “learned” brains—in terms of morphological descriptors.

23.4 Other Combined Usages of EC and ML in Robotics

There are many other cases in which EC has been successfully used for optimizing other components of scenarios involving robots than “just” body and brain. These cases include other parts of the robots, such as the sensory apparatus of voxel-based soft robots [43, 45] or the object recognition part of robot grippers [67], the way robot develops during their life [94, 120, 124], as well as the task itself [132, 187], or even the simulator used for optimizing, in simulation, a real robot [11]. While all these approaches can be encompassed in the field of ER, they hardly fit the definition of EML, since they do not directly and explicitly employ ML.

In a few other cases, ML and EC “met” in contexts where they both concurred in determining a solution for a broader problem. For example, [142] studied what are the key evolutionary factors (namely, the employed EA) affecting the diversity of evolved voxel-based soft robots: the authors interpreted the notion of diversity taking the inspiration from biology, where a human observer categorizes individuals within a predefined structure of categories (species). In the cited work, the authors used a supervised ML pipeline to replace the human observer with a model learned from a few data points labeled by an actual human observer. The model is hence used for measuring the diversity of a large number of evolved populations of robots that, for the scale, could not have been manually inspected by an actual human. The authors found that those EAs that are designed to promote diversity are indeed able to achieve this goal, in particular when behavioral traits are used for telling apart individuals. However, they also found that external conditions (i.e., the environment) may have an even greater impact on observed diversity: e.g., robots evolved for doing locomotion on a downhill terrain very often ended up having a rounded shape which facilitated rolling.

A similar approach, i.e., one in which ML powers a component of a larger system in which at least another part is based on EC, has been adopted by [91]. For the aim of fighting the reality gap (see Sect. 23.5), the authors trained a ML model for estimating the transferability of robot controllers evolved in simulation, i.e., the degree to which their performance, as seen in simulation, do not change when moved in reality.

The evolution itself was bi-objective, with one objective being the transferability and the other being the effectiveness. According to the authors, evolving for transferability and effectiveness together is practical way for fighting the reality gap problem.

23.5 The Reality Gap Problem

As already mentioned in the previous sections of the chapter, in ER an EA operates on a population of robots plunged into an environment. The ultimate goal is that of designing a robot (its brain, body, or both) which maximizes all the desired performance when involved in a given task.

Since EAs are loosely inspired by natural evolution, which has been quite successful in evolving various kinds of biological agents, i.e., living creatures, they appear promising for optimizing artificial agents, i.e., robots, too [101]. However, their stochastic nature implies that multiple executions have to be performed to assess their outcome in practice, and calls for a solid statistical analysis for determining the best strategy to be adopted [8, 36]. Unfortunately, repeating several (often thousands) experiments on real robotic platforms can be hugely expensive, time-consuming, and often potentially dangerous. For this reason, most ER applications focus on evolving robots using simulated robot-environment interactions, and then transfer the obtained results to the real robot-environment system (i.e., simulator-to-reality transfer, otherwise known as *sim-to-real* transfer) [91]. Only few works evolve controllers directly on the robot, and often optimize few individuals during few generations, which reduces the effectiveness of the evolutionary methods [37, 53, 101, 148].

Simulation is therefore considered a very powerful and useful tool in the context of evolution of robots, since a long time [184]. However, at the same time, it carries some disadvantages that often result in an under-performing, or, in the worst case, completely ineffective solutions when applied on the real system [40, 62]. This effect is often called *reality gap* [76], i.e., the difference between the effectiveness measured in simulation and the one measured in reality of a solution that has been obtained in simulation (see Fig. 23.20).

Although the term reality gap is widely used in the ER community, this problem is not limited only to applications of EAs in robotics. Instead, as a matter of fact, it can occur on any system designed and developed using simulators, and then implemented in reality. RL applications, for example, can also be affected by this problem when trained using simulators [156, 157]. More generally, in the control system community, where dynamic systems are controlled typically relying on model-based approaches—i.e., where the way the simulation models the real system is known and exploited in the control strategy—the reality gap problem is for example referred to as model-plant mismatch [5, 6, 88, 135, 162]. In this section, however, we focus our attention on the reality gap in ER.

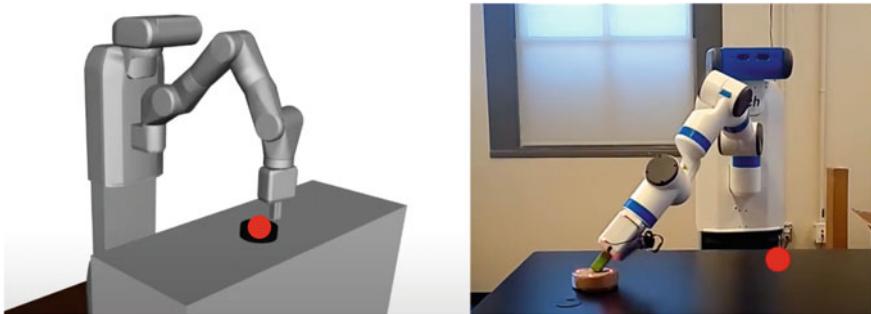


Fig. 23.20 Example of reality gap, from [139]. The robot effector reaches the target (the red circle) in simulation (left), but fails to reach it in the real application (right)

The main questions one needs to ask when dealing with a reality gap are:

1. what are the differences between simulators and reality that affect the effectiveness of the solution found in the simulation?
2. how can the mismatch be reduced?

Of course, as one might expect, the above questions are interrelated, often dependent on the problem addressed, and thus have no one-size-fits-all answers.

From a practical point of view, the mismatches between simulator and reality, being the cause of the reality gap, can be classified into three macro-categories [134]:

- *robot-robot* correspondence,
- *robot-environment* correspondence,
- *environment-environment* correspondence.

The former refers to the physical differences between the simulated robot and the real one, such as morphological differences. Robot-environment differences refer to errors, or approximations, in the dynamic interactions between the robot and the environment, and include both perception and actuation. On the other hand, the latter, i.e., environment-environment correspondence, concerns the misrepresentation of significant features of the environment.

Clearly, once the source of such differences is recognized, there is a trivial solution to reduce the gap: improving the simulator by bringing it as close as possible to the real system. In [110], for example, simulator is integrated with real data, in [76] with noise on sensor level, while in [11, 192] the robot model is directly learned online. [90] presented a broad comparison of model improvement approaches in dealing with the reality gap.

However, this general solution has a negative impact on the efficiency of EAs even when applied in simulation: the time or computational effort needed for finding a solution increases. Simulators with higher accuracy tend to be more demanding in terms of computational time, thus losing one of the main advantages of using simulators. Hence, a trade-off exists between the *effectiveness* of an optimization

technique (i.e., the quality of the solution it finds for a given task), and its *efficiency*. Assuming, therefore, that a certain amount of reality gap must be tolerated, it is necessary to build or tune optimization techniques (including evolutionary ones) in such a way that they match the user expectation in terms of reality gap, effectiveness, and efficiency.

There are different works in the literature proposing approaches able to mitigate the reality gap effect in ER. Although we cannot discuss each of them in-depth, we provide below a categorization of those which work in the optimization phase—i.e., the vast majority of them. A rather different approach is to try to fill the reality gap by building robots that can adapt to changes while they operate. For example, [168] employed ES to realize a form of meta-learning that allows a real legged robot to adapt to changes in its body dynamics (due to robot load and battery voltage change). In principle, this kind of adaptation could be achieved also with controllers that exhibit plasticity, such as SNNs or ANNs with Hebbian learning.

23.5.1 Domain Randomization

The idea of domain randomization is to improve the simulator robustness by providing sufficient simulated variability at the training time, such that the model is able to generalize to real-world data in test. The result is a more robust solution to model variation, and it can be achieved following different strategies: performing adaptive feedback control [117, 147, 182], i.e., closed-loop control strategies in which some measurements are fed back to the controller to properly adapt the control law to instantaneous changes in the environment or in the robot itself, developing robots within a variety of different environments [9, 12, 52], or randomly perturbing model parameters during the evolution [74, 75, 178].

23.5.2 Simulator Flaws Avoidance

Avoid the exploitation of simulator flaws is grounded on the idea that, if the solution search space during evolution is such that it exploits the defects of the simulator, then the learned solution will necessarily be subject to the reality gap when transferred to the real robot. Therefore, the approaches that address the reality gap out of this point of view try to reduce the exploitation of simulator pitfalls automatically adjusting hyper-parameters [47, 48, 180], or slowly increasing the representative power of the search space [49].

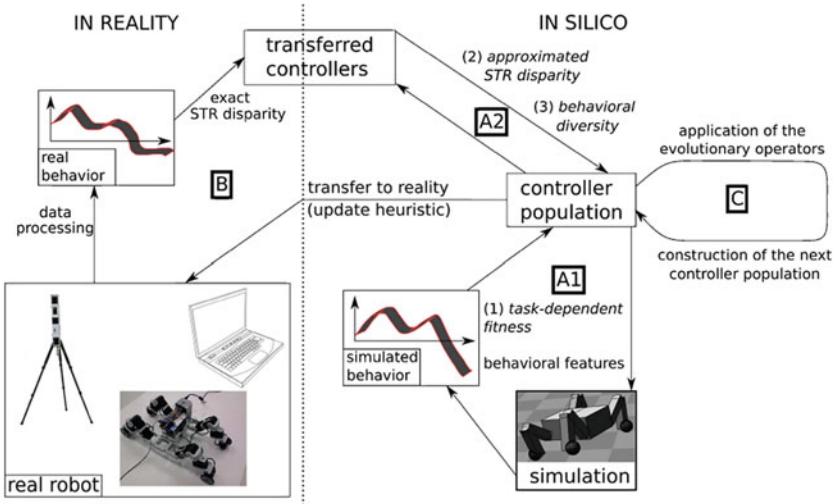


Fig. 23.21 An overview of the algorithm proposed by [91] for evolving robot controllers avoiding to fall in the reality gap problem. The designer needs to identify a few behavioral descriptors that can be computed both in simulation and reality; based on these descriptors, an estimation of the (non-)transferability (STR, i.e., sim-to-real, disparity) of each solution evaluated in simulation is produced using a supervised ML model. The evolution is bi-objective, with one objective being the non-transferability, to be minimized, and the other being the task-dependent effectiveness. The ML model for estimating the non-transferability is maintained and updated by assessing in reality some of the solutions found in simulation at regular intervals. Image taken from [91]

23.5.3 Fostering Rransferability

A promising idea is the one that consists in learning a model that can roughly quantify the reality gap between a simulator and a real robot for each investigated design, i.e., to estimate the design (non-)transferability, and use the estimate to constrain the design process. This is the main idea of [91], where authors formulated the transferability approach, i.e., a bi-objective algorithm in which a task-dependent performance metric is maximized, while a disparity measure between performance in simulation and in reality (i.e., a non-transferability measure) is simultaneously minimized (see Fig. 23.21). The cited work is particularly relevant to this chapter because the authors use a classical ML pipeline for estimating the transferability based on some features that can be computed for each possible solution being evolved.

The same idea has been subsequently pursued in other works in which, for example, the discrepancy between simulator and real robot has been monitored in morphology [95, 158, 183].

23.6 Concluding Remarks and Open Challenges

The field of robotics offers a plethora of opportunities for applying evolutionary optimization. Many components of robots and their tasks can be optimized, rather than manually designed, and the corresponding search spaces are often very large and hardly amenable to be searched with more traditional search methods: hence EAs can deploy their potential as universal search techniques. Moreover, the combination of EC and ML appears to be particularly effective when used to tackle different facets of a larger problem.

There are, however, some open challenges. We believe they are well captured by the recent work of [40], who suggested that future research in ER should attempt to

1. Target more realistic robots and real tasks: this will require to (i) focus on robotic subsystems that are currently overlooked, such as sensors, (ii) consider more complex tasks, such as, e.g., object transportation instead of the simple locomotion, and (iii) make assessment more scalable, in such a way that it can actually be executed for many candidate solutions. We think that the latter point is a particularly fertile terrain for the combined use of EC and ML.
2. Increase sample efficiency, i.e., reduce the number of assessments that are needed to obtain a given solution quality upon optimization.
3. Provide a more solid formalization of the properties of a robotic system that can affect the success of evolutionary optimization. In particular, we think that a finer characterization of the body-brain duality, i.e., their ability to host the cognition that the robot needs to perform its task, would be beneficial.

References

1. Akinci, K., Philippides, A.: Evolving recurrent neural network controllers by incremental fitness shaping. In: Volume ALIFE 2019: the 2019 Conference on Artificial Life of Artificial Life Conference Proceedings, pp. 416–423 (2019)
2. Albrigtsen, S.I., Imenes, A., Goodwin, M., Jiao, L., Nunavath, V.: Neuroevolution of actively controlled virtual characters—an experiment for an eight-legged character. In: Pimenidis, E., Jayne, C. (eds.) *Engineering Applications of Neural Networks*, pp. 94–105. Springer International Publishing, Cham (2018)
3. Auerbach, J., Aydin, D., Maesani, A., Kornatowski, P., Cieslewski, T., Heitz, G., Fernando, P., Loshchilov, I., Daler, L., Floreano, D.: Robogen: robot generation through artificial evolution. In: Volume ALIFE 14: the Fourteenth International Conference on the Synthesis and Simulation of Living Systems of Artificial Life Conference Proceedings, pp. 136–137 (2014)
4. Auerbach, J.E., Bongard, J.C.: Evolving CPPNs to grow three-dimensional physical structures. In: *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, GECCO ’10, pp. 627–634. Association for Computing Machinery, New York, NY, USA (2010)
5. Badwe, A.S., Gudi, R.D., Patwardhan, R.S., Shah, S.L., Patwardhan, S.C.: Detection of model-plant mismatch in MPC applications. *J. Process. Control.* **19**(8), 1305–1313 (2009). Special Section on Hybrid Systems: Modeling, Simulation and Optimization

6. Badwe, A.S., Patwardhan, R.S., Shah, S.L., Patwardhan, S.C., Gudi, R.D.: Quantifying the impact of model-plant mismatch on controller performance. *J. Process. Control.* **20**(4), 408–425 (2010)
7. Bartoli, A., Castelli, M., Medvet, E.: Weighted hierarchical grammatical evolution. *IEEE Trans. Cybern.* **50**(2), 476–488 (2020)
8. Bartz-Beielstein, T., Preuss, M.: Experimental research in evolutionary computation. In: *Proceedings of the 9th Annual Conference Companion on Genetic and Evolutionary Computation, GECCO '07*, pp. 3001–3020. Association for Computing Machinery, New York, NY, USA (2007)
9. Boeing, A., Bräunl, T.: Leveraging multiple simulators for crossing the reality gap. In: *2012 12th International Conference on Control Automation Robotics & Vision (ICARCV)*, pp. 1113–1119 (2012)
10. Bojeri, A., Iacca, G.: Evolutionary optimization of drone trajectories based on optimal reciprocal collision avoidance. In: *2020 27th Conference of Open Innovations Association (FRUCT)*, pp. 18–26 (2020)
11. Bongard, J., Lipson, H.: Once more unto the breach: co-evolving a robot and its simulator. In: *Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems (ALIFE9)*, pp. 57–62 (2004)
12. Bongard, J., Zykov, V., Lipson, H.: Resilient machines through continuous self-modeling. *Sci.* **314**(5802), 1118–1121 (2006)
13. Bongard, J.C., Pfeifer, R.: Evolving complete agents using artificial ontogeny. In: Hara, F., Pfeifer, R. (eds.) *Morpho-functional Machines: the New Species*, pp. 237–258. Springer, Japan, Tokyo (2003)
14. Bredeche, N., Haasdijk, E., Prieto, A.: Embodied evolution in collective robotics: a review. *Front. Robot. AI.* **5** (2018)
15. Bucher, D., Haspel, G., Golowasch, J., Nadim, F.: Central pattern generators, pp. 1–12. John Wiley & Sons, Ltd (2015)
16. Bull, M., Kroo, L.A., Prakash, M.: Excitable mechanics embodied in a walking cilium. In: *APS March Meeting Abstracts, volume 2022 of APS Meeting Abstracts*, pp. Y04.002 (2022)
17. Cáceres, C., Rosário, J.M., Amaya, D.: Approach of kinematic control for a nonholonomic wheeled robot using artificial neural networks and genetic algorithms. In: *2017 International Conference and Workshop on Bioinspired Intelligence (IWobi)*, pp. 1–6. IEEE (2017)
18. Cáceres Flórez, C.A., Rosário, J.M., Amaya, D.: Control structure for a car-like robot using artificial neural networks and genetic algorithms. *Neural Comput. Appl.* **32**(20), 15771–15784 (2020)
19. Cazenille, L., Bredeche, N., Hamann, H., Stradner, J.: Impact of neuron models and network structure on evolving modular robot neural network controllers. In: *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation, GECCO '12*, pp. 89–96. Association for Computing Machinery, New York, NY, USA (2012)
20. Chapelle, F., Bidaud, P.: A closed form for inverse kinematics approximation of general 6R manipulators using genetic programming. In: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No. 01CH37164)*, vol. 4, pp. 3364–3369 (2001)
21. Cheney, N., Bongard, J., Lipson, H.: Evolving soft robots in tight spaces. In: *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, GECCO '15*, pp. 935–942. Association for Computing Machinery, New York, NY, USA (2015)
22. Cheney, N., Bongard, J., SunSpiral, V., Lipson, H.: Scalable co-optimization of morphology and control in embodied machines. *J. R. Soc. Interface.* **15**(143), 20170937 (2018)
23. Cheney, N., MacCurdy, R., Clune, J., Lipson, H.: Unshackling Evolution: Evolving Soft Robots with Multiple Materials and a Powerful Generative Encoding. In: *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation, GECCO '13*, pp. 167–174. Association for Computing Machinery, New York, NY, USA (2013)
24. Chou, C.-Y., Juan, C.-F.: Navigation of an autonomous wheeled robot in unknown environments based on evolutionary fuzzy control. *Inven.* **3**(1) (2018)

25. Clune, J., Beckmann, B.E., Ofria, C., Pennock, R.T.: Evolving coordinated quadruped gaits with the HyperNEAT generative encoding. In: 2009 IEEE Congress on Evolutionary Computation, pp. 2764–2771 (2009)
26. Collins, S., Ruina, A., Tedrake, R., Wisse, M.: Efficient bipedal robots based on passive-dynamic walkers. *Sci.* **307**(5712), 1082–1085 (2005)
27. Conti, E., Madhavan, V., Petroski Such, F., Lehman, J., Stanley, K.O., Clune, J.: Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems, NIPS’18, pp. 5032–5043. Curran Associates Inc., Red Hook, NY, USA (2018)
28. Corucci, F., Cheney, N., Giorgio-Serchi, F., Bongard, J., Laschi, C.: Evolving soft locomotion in aquatic and terrestrial environments: effects of material properties and environmental transitions. *Soft Robot.* **5**(4), 475–495 (2018). PMID: 29985740
29. Custode, L.L., Iacca, G.: A co-evolutionary approach to interpretable reinforcement learning in environments with continuous action spaces. In: 2021 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 1–8 (2021)
30. D’Ambrosio, D.B., Stanley, K.O.: Generative encoding for multiagent learning. In: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, GECCO ’08, pp. 819–826. Association for Computing Machinery, New York, NY, USA (2008)
31. Das, S., Shankar, A., Aggarwal, V.: Training spiking neural networks with a multi-agent evolutionary robotics framework. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO ’21, pp. 858–865. Association for Computing Machinery, New York, NY, USA (2021)
32. De Carlo, M., Ferrante, E., Zeeuw, D., Ellers, J., Meynen, G., Eiben, A.E.: Heritability in morphological robot evolution (2021). [arXiv:2110.11187](https://arxiv.org/abs/2110.11187)
33. Di Paolo, E.: Spike-timing dependent plasticity for evolved robots. *Adapt. Behav.* **10**(3–4), 243–263 (2002)
34. Soorati, M.D., Hamann, H.: The effect of fitness function design on performance in evolutionary robotics: the influence of a priori knowledge. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, GECCO ’15, pp. 153–160. Association for Computing Machinery, New York, NY, USA (2015)
35. Dolinsky, J.-U., Jenkinson, I.D., Colquhoun, G.J.: Application of genetic programming to the calibration of industrial robots. *Comput. Ind.* **58**(3), 255–264 (2007)
36. Doncieux, S., Bredeche, N., Mouret, J.-B., Eiben, A.E.G.: Evolutionary robotics: what, why, and where to. *Front. Robot. AI.* **2** (2015)
37. Doncieux, S., Bredeche, N., Mouret, J.-B., Eiben, A.E.G.: Evolutionary robotics: what, why, and where to. *Front. Robot. AI.* **2** (2015)
38. Downing, K.L.: Adaptive genetic programs via reinforcement learning. In: Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation, GECCO’01, pp. 19–26. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2001)
39. Eiben, A.E., Hart, E.: If it evolves it needs to learn. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion, GECCO ’20, pp. 1383–1384. Association for Computing Machinery, New York, NY, USA (2020)
40. Eiben, A.E.: Real-world robot evolution: why would it (not) work?. *Front. Robot. AI.* **8** (2021)
41. Eiben, A.E., Smith, J.: From evolutionary computation to the evolution of things. *Nat.* **521**(7553), 476–482 (2015)
42. Feng, G.: A survey on analysis and design of model-based fuzzy control systems. *IEEE Trans. Fuzzy Syst.* **14**(5), 676–697 (2006)
43. Ferigo, A., Iacca, G., Medvet, E.: Beyond body shape and brain: evolving the sensory apparatus of voxel-based soft robots. In: Castillo, P.A., Laredo, J.L.J. (eds.) Applications of Evolutionary Computation, pp. 210–226. Springer International Publishing, Cham (2021)
44. Ferigo, A., Iacca, G., Medvet, E., Pigozzi, F.: Evolving Hebbian learning rules in voxel-based soft robots. *IEEE Trans. Cogn. Dev. Syst.* 1–1 (2022)

45. Ferigo, A., Medvet, E., Iacca, G.: Optimizing the sensory apparatus of voxel-based soft robots through evolution and babbling. *SN Comput. Sci.* **3**(2), 109 (2021)
46. Ferigo, A., Soros, L.B., Medvet, E., Iacca, G.: On the entanglement between evolvability and fitness: an experimental study on voxel-based soft robots. In: Volume ALIFE 2022: the 2022 Conference on Artificial Life of Artificial Life Conference Proceedings (2022)
47. Floreano, D., Mattiussi, C.: Evolution of spiking neural controllers for autonomous vision-based robots. In: Gomi, T. (eds.) *Evolutionary Robotics. From Intelligent Robotics to Artificial Life*, pp. 38–61. Springer, Berlin, Heidelberg (2001)
48. Floreano, D., Mondada, F.: Evolution of plastic neurocontrollers for situated agents. In: *Proceeding of the Fourth International Conference on Simulation of Adaptive Behavior (SAB), From Animals to Animats*. ETH Zürich (1996)
49. Francesca, G., Brambilla, M., Brutschy, A., Trianni, V., Birattari, M.: AutoMoDe: a novel approach to the automatic design of control software for robot swarms. *Swarm Intell.* **8**(2), 89–112 (2014)
50. Gaier, A., Ha, D.: Weight agnostic neural networks. In: *Proceedings of the 33rd International Conference on Neural Information Processing Systems*. Curran Associates Inc., Red Hook, NY, USA (2019)
51. Gibson, J.J.: *The Ecological Approach to Visual Perception: Classic Edition*. Psychology press (2014)
52. Glette, K., Johnsen, A.L., Samuelsen, E.: Filling the reality gap: using obstacles to promote robust gaits in evolutionary robotics. In: *2014 IEEE International Conference on Evolvable Systems*, pp. 181–186 (2014)
53. Gongora, M.A., Passow, B.N., Hopgood, A.A.: Robustness analysis of evolutionary controller tuning using real systems. In: *2009 IEEE Congress on Evolutionary Computation*, pp. 606–613 (2009)
54. Gruau, F.: Automatic definition of modular neural networks. *Adapt. Behav.* **3**(2), 151–183 (1994)
55. Guo, J., Hu, P., Li, L., Wang, R.: Design of automatic steering controller for trajectory tracking of unmanned vehicles using genetic algorithms. *IEEE Trans. Veh. Technol.* **61**(7), 2913–2924 (2012)
56. Gupta, A., Savarese, S., Ganguli, S., Fei-Fei, L.: Embodied intelligence via learning and evolution. *Nat. Commun.* **12**(1), 5721 (2021)
57. Ha, D.: Reinforcement learning for improving agent design. *Artif. Life.* **25**(4), 352–365 (2019)
58. Haasdijk, E., Rusu, A.A., Eiben, A.E.: HyperNEAT for locomotion control in modular robots. In: Tempesti, G., Tyrrell, A.M., Miller, J.F. (eds.) *Evolvable Systems: From Biology to Hardware*, pp. 169–180. Springer, Berlin, Heidelberg (2010)
59. Hagras, H., Pounds-Cornish, A., Colley, M., Callaghan, V., Clarke, G.: Evolving spiking neural network controllers for autonomous robots. In: *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, vol. 5, pp. 4620–4626 (2004)
60. Hallawa, A., Born, T., Schmeink, A., Dartmann, G., Peine, A., Martin, L., Iacca, G., Eiben, A.E., Ascheid, G.: Evo-RL: evolutionary-driven reinforcement learning. In: *Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO '21*, pp. 153–154. Association for Computing Machinery, New York, NY, USA (2021)
61. Hallawa, A., Schug, S., Iacca, G., Ascheid, G.: Evolving instinctive behaviour in resource-constrained autonomous agents using grammatical evolution. In: Castillo, P.A., Laredo, J.L.J., de Vega, F.F. (eds.) *Applications of Evolutionary Computation*, pp. 369–383. Springer International Publishing, Cham (2020)
62. Hasselmann, K., Ligot, A., Ruddick, J., Birattari, M.: Empirical assessment and comparison of neuro-evolutionary methods for the automatic off-line design of robot swarms. *Nat. Commun.* **12**(1), 4345 (2021)
63. Hauser, H., Ijspeert, A.J., Füchslin, R.M., Pfeifer, R., Maass, W.: Towards a theoretical foundation for morphological computation with compliant bodies. *Biol. Cybern.* **105**(5), 355–370 (2011)

64. Hein, D., Udluft, S., Runkler, T.A.: Interpretable policies for reinforcement learning by genetic programming. *Eng. Appl. Artif. Intell.* **76**, 158–169 (2018)
65. Hornby, G.S., Pollack, J.B.: Evolving L-systems to generate virtual creatures. *Comput. & Graph.* **25**(6), 1041–1048 (2001). *Artificial Life*
66. Hornby, G.S., Takamura, S., Yamamoto, T., Fujita, M.: Autonomous evolution of dynamic gaits with two quadruped robots. *IEEE Trans. Robot.* **21**(3), 402–410 (2005)
67. Hossain, D., Capi, G.: Multiobjective evolution of deep learning parameters for robot manipulator object recognition and grasping. *Adv. Robot.* **32**(20), 1090–1101 (2018)
68. Hu, Y., Wu, X., Geng, P., Li, Z.: Evolution strategies learning with variable impedance control for grasping under uncertainty. *IEEE Trans. Ind. Electron.* **66**(10), 7788–7799 (2018)
69. Hupkes, E., Jelisavcic, M., Eiben, A.E.: Revolve: a versatile simulator for online robot evolution. In: Sim, K., Kaufmann, P. (eds.) *Applications of Evolutionary Computation*, pp. 687–702. Springer International Publishing, Cham (2018)
70. Iovino, M., Scukins, E., Styrudd, J., Ögren, P., Smith, C.: A survey of behavior trees in robotics and AI. *Robot. Auton. Syst.* **154**, 104096 (2022)
71. Iovino, M., Styrudd, J., Falco, P., Christian, S.: Learning behavior trees with genetic programming in unpredictable environments. In: 2021 IEEE International Conference on Robotics and Automation (ICRA), pp. 4591–4597 (2021)
72. Izhikevich, E.M.: Simple model of spiking neurons. *IEEE Trans. Neural Netw.* **14**(6), 1569–1572 (2003)
73. Jaderberg, M., Dalibard, V., Osindero, S., Czarnecki, W.M., Donahue, J., Razavi, A., Vinyals, O., Green, T., Dunning, I., Simonyan, K., et al.: Population based training of neural networks (2017). [arXiv:1711.09846](https://arxiv.org/abs/1711.09846)
74. Jakobi, N.: Evolutionary robotics and the radical envelope-of-noise hypothesis. *Adapt. Behav.* **6**(2), 325–368 (1997)
75. Jakobi, N.: Minimal simulations for evolutionary robotics. Ph.D. thesis, University of Sussex (1998)
76. Jakobi, N., Husbands, P., Harvey, I.: Noise and the reality gap: the use of simulation in evolutionary robotics. In: Morán, F., Moreno, A., Merelo, J.J., Chacón, P. (eds.) *Advances in Artificial Life*, pp. 704–720. Springer, Berlin, Heidelberg (1995)
77. Jansen, T.: Strandbeests. *Arch. Des.* **78**(4), 22–27 (2008)
78. Jelisavcic, M., Glette, K., Haasdijk, E., Eiben, A.E.: Lamarckian evolution of simulated modular robots. *Front. Robot. AI.* **6** (2019)
79. Yang, J., Zhang, Q., Zeng, J., Yin, Q.: Survey of evolutionary behavior tree algorithm. *J. Syst. Simul.* **33**(10), 2315 (2021)
80. Joachimczak, M., Suzuki, R., Arita, T.: Artificial metamorphosis: evolutionary design of transforming, soft-bodied robots. *Artif. Life.* **22**(3), 271–298 (2016)
81. Jones, S., Studley, M., Hauer, S., Winfield, A.: Evolving behaviour trees for swarm robotics. In: Grob, R., Kolling, A., Berman, S., Fazzoli, E., Martinoli, A., Matsuno, F., Gauci, M. (eds.) *Distributed Autonomous Robotic Systems: The 13th International Symposium*, pp. 487–501. Springer International Publishing, Cham (2018)
82. Juang, C.-F., Chang, Y.-C.: Evolutionary-group-based particle-swarm-optimized fuzzy controller with application to mobile-robot navigation in unknown environments. *IEEE Trans. Fuzzy Syst.* **19**(2), 379–392 (2011)
83. Juang, C.-F., Hsu, C.-H.: Reinforcement ant optimized fuzzy controller for mobile-robot wall-following control. *IEEE Trans. Ind. Electron.* **56**(10), 3931–3940 (2009)
84. Juang, C.-F., Jeng, T.-L., Chang, Y.-C.: An interpretable fuzzy system learned through online rule generation and multiobjective ACO with a mobile robot control application. *IEEE Trans. Cybern.* **46**(12), 2706–2718 (2016)
85. Kaiser, T.K., Hamann, H.: Engineered self-organization for resilient robot self-assembly with minimal surprise. *Robot. Auton. Syst.* **122**, 103293 (2019)
86. Kalra, P., Mahapatra, P.B., Aggarwal, D.K.: An evolutionary approach for solving the multi-modal inverse kinematics problem of industrial robots. *Mech. Mach. Theory.* **41**(10), 1213–1229 (2006)

87. Kamimura, A., Kurokawa, H., Yoshida, E., Tomita, K., Kokaji, S., Murata, S.: Distributed adaptive locomotion by a modular robotic system, M-TRAN II. In: 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No. 04CH37566), vol. 3, pp. 2370–2377. IEEE (2004)
88. Kano, M., Shigi, Y., Hasebe, S., Ooyama, S.: Detection of significant model-plant mismatch from routine operation data of model predictive control system. IFAC Proc. Vol. **43**(5), 685–690 (2010). 9th IFAC Symposium on Dynamics and Control of Process Systems
89. Khadka, S., Turner, K.: Evolution-guided policy gradient in reinforcement learning. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems, NIPS'18, pp. 1196–1208. Curran Associates Inc, Red Hook, NY, USA (2018)
90. Klaus, G., Glette, K., Tørresen, J.: A comparison of sampling strategies for parameter estimation of a robot simulator. In: Noda, I., Ando, N., Brugali, D., Kuffner, J.J. (eds.) Simulation, Modeling, and Programming for Autonomous Robots, pp. 173–184. Springer, Berlin, Heidelberg (2012)
91. Koos, S., Mouret, J.-B., Doncieux, S.: The transferability approach: crossing the reality gap in evolutionary robotics. IEEE Trans. Evol. Comput. **17**(1), 122–145 (2012)
92. Koza, J.R.: Genetic Programming: a Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems, vol. 34. Stanford University, Department of Computer Science Stanford, CA (1990)
93. Kriegman, S.: Blackiston, D., Levin, M., Bongard, J.: Kinematic self-replication in reconfigurable organisms. In: Proc. Natl. Acad. Sci. **118**(49), e2112672118 (2021)
94. Kriegman, S., Cheney, N., Bongard, J.: How morphological development can guide evolution. Sci. Rep. **8**(1), 13934 (2018)
95. Kriegman, S., Nasab, A.M., Shah, D., Steele, H., Branin, G., Levin, M., Bongard, J., Kramer-Bottiglio, R.: Scalable sim-to-real transfer of soft robot designs. In: 2020 3rd IEEE International Conference on Soft Robotics (RoboSoft), pp. 359–366 (2020)
96. Kriegman, S., Walker, S., Shah, D., Levin, M., Kramer-Bottiglio, R., Bongard, J.: Automated shapeshifting for function recovery in damaged robots. Proc. Robot. Sci. Syst. (2019)
97. Le Goff, L.K., Buchanan, E., Hart, E., Eiben, A.E., Li, W., De Carlo, M., Winfield, A.F., Hale, M.F., Woolley, R., Angus, M., et al.: Morpho-evolution with learning using a controller archive as an inheritance mechanism. IEEE Trans. Cogn. Dev. Syst. (2022)
98. Levin, M., Pietak, A.M., Bischof, J.: Planarian regeneration as a model of anatomical homeostasis: recent progress in biophysical and computational approaches. Semin. Cell & Dev. Biol. **87**, 125–144 (2019)
99. Lindenmayer, A.: Mathematical models for cellular interactions in development I. Filaments with one-sided inputs. J. Theor. Biol. **18**(3), 280–299 (1968)
100. Lipson, H., Sunspiral, V., Bongard, J., Cheney, N.: On the difficulty of co-optimizing morphology and control in evolved virtual creatures. In: Volume ALIFE 2016, The Fifteenth International Conference on the Synthesis and Simulation of Living Systems of Artificial Life Conference Proceedings, pp. 226–233 (2022)
101. Long, J.: Darwin's Devices: What Evolving Robots Can Teach Us About the History of Life and the Future of Technology. Hachette, UK (2012)
102. Luo, J., Stuurman, A.C., Tomczak, J.M., Ellers, J., Eiben, A.E.: The effects of learning in morphologically evolving robot systems. Front. Robot. AI. **9** (2022)
103. Mabu, S., Hirasawa, K., Hu, J.: Genetic network programming with reinforcement learning and its performance evaluation. In: Deb, K. (ed.) Genetic and Evolutionary Computation—GECCO 2004, pp. 710–711. Springer, Berlin, Heidelberg (2004)
104. Mabu, S., Hirasawa, K., Hu, J.: A Graph-based evolutionary algorithm: genetic network programming (GNP) and its extension using reinforcement learning. Evol. Comput. **15**(3), 369–398 (2007)
105. Mahdavi, S.H., Bentley, P.J.: An evolutionary approach to damage recovery of robot motion with muscles. In: European Conference on Artificial Life, pp. 248–255. Springer (2003)
106. Marsland, S.: Machine Learning: an Algorithmic Perspective. CRC press (2015)
107. McGeer, T.: Passive dynamic walking. Int. J. Robot. Res. **9**(2), 62–82 (1990)

108. Medvet, E., Bartoli, A., De Lorenzo, A., Fidel, G.: Evolution of distributed neural controllers for voxel-based soft robots. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference, GECCO '20, pp. 112–120. Association for Computing Machinery, New York, NY, USA (2020)
109. Mégane, J., Lourenço, N., Machado, P.: Probabilistic grammatical evolution. In: Ting, H., Nuno, L., Medvet, E. (eds.) Genetic Programming, pp. 198–213. Springer International Publishing, Cham (2021)
110. Miglino, O., Lund, H.H., Nolfi, S.: Evolving mobile robots in simulated and real environments. *Artif. Life.* **2**(4), 417–434 (1995)
111. Miras, K.: Constrained by design: influence of genetic encodings on evolved traits of robots. *Front. Robot. AI.* **8** (2021)
112. Miras, K., De Carlo, M., Akhatou, S., Eiben, A.E.: Evolving-controllers versus learning-controllers for morphologically evolvable robots. In: Castillo, P.A., Laredo, J.L.J., de Vega, F.F. (eds.) Applications of Evolutionary Computation, pp. 86–99. Springer International Publishing, Cham (2020)
113. Miras, K., Eiben, A.E.: Effects of environmental conditions on evolved robot morphologies and behavior. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '19, pp. 125–132. Association for Computing Machinery, New York, NY, USA (2019)
114. Miras, K., Ferrante, E., Eiben, A.E.: Environmental influences on evolvable robots. *PLOS ONE.* **15**(5), 1–23 (2020)
115. Miras, K., Haasdijk, E., Glette, K., Eiben, A.E.: Search space analysis of evolvable robot morphologies. In: Sim, K., Kaufmann, P. (eds.) Applications of Evolutionary Computation, pp. 703–718. Springer International Publishing, Cham (2018)
116. Möller, F.J.D., Bernardino, H.S., Gonçalves, L.B., Soares, S.S.R.F.: A reinforcement learning based adaptive mutation for cartesian genetic programming applied to the design of combinational logic circuits. In: Cerri, R., Prati, R.C. (eds.) Intelligent Systems, pp. 18–32. Springer International Publishing, Cham (2020)
117. Montanier, J.-M., Bredeche, N.: Embedded evolutionary robotics: the (1+1)-restart-online adaptation algorithm. In: Doncieux, S., Bredèche, N., Mouret, J.-B. (eds.) New Horizons in Evolutionary Robotics, pp. 155–169. Springer, Berlin, Heidelberg (2011)
118. Mouret, J.-B., Clune, J.: Illuminating search spaces by mapping elites (2015). [arXiv:1504.04909](https://arxiv.org/abs/1504.04909)
119. Nadizar, G., Medvet, E., Ramstad, H.H., Nichele, S., Pellegrino, F.A., Zullich, M.: Merging pruning and neuroevolution: towards robust and efficient controllers for modular soft robots. *Knowl. Eng. Rev.* **37**, e3 (2022)
120. Nadizar, G., Medvet, E., Miras, K.: On the schedule for morphological development of evolved modular soft robots. In: Medvet, E., Pappa, G., Xue, B. (eds.) Genetic Programming, pp. 146–161. Springer International Publishing, Cham (2022)
121. Nadizar, G., Medvet, E., Nichele, S., Pontes-Filho, S.: Collective control of modular soft robots via embodied Spiking Neural Cellular Automata (2022). [arXiv:2204.02099](https://arxiv.org/abs/2204.02099)
122. Nadizar, G., Medvet, E., Pellegrino, F.A., Zullich, M., Nichele, S.: On the Effects of Pruning on Evolved Neural Controllers for Soft Robots. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO '21, pp. 1744–1752. Association for Computing Machinery, New York, NY, USA (2021)
123. Najarro, E., Risi, S.: Meta-learning through hebbian plasticity in random networks. In: Proceedings of the 34th International Conference on Neural Information Processing Systems, NIPS'20. Curran Associates Inc., Red Hook, NY, USA (2020)
124. Naya-Varela, M., Faina, A., Mallo, A., Duro, R.J.: A study of growth based morphological development in neural network controlled walkers. *Neurocomputing.* **500**, 279–294 (2022)
125. Neupane, A., Goodrich, M.: Learning swarm behaviors using grammatical evolution and behavior trees. In: Proceedings of the 28th International Joint Conference on Artificial Intelligence, IJCAI'19, pp. 513–520. AAAI Press (2019)

126. Neupane, A., Goodrich, M.A.: Designing emergent swarm behaviors using behavior trees and grammatical evolution. In: Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19, pp. 2138–2140. International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC (2019)
127. Nguyen, A.-T., Taniguchi, T., Eciolaza, L., Campos, V., Palhares, R., Sugeno, M.: Fuzzy control systems: past, present and future. *IEEE Comput. Intell. Mag.* **14**(1), 56–68 (2019)
128. Niekum, S., Barto, A.G., Spector, L.: Genetic programming for reward function search. *IEEE Trans. Auton. Ment. Dev.* **2**(2), 83–90 (2010)
129. Nolfi, S., Bongard, J., Husbands, P., Floreano, D.: Evolutionary Robotics, pp. 2035–2068. Springer International Publishing, Cham (2016)
130. Nolfi, S., Floreano, D.: Learning and evolution. *Auton. Robot.* **7**(1), 89–113 (1999)
131. Nolfi, S., Parisi, D.: Learning to adapt to changing environments in evolving neural networks. *Adapt. Behav.* **5**(1), 75–98 (1996)
132. Norstein, E.S., Ellefsen, K.O., Glette, K.: Open-Ended Search for Environments and Adapted Agents Using MAP-Elites. In: Laredo, J.L.J., Hidalgo, J.I., Babaagba, K.O. (eds.) Applications of Evolutionary Computation, pp. 651–666. Springer International Publishing, Cham (2022)
133. Nygaard, T.F., Martin, C.P., Samuels, E., Torresen, J., Glette, K.: Real-world evolution adapts robot morphology and control to hardware limitations. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '18, pp. 125–132. Association for Computing Machinery, New York, NY, USA (2018)
134. O'Dowd, P.J., Winfield, A.F.T., Studley, M.: The distributed co-evolution of an embodied simulator and controller for swarm robot behaviours. In: 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 4995–5000 (2011)
135. Olivier, L.E., Craig, I.K.: Model-plant mismatch detection and model update for a run-of-mine ore milling circuit under model predictive control. *J. Process. Control.* **23**(2), 100–107 (2013). IFAC World Congress Special Issue
136. Pathak, D., Lu, C., Darrell, T., Phillip, I., Efros, A.A.: Learning to Control Self-Assembling Morphologies: a Study of Generalization via Modularity. Curran Associates Inc., Red Hook, NY, USA (2019)
137. Paul, S.K., Bhau mik, P.: A reinforcement learning agent based on genetic programming and universal search. In: 2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS), pp. 122–128 (2020)
138. Pedersen, J.W., Risi, S.: Evolving and merging hebbian learning rules: increasing generalization by decreasing the number of rules. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '21, pp. 892–900. Association for Computing Machinery, New York, NY, USA (2021)
139. Peng, X.B., Andrychowicz, M., Zaremba, W., Abbeel, P.: Sim-to-real transfer of robotic control with dynamics randomization. In: 2018 IEEE International Conference on Robotics and Automation (ICRA), pp. 3803–3810 (2018)
140. Pfeifer, R., Bongard, J.: How the body shapes the way we think: a new view of intelligence. MIT press (2006)
141. Pigozzi, F.: Shape change and control of pressure-based soft agents. In: Holler, S., Loeffler, R., Bartlett, S. (eds.) ALIFE 2022: the 2022 Conference on Artificial Life, vol. 37, pp. 1–10. MIT Press (2022)
142. Pigozzi, F., Medvet, E., Bartoli, A., Rochelli, M.: Factors impacting diversity and effectiveness of evolved modular robots. *ACM Trans. Evol. Learn.* **3**(1), 1–33 (2023)
143. Pigozzi, F., Tang, Y., Medvet, E., Ha, D.: Evolving modular soft robots without explicit inter-module communication using local self-attention. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '22, pp. 148–157. Association for Computing Machinery, New York, NY, USA (2022)
144. Pontes-Filho, S., Walker, K., Najarro, E., Nichele, S., Risi, S.: A unified substrate for body-brain co-evolution (2022). [arXiv:2203.12066](https://arxiv.org/abs/2203.12066)
145. Pourchot, A., Perrin, N., Sigaud, O.: Importance mixing: improving sample reuse in evolutionary policy search methods (2018). [arXiv:1808.05832](https://arxiv.org/abs/1808.05832)

146. Precup, R.-E., Hellendoorn, H.: A survey on industrial applications of fuzzy control. *Comput. Ind.* **62**(3), 213–226 (2011)
147. Qiu, H., Garratt, M., Howard, D., Anavatti, S.: Towards crossing the reality gap with evolved plastic neurocontrollers. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference, GECCO '20, pp. 130–138. Association for Computing Machinery, New York, NY, USA (2020)
148. Regan, W., Van Breugel, F., Lipson, H.: Towards Evolvable Hovering Flight on a Physical Ornithopter. *Alife X*, Bloomington, USA (2006)
149. Reuter, J., Steup, C., Mostaghim, S.: Genetic programming-based inverse kinematics for robotic manipulators. In: Medvet, E., Pappa, G., Xue, B. (eds.) *Genetic Programming*, pp. 130–145. Springer International Publishing, Cham (2022)
150. Reyes, P., Escobar, M.-J.: Neuroevolutionary algorithms for learning gaits in legged robots. *IEEE Access.* **7**, 142406–142420 (2019)
151. Rosenzweig, M.R., Breedlove, S.M., Leiman, A.L.: *Biological Psychology: an Introduction to Behavioral, Cognitive, and Clinical Neuroscience*. Sinauer Associates (2002)
152. Rothlauf, F.: On the locality of representations. In: Proceedings of the 2003 International Conference on Genetic and Evolutionary Computation: PartII, GECCO'03, pp. 1608–1609. Springer-Verlag, Berlin, Heidelberg (2003)
153. Rothlauf, F.: *Representations for Genetic and Evolutionary Algorithms*, pp. 9–32. Springer, Berlin, Heidelberg (2006)
154. Roy, K., Jaiswal, A., Panda, P.: Towards spike-based machine intelligence with neuromorphic computing. *Nat.* **575**(7784), 607–617 (2019)
155. Salimans, T., Ho, J., Chen, X., Sidor, S., Sutskever, I.: Evolution strategies as a scalable alternative to reinforcement learning (2017). [arXiv:1703.03864](https://arxiv.org/abs/1703.03864)
156. Salvato, E., Fenu, G., Medvet, E., Pellegrino, F.A.: Characterization of modeling errors affecting performances of a robotics deep reinforcement learning controller in a sim-to-real transfer. In: 2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO), pp. 1154–1159 (2021)
157. Salvato, E., Fenu, G., Medvet, E., Pellegrino, F.A.: Crossing the reality gap: a survey on sim-to-real transferability of robot controllers in reinforcement learning. *IEEE Access.* **9**, 153171–153187 (2021)
158. Samuelsen, E., Glette, K.: Some distance measures for morphological diversification in generative evolutionary robotics. In: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, GECCO '14, pp. 721–728. Association for Computing Machinery, New York, NY, USA (2014)
159. Sasaki, H., Kubota, N.: Virus-evolutionary genetic algorithm for fuzzy spiking neural network of a mobile robot in a dynamic environment. In: 2006 SICE-ICASE International Joint Conference, pp. 4214–4219 (2006)
160. Sasaki, H., Kubota, N.: Distributed behavior learning of multiple mobile robots based on spiking neural network and steady-state genetic algorithm. In: 2009 IEEE Workshop on Robotic Intelligence in Informationally Structured Space, pp. 73–78 (2009)
161. Schepers, K.Y.W., Tijmons, S., de Visser, C.C., de Croon, G.C.H.E.: Behavior trees for evolutionary robotics †. *Artif. Life.* **22**(1), 23–48 (2016)
162. Selvanathan, S., Tangirala, A.K.: Diagnosis of poor control loop performance due to model-plant mismatch. *Ind. & Eng. Chem. Res.* **49**(9), 4210–4229 (2010)
163. Seriani, S., Marcini, L., Caruso, M., Gallina, P., Medvet, E.: Crowded environment navigation with NEAT: impact of perception resolution on controller optimization. *J. Intell. & Robot. Syst.* **101**(2), 36 (2021)
164. Sigaud, O., Stulp, F.: Policy search in continuous action domains: an overview. *Neural Netw.* **113**, 28–40 (2019)
165. Sims, K.: Evolving 3D morphology and behavior by competition. *Artif. Life.* **1**(4), 353–372 (1994)
166. Sims, K.: Evolving virtual creatures. In: Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '94, pp. 15–22. Association for Computing Machinery, New York, NY, USA (1994)

167. Sipper, M., Sanchez, E., Mange, D., Tomassini, M., Perez-Uribe, A., Stauffer, A.: A phylogenetic, ontogenetic, and epigenetic view of bio-inspired hardware systems. *IEEE Trans. Evol. Comput.* **1**(1), 83–97 (1997)
168. Song, X., Yang, Y., Choromanski, K., Caluwaerts, K., Gao, W., Finn, C., Tan, J.: Rapidly adaptable legged robots via evolutionary meta-learning. In: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 3769–3776 (2020)
169. Stanley, K.O.: Compositional pattern producing networks: a novel abstraction of development. *Genet. Program. Evolvable Mach.* **8**(2), 131–162 (2007)
170. Stanley, K.O., D'Ambrosio, D.B., Gauci, J.: A hypercube-based encoding for evolving large-scale neural networks. *Artif. Life.* **15**(2), 185–212 (2009)
171. Stanley, K.O., Miikkulainen, R.: Efficient reinforcement learning through evolving neural network topologies. In: Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation, GECCO'02, pp. 569–577. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2002)
172. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. *Evol. Comput.* **10**(2), 99–127 (2002)
173. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT press (2018)
174. Talamini, J., Medvet, E., Bartoli, A., De Lorenzo, A.: Evolutionary synthesis of sensing controllers for voxel-based soft robots. In: Volume ALIFE 2019: the 2019 Conference on Artificial Life of Artificial Life Conference Proceedings, pp. 574–581 (2019)
175. Tang, Y., Nguyen, D., Ha, D.: Neuroevolution of self-interpretable agents. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference, GECCO '20, pp. 414–424. Association for Computing Machinery, New York, NY, USA (2020)
176. Tavanaei, A., Ghodrati, M., Kheradpisheh, S.R., Masquelier, T., Maida, A.: Deep learning in spiking neural networks. *Neural Netw.* **111**, 47–63 (2019)
177. Téllez, R.A., Angulo, C., Pardo, D.E.: Evolving the walking behaviour of a 12 DOF quadruped using a distributed neural architecture. In: Ijspeert, A.J., Masuzawa, T., Kusumoto, S. (eds.) Biologically Inspired Approaches to Advanced Information Technology, pp. 5–19. Springer, Berlin, Heidelberg (2006)
178. Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., Abbeel, P.: Domain randomization for transferring deep neural networks from simulation to the real world. In: 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 23–30 (2017)
179. Tuci, E., Massera, G., Nolfi, S.: Active categorical perception in an evolved anthropomorphic robotic arm. In: 2009 IEEE Congress on Evolutionary Computation, pp. 31–38 (2009)
180. Urzelai, J., Floreano, D.: Evolutionary robotics: coping with environmental change. In: Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation, GECCO'00, pp. 941–948. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2000)
181. Valsalam, V.K., Miikkulainen, R.: Modular neuroevolution for multilegged locomotion. In: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, GECCO '08, pp. 265–272. Association for Computing Machinery, New York, NY, USA (2008)
182. van Diggelen, F., Babuska, R., Eiben, A.E.: The effects of adaptive control on learning directed locomotion. In: 2020 IEEE Symposium Series on Computational Intelligence (SSCI), pp. 2117–2124 (2020)
183. van Diggelen, F., Ferrante, E., Harrak, N., Luo, J., Zeeuw, D., Eiben, A.E.: The influence of robot traits and evolutionary dynamics on the reality gap. *IEEE Trans. Cogn. Dev. Syst.* (2021)
184. Varela, F.J., Bourgine, P.: Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life. MIT press (1992)
185. Veenstra, F., Faina, A., Risi, S., Stoy, K.: Evolution and morphogenesis of simulated modular robots: a comparison between a direct and generative encoding. In: Squillero, G., Sim, K. (eds.) Applications of Evolutionary Computation, pp. 870–885. Springer International Publishing, Cham (2017)

186. Wang, M., Luo, J., Fang, J., Yuan, J.: Optimal trajectory planning of free-floating space manipulator using differential evolution algorithm. *Adv. Space Res.* **61**(6), 1525–1536 (2018)
187. Wang, R., Lehman, J., Clune, J., Stanley, K.O.: Paired open-ended trailblazer (poet): endlessly generating increasingly complex and diverse learning environments and their solutions (2019). [arXiv:1901.01753](https://arxiv.org/abs/1901.01753)
188. Wen, R., Guo, Z., Zhao, T., Ma, X., Wang, Q., Wu, Z.: Neuroevolution of augmenting topologies based musculoskeletal arm neurocontroller. In: 2017 IEEE International Instrumentation and Measurement Technology Conference (I2MTC), pp. 1–6 (2017)
189. Willyard, C.: New human gene tally reignites debate. *Nat.* **558**(7710), 354–356 (2018)
190. Xu, S., Moriguchi, H., Honiden, S.: Sample efficiency analysis of neuroevolution algorithms on a quadruped robot. In: 2013 IEEE Congress on Evolutionary Computation, pp. 2170–2177 (2013)
191. Zador, A.M.: A critique of pure learning and what artificial neural networks can learn from animal brains. *Nat. Commun.* **10**(1), 3770 (2019)
192. Zagal, J.C., Ruiz del Solar, J., Vallejos, P.: Back to reality: crossing the reality gap in evolutionary robotics. *IFAC Proc. Vol.* **37**(8), 834–839 (2004). IFAC/EURON Symposium on Intelligent Autonomous Vehicles, Lisbon, Portugal, 5–7 July 2004
193. Zhang, H., Zhou, A., Lin, X.: Interpretable policy derivation for reinforcement learning based on evolutionary feature synthesis. *Complex & Intell. Syst.* **6**(3), 741–753 (2020)
194. Zhang, T., Zhang, W., Gupta, M.M.: An underactuated self-reconfigurable robot and the reconfiguration evolution. *Mech. Mach. Theory* **124**, 248–258 (2018)
195. Zhao, A., Xu, J., Konaković-Luković, M., Hughes, J., Spielberg, A., Rus, D., Matusik, W.: RoboGrammar: graph grammar for terrain-optimized robot design. *ACM Trans. Graph.* **39**(6) (2020)
196. Zou, X., Scott, E., Johnson, A., Chen, K., Nitz, D., De Jong, K., Krichmar, J.: Neuroevolution of a recurrent neural network for spatial and working memory in a simulated robotic environment. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO '21, pp. 289–290. Association for Computing Machinery, New York, NY, USA (2021)

Chapter 24

Evolutionary Machine Learning in Finance



Michael O'Neill and Anthony Brabazon

Abstract One way to measure the impact of our field of research on finance is to analyse the adoption of evolutionary machine learning in the finance literature. In this study, we focus on articles appearing in the top-ranked journals in finance. A number of interesting observations are made including that there appears to be a trend in the adoption of evolutionary machine learning across a growing and diverse set of topics.

24.1 Introduction

In this chapter, we examine the uptake of evolutionary machine learning in the finance literature. Evolutionary machine learning involves the use of evolutionary computation for machine learning tasks including pre-processing, learning and post-processing [1]. By evolutionary computation we include approaches such as genetic algorithms, genetic programming, evolutionary strategies, evolutionary programming, differential evolution, etc.

Within the evolutionary computation community there has been a keen interest in the application of evolutionary computation to problems in finance, including areas such as portfolio optimisation, trading and credit rating. There have been dedicated workshops and conference strands focussing on applications in finance, such as the EvoFIN event which was held annually at the EvoStar conferences from 2007 to 2018 with proceedings published by Springer (e.g. [2]), the Real-World Applications track at GECCO, and at IEEE CEC track on Finance and Economics. There have also been a number of books and edited books (e.g. [3–13, 15]), and the GP-bibliography (<http://gpbib.cs.ucl.ac.uk/>) contains many examples of the application of Genetic Programming to finance.

M. O'Neill (✉) · A. Brabazon
Natural Computing Research & Applications Group, School of Business, University College
Dublin, Dublin, Ireland
e-mail: m.oneill@ucd.ie

It is perhaps not surprising that researchers in evolutionary computation have found the world of finance as an attractive application domain. Evolutionary computation being inspired by the adaptive process of biological evolution, and producing tools that might be capable of performing well in such environments, it seems a natural fit to the complex adaptive systems that the world of finance inhabits.

A recent study by Brabazon et al. [14] focussed on the application of genetic programming to finance and economics, and noted the scant evidence of the uptake of genetic programming in the mainstream finance community. EC researchers tending to favour publication in computer science venues, and focussing more on method development than advances in the domain of application of finance itself. The main contribution of this study is to analyse the uptake of evolutionary machine learning in the mainstream finance literature, in order to ascertain the impact our community may be having upon the community of scholars in this application domain.

In the remainder of this chapter, we set out our approach in Sect. 24.2 detailing the set of finance journals we examine, before highlighting our main findings in Sect. 24.3. We conclude the chapter with some Discussion (Sect. 24.4) and a Summary and Conclusions in Sect. 24.5.

24.2 EML in Finance Journals

We focus our attention on the top 50 finance journals, as determined by the FT50 list,¹ the Association of Business Schools (ABS) Academic Journal Guide 2021² and Google Metrics.³ Table 24.1 lists the complete set of journals alongside which list they are members of. The membership of each list was accessed on 21 October 2022. The FT50 list contains a set of 50 journals that are used by the Financial Times as a component of their method to rank global business schools, based on the number of publications in this list by members of the faculty in each school. By our categorisation six of the FT50 journals are in the field of finance, and these are the ones we focus on. The UK's Association of Business Schools list is based on both bibliometrics and qualitative views of a scientific committee, and ranks journals from the lowest grade of ABS1 up to the highest of ABS4*. Many of the ABS4* journals also appear in the FT50 list. While the ABS list has its origins in the UK, it is recognised by many European business schools. In this study, we focus on the higher ranked ABS journals including ABS3, ABS4 and ABS4* listed under their category of **Finance**. For Google Metrics, we focus on the sub-category of **Finance** within the category of **Business, Economics & Management**. Google Metrics ranks journals based on citation counts captured on Google Scholar, using their calculation of a 5-year h-index.

¹ <https://www.ft.com/content/3405a512-5cbb-11e1-8f1f-00144feabdc0>.

² <https://charteredabs.org/academic-journal-guide-2021/>.

³ <https://scholar.google.com>.

Table 24.1 The journals examined in this study represent the majority of top academic journals in finance according to the FT50, Association of Business Schools (ABS) Journal Guide 2021, and Google Metrics (Top 20). We have omitted one ABS3 journal (Journal of Portfolio Management) from this study as access to it from our institution is behind a paywall

Journal	#EML Articles	FT50 4 * /4	ABS 3	ABS (Top20)	Google Metrics
Quantitative Finance	48		✓		
Journal of Banking & Finance	17		✓	✓	
Insurance: Mathematics and Economics	14		✓		
European Journal of Finance	9		✓		
Finance Research Letters	9			✓	
International Review of Financial Analysis	8		✓	✓	
Journal of International Money and Finance	8		✓	✓	
International Journal of Finance and Economics	7		✓		
Journal of Finance	7	✓	✓		✓
Journal of International Financial Markets, Institutions and Money	7		✓	✓	
Journal of Empirical Finance	6		✓		
Research in International Business and Finance	6			✓	
Journal of Financial Economics	5	✓	✓		✓
Emerging Markets Finance and Trade	4			✓	
International Review of Economics & Finance	4			✓	
Journal of Futures Markets	4			✓	
Journal of Risk and Financial Management	4			✓	
Review of Finance	4	✓	✓		✓
Review of Financial Studies	4	✓	✓		✓
The Quarterly Review of Economics and Finance	4			✓	
Journal of Risk and Insurance	3			✓	
European Financial Management	2			✓	
Journal of Financial and Quantitative Analysis	2	✓	✓		✓
Journal of Financial Econometrics	2			✓	
Pacific-Basin Finance Journal	2				✓
Review of Economic Studies	2	✓	✓		
Finance and Stochastics	1			✓	
Financial Review	1			✓	
Journal of Financial Markets	1			✓	
Journal of Financial Research	1			✓	
Journal of Financial Services Research	1			✓	
Journal of Financial Stability	1			✓	✓
Journal of Money, Credit and Banking	1		✓		
Journal of Real Estate Finance and Economics	1			✓	
Mathematical Finance	1			✓	
Annual Review of Financial Economics	0			✓	
Corporate Governance: An International Review	0			✓	
Financial Analysts Journal	0			✓	
Financial Management	0			✓	

(continued)

Table 24.1 (continued)

Journal	#EML Articles	FT50	ABS 4 *	ABS 3	Google Metrics (Top20)
Financial Markets, Institutions and Instruments	0			✓	
International Journal of Central Banking	0			✓	
Journal of Accounting and Economics	0				✓
Journal of Commodity Markets	0			✓	
Journal of Corporate Finance	0		✓		✓
Journal of Financial Intermediation	0		✓		
Review of Accounting Studies	0				✓
Review of Asset Pricing Studies	0			✓	
Review of Corporate Finance Studies	0			✓	
Review of Quantitative Finance and Accounting	0			✓	
Journal of Portfolio Management	na				✓

Using each journals' online search interface, we used the following search terms to capture instances of evolutionary machine learning:

- evolutionary machine learning,
- evolutionary computation,
- evolutionary algorithm,
- genetic algorithm,
- evolution strategy,
- genetic programming,
- differential evolution.

We have uncovered 199 papers over a period of time ranging from 1997 to 2022. The number of articles found in each journal is provided in Table 24.1 and Fig. 24.1 reports the number of articles published each year. There is a trend of increasing publication activity using evolutionary algorithms over the period of time, in particular, there is a marked increase in frequency since 2010, and in particular in the past 4 years since 2019. 14 of the journals examined did not have any papers published that contain evolutionary machine learning. 10 of these belong to the ABS3 list, with 2 from the ABS4 list, and the remaining 2 from the Google Metrics list. All of the FT50/ABS4* journals examined have published articles related to evolutionary machine learning.

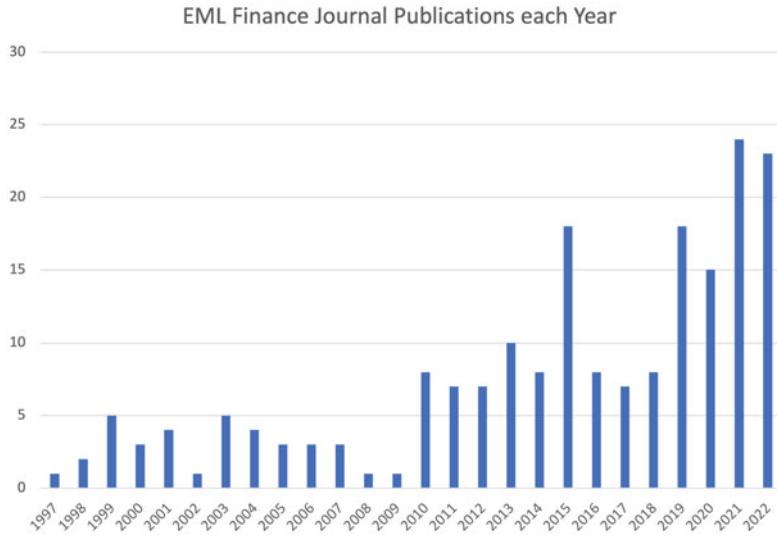


Fig. 24.1 Using the search terms provided in Sect. 24.2 we have uncovered articles over a period of time starting in 1997 ranging up to the present day. The number of articles on evolutionary algorithms published each year

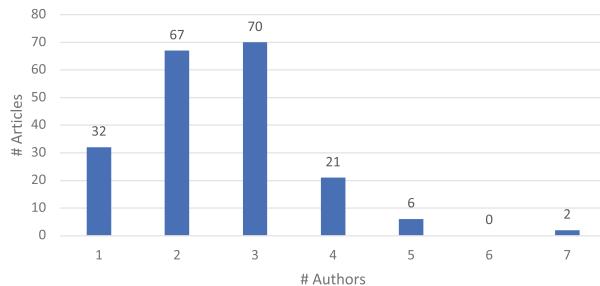


Fig. 24.2 We count the number of articles for the size of each set of authors. The majority of articles have two to three authors, with two articles having seven authors

In the 199 articles, we count 446 distinct authors, 36 of whom have published more than one article. Figure 24.2 captures the number of articles with multiple authors. The majority of articles have between two and three authors. Two articles have seven authors. Table 24.2 lists the top 12 authors who have published 3 or more articles related to evolutionary machine learning in finance.

Figure 24.3 presents a heatmap of the number of articles published each year in every journal examined. We have clustered the journals into three groups based on their relative prestige. The most prestigious journals for faculty in finance are more often considered to be those in the FT50, ABS4* and ABS4 lists. The next most prestigious would be the ABS3, the remaining journals in the Google Metrics Top

Table 24.2 There are 36 authors who have published more than one article related to EML in Finance. Here we present authors who have published three or more articles

Author	#Articles
Manahov, Viktor	7
Neely, Christopher J.	6
Dunis, Christian L.	5
Hudson, Robert	4
Shapiro, Arnold F.	4
Weller, Paul A.	4
Karathanasopoulos, Andreas	3
Ladley, Daniel	3
Laws, Jason	3
Sermpinis, Georgios	3
Wang, Donghua	3
Wang, J.	3

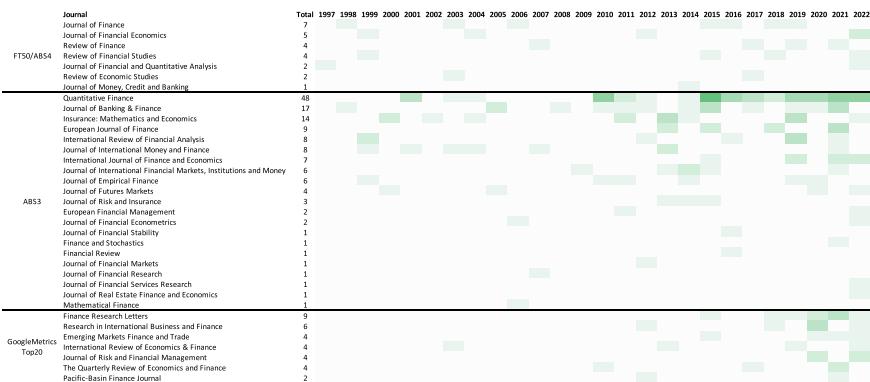


Fig. 24.3 A heatmap of the number of articles published in each journal each year. The journals are clustered by perceived quality from top to bottom, with the FT50, ABS4*, and ABS3 clustered as the top ranking journals, followed by ABS3, with the final cluster being the remaining journals captured by Google Metrics Top 20

20 list are grouped into the third cluster. A total of 25 articles (13% of the total captured in our analysis) have appeared in the most prestigious cluster. While there are a relatively smaller number of articles in this group they are spread out over time capturing both the earliest (1997) and most recent (2022) publications relating to EML in finance. Up to 2010 there are a total of 9 articles in this first group, with the remaining 16 articles appearing in the last decade.

142 articles appear in the second cluster of journals, which is also the largest cluster, and represents 71% of articles captured in our analysis. By far the most articles (48 equating to 24% of the total) appear in the journal Quantitative Finance.

A total of 33 articles (or 17%) appear in the third cluster of journals. Note there is an overlap between some journals that appear in the Google Metrics list and those that appear in the first (5) and second (5) clusters, and as such the third cluster represents articles that appear exclusively in the Google Metrics Top 20. In contrast to the first clusters relatively even spread of articles over time, the third cluster articles all appear in the last decade (with the exception of 2, which were published earlier).

24.3 Topic analysis—Applications in Finance

We examine the topics of focus to which evolutionary machine learning has been applied to in the finance literature. We manually determine the area of application of each paper, and count the number of occurrences. We uncovered just over 40 topics.

The first article to appear in the literature in 1997 is focussed on technical analysis of foreign exchange markets using genetic programming [24]. The most recent articles appearing in 2022 focus on 14 distinct topics, including markets [144, 187, 211], risk [88, 154, 212], futures [47, 159], insurance [68, 138], portfolios [146, 205], trading [23, 99], cryptocurrencies [102], pricing [35], derivatives [196], loans [29], real estate [162], financial systems [95], mergers and acquisitions [157], and venture capital [27].

Figure 24.4 provides a word cloud representation of the different topics, and Fig. 24.5 presents a topic heatmap. From these figures, we observe that trading, markets, portfolios, pricing and foreign exchange are the top five topics, and repre-

Fig. 24.4 A word cloud of the key words representing the application area of each paper examined



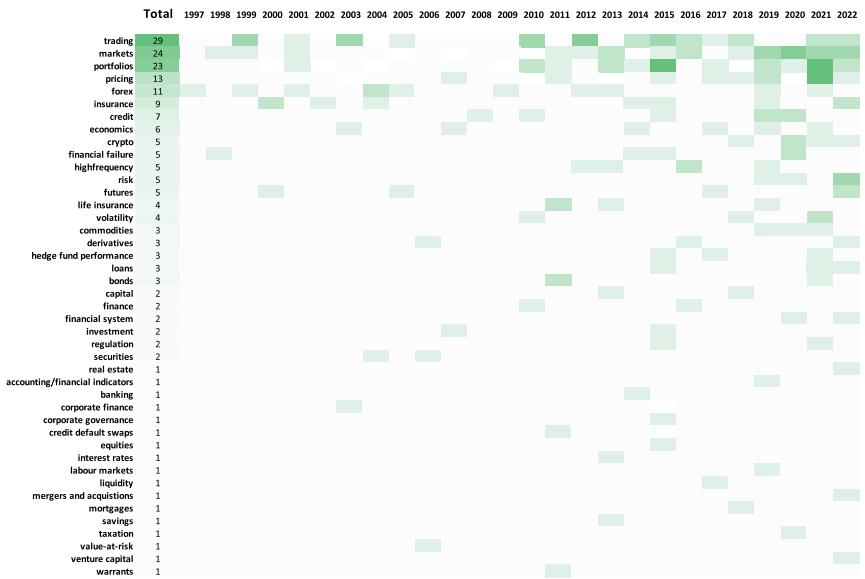


Fig. 24.5 A heatmap of the topic analysis of the papers uncovered, which focusses on the application area of each study. Over 40 different application areas are found

sent just over 50% of the papers found. Other topics range from mortgages, taxation, labour markets, mergers and acquisitions to securities, derivatives, real estate and cryptocurrencies to name a few.

Table 24.3 provides a complete mapping of articles to topics.

24.4 Discussion

Aziz et al. [113] recently undertook a topic analysis of machine learning in finance. The method adopted was to use the Elsevier Scopus database focussing on the subject areas of

1. computer science;
2. business, management and accounting;
3. decision sciences;
4. economics;
5. econometrics and finance.

The publication types included both conferences and journals, and unlike this study do not focus exclusively on the mainstream finance literature. Four main categories of topics were observed in their analysis, which were Price Forecasting, Financial Market Analysis, Risk Forecasting and Financial Perspectives (which include

economic development). These represent an aggregation of 15 distinct topics. Interestingly research on GP and PSO was identified under their Price Forecasting topic, even though evolutionary machine learning and any of the keywords we adopted in this study were not one of the keywords used in their literature search protocol.

Our focus on evolutionary machine learning in the mainstream finance literature has uncovered a much wider set of application areas including taxation, insurance, economics, mergers and acquisitions, cryptocurrencies, etc. See Fig. 24.5 for the full list. In terms of the numbers of articles, a similar trend over time is observed with a greater uptake of (evolutionary) machine learning in finance this century, in particular this past decade since around 2010.

Brabazon et al. [14] in a recent analysis of the application of genetic programming to finance found the method applied in forecasting, trading, portfolio construction, derivatives, solvency and agent-based modelling. Also noting that much of these publications appeared in the computer science, rather than finance literature. As can be seen in this study we have uncovered that the wider set of methods encompassing evolutionary machine learning has been applied to a much more extensive set of topics in the mainstream finance literature.

Similar to Brabazon et al. [14], Aziz et al. [113] also note that the uptake of ML in the finance literature is significantly exceeded by the multiples of research published in the ICT (i.e. method development) literature.

24.5 Summary and Conclusions

We set out to analyse the mainstream finance literature to determine the uptake of evolutionary machine learning in that community of scholars. We see a marked uptick in the application to finance since 2010, and, in particular, a burst of activity in the past 4 years since 2019. This activity covers 199 articles, with over 400 authors, and a broad set of over 40 different topic areas within finance. Articles are appearing in the most prestigious journals, with the most frequent activity in the mid-rank journals captured in the ABS3 list, in particular, Quantitative Finance is the journal most frequently publishing articles related to evolutionary machine learning. The findings of this study bode well for our community, demonstrating a good deal of crossover into the academic literature in finance. At the very least, this presents an opportunity to bring the latest advances in evolutionary machine learning method development to the finance community.

Table 24.3 A mapping of the articles uncovered in our search to topics in finance

Topic	Articles
Trading	[25, 53, 83, 85, 90, 150, 177, 183, 188, 194]
	[28, 62, 76, 97, 108–110, 121, 147, 184]
	[23, 99, 103, 115, 142, 167, 176, 200, 213]
Markets	[16, 20, 34, 72, 78, 116, 118, 169, 190, 201, 207, 208]
	[21, 42, 58, 77, 119, 120, 144, 187, 189, 192, 210, 211]
Portfolios	[48, 104, 127, 129, 131, 141, 175, 178, 181, 202–204]
	[41, 43, 45, 64, 71, 106, 143, 146, 199, 205, 214]
Pricing	[35, 56, 63, 67, 105, 122, 124, 139, 156, 170, 186, 209]
Forex	[24, 59, 66, 75, 82, 86, 87, 111, 155, 195, 197]
Insurance	[68, 126, 128, 130, 133, 136, 138, 163, 165]
Credit	[44, 55, 57, 60, 61, 145, 193]
Economics	[38, 39, 80, 81, 151, 152]
Crypto	[46, 69, 70, 73, 102]
Financial failure	[51, 101, 173, 180, 198]
Futures	[47, 159–161, 179]
High frequency	[52, 79, 107, 125, 140]
Risk	[88, 148, 154, 172, 212]
Life insurance	[132, 134, 135, 137]
Volatility	[49, 123, 182, 185]
Commodities	[74, 94, 158]
Derivatives	[92, 166, 196]
Hedge fund performance	[30, 32, 171]
Loans	[19, 29, 98]
Bonds	[40, 65, 117]
Capital	[22, 84]
Finance	[168, 206]
Financial system	[95, 96]
Investment	[31, 36]
Regulation	[50, 112]
Securities	[17, 26]
Accounting/financial indicators	[91]
Banking	[54]
Corporate finance	[18]
Corporate governance	[93]
Credit default swaps	[114]
Equities	[191]

(continued)

Table 24.3 (continued)

Topic	Articles
Interest rates	[89]
Labour markets	[33]
Liquidity	[174]
Mergers and acquisitions	[157]
Mortgages	[37]
Real estate	[162]
Savings	[164]
Taxation	[100]
Value-at-risk	[153]
Venture capital	[27]
Warrants	[149]

References

1. Telikani, A., Tahmassebi, A., Banzhaf, W., Gandomi, A.H.: Evolutionary machine learning: a survey. *ACM Comput. Surv.* **54**(8), Article 161, 35 (2021)
2. EvoWorkshops, G., et al. (ed.): Proceedings of Applications of evolutionary computing : EvoWorkshops 2007, EvoCOMNET, EvoFIN, EvoIASP, EvoINTERACTION, Evo-MUSART, EvoSTOC and EvoTRANSLOG, Valencia, Spain, April 11–13, 2007. Lecture Notes in Computer Science, vol. 4448. Springer (2007)
3. Chen, S.-H. (ed.): Evolutionary Computation in Economics and Finance. Physica-Verlag (2002)
4. Chen, S.-H. (ed.): Genetic Algorithms and Genetic Programming in Computational Finance. Springer (2002)
5. Chen, S.-H., Wang, P.P. (eds.): Computational Intelligence in Economics and Finance. Springer (2003)
6. Rennard, J.-P. (ed.): Handbook of Research on Nature-Inspired Computing for Economics and Management. IGI Global (2006)
7. Brabazon, A., O'Neill, M.: Biologically Inspired Algorithms for Financial Modelling. Springer
8. Chen, S.-H., Wang, P.P., Kuo, T.-W. (eds.): Computational Intelligence in Economics and Finance, vol. 2. Springer (2007)
9. Brabazon, A., O'Neill, M. (eds.): Natural Computing in Computational Finance. Springer (2008)
10. Brabazon, A., O'Neill, M. (eds.): Natural Computing in Computational Finance, (Volume 2). Springer (2009)
11. Brabazon, A., O'Neill, M., Maringer, D. (eds.): Natural Computing in Computational Finance (Volume 3). Springer (2010)
12. Brabazon, A., O'Neill, M., Maringer, D. (eds.): Natural Computing in Computational Finance (Volume 4). Springer (2011)
13. Chen, S.-H., Kaboudan, M., Du, Y.-R. (eds.): The Oxford Handbook of Computational Economics and Finance. Oxford University Press (2018)
14. Brabazon, A., Kampouridis, M., O'Neill, M.: Applications of genetic programming to finance and economics: past, present, future. *Genet. Program. Evolvable Mach.* **21**(1–2), 33–53 (2020)

15. Iba, H., Aranha, C.C.: Practical Applications of Evolutionary Computation to Financial Engineering: Robust Techniques for Forecasting, Trading and Hedging. Springer-Verlag, New York Inc (2012). ISBN-10: 3642276474
16. Brown, S.J., Goetzmann, W.N., Kumar, A.: The Dow theory: William Peter Hamilton's track record reconsidered. *J. Financ.* **53**, 1311–1333 (1998)
17. Noe, T.H., Rebello, M.J., Wang, J.: The evolution of security designs. *J. Financ.* **61**, 2103–2135 (2006)
18. Noe, T.H., Rebello, M.J., Wang, J.: Corporate financing: an artificial agent-based analysis. *J. Financ.* **58**, 943–973 (2003)
19. Garmaise, M.J.: Borrower misreporting and loan performance. *J. Financ.* **70**, 449–484 (2015)
20. Cieslak, A., Povala, P.: Information in the term structure of yield curve volatility. *J. Financ.* **71**, 1393–1436 (2016)
21. Schneider, P., Trojani, F.: (Almost) model-free recovery. *J. Financ.* **74**, 323–370 (2019)
22. Schwert, M.: Bank capital and lending relationships. *J. Financ.* **73**, 787–830 (2018)
23. Brogaard, J., Zareei, A.: Machine learning and the stock market. *J. Financ. Quant. Anal.* **1–66** (2022). <https://doi.org/10.1017/S0022109022001120>
24. Neely, C., Weller, P., Dittmar, R.: Is technical analysis in the foreign exchange market profitable? a genetic programming approach. *J. Financ. Quant. Anal.* **32**(4), 405–426 (1997). <https://doi.org/10.2307/2331231>
25. Allen, F., Karjalainen, R.: Using genetic algorithms to find technical trading rules. *J. Financ. Econ.* **51**(2), 245–271 (1999)
26. Cummins, J.D., Lalonde, D., Phillips, R.D.: The basis risk of catastrophic-loss index securities. *J. Financ. Econ.* **71**(1), 77–111 (2004)
27. Ewens, M., Gorbenko, A., Korteweg, A.: Venture capital contracts. *J. Financ. Econ.* **143**(1), 131–158 (2022)
28. Bajgrowicz, P., Scaillet, O.: Technical trading revisited: false discoveries, persistence tests, and transaction costs. *J. Financ. Econ.* **106**(3), 473–491 (2012)
29. Craig, B., Ma, Y.: Intermediation in the interbank lending market. *J. Financ. Econ.* **145**(2), Part A, 179–207 (2022)
30. Joenväärä, J., Kosowski, R.: The effect of regulatory constraints on fund performance: new evidence from UCITS hedge funds. *Rev. Financ.* **25**(1), 189–233 (2021)
31. Lensberg, T., Schenk-Hoppé, K.R.: On the evolution of investment strategies and the Kelly rule-A Darwinian approach. *Rev. Financ.* **11**(1), 25–50 (2007)
32. O'Doherty, M.S., Savin, N.E., Tiwari, A.: Hedge fund replication: a model combination approach. *Rev. Financ.* **21**(4), 1767–1804 (2017)
33. Michaels, R., Beau Page, T., Whited, T.M.: Labor and capital dynamics under financing frictions. *Rev. Financ.* **23**(2), 279–323 (2019)
34. Routledge, B.R.: Adaptive learning in financial markets. *Rev. Financ. Stud.* **12**(5), 1165–1202 (1999)
35. Patton, A.J., Weller, B.M.: Risk price variation: the missing half of empirical asset pricing. *Rev. Financ. Stud.* (2022). hhac012
36. Payzan-LeNestour, E., Bossaerts, P.: Learning about unstable, publicly unobservable payoffs. *Rev. Financ. Stud.* **28**(7), 1874–1913 (2015)
37. Chernov, M., Dunn, B.R., Longstaff, F.A.: Macroeconomic-driven prepayment risk and the valuation of mortgage-backed securities. *Rev. Financ. Stud.* **31**(3), 1132–1183 (2018)
38. Adam, K.: Learning and equilibrium selection in a monetary overlapping generations model with sticky prices. *Rev. Econ. Stud.* **70**(4), 887–907 (2003)
39. Qu, Z., Tkachenko, D.: Global identification in DSGE models allowing for indeterminacy. *Rev. Econ. Stud.* **84**(3), 1306–1345 (2017)
40. Dew-Becker, I.: Bond pricing with a time-varying price of risk in an estimated medium-scale Bayesian DSGE model. *J. Money Credit. Bank.* **46**, 837–888 (2014)
41. Drenovak, M., Ranković, V., Urošević, B., Jelic, R.: Mean-maximum drawdown optimization of buy-and-hold portfolios using a multi-objective evolutionary algorithm. *Financ. Res. Lett.* **46**, Part A, 102328 (2021)

42. Baek, S., Mohanty, S.K., Glambosky, M.: COVID-19 and stock market volatility: an industry level analysis. *Financ. Res. Lett.* **37**(2020), 101748 (2020)
43. Gong, X.-L., Xiong, X.: Multi-objective portfolio optimization under tempered stable Lévy distribution with Copula dependence. *Financ. Res. Lett.* **38**, 101506 (2021)
44. Yu, L., Zhang, X.: Can small sample dataset be used for efficient internet loan credit risk assessment? Evidence from online peer to peer lending. *Financ. Res. Lett.* **38**, 101521 (2020)
45. Kamali, R., Mahmoodi, S., Jahandideh, M.-T.: Optimization of multi-period portfolio model after fitting best distribution. *Financ. Res. Lett.* **30**, 44–50 (2019)
46. Geuder, J., Kinateder, H., Wagner, N.F.: Cryptocurrencies as financial bubbles: The case of Bitcoin. *Financ. Res. Lett.* **31**, 2019 (2018)
47. Ma, C., Xiao, R., Mi, X.: Measuring the dynamic lead-lag relationship between the cash market and stock index futures market. *Financ. Res. Lett.* **47**, Part B, 102940 (2022)
48. Boudt, K., Lu, W., Peeters, B.: Higher order comoments of multifactor models and asset allocation. *Financ. Res. Lett.* **13**, 225–233 (2015)
49. Shi, Y.: A closed-form estimator for the Markov switching in mean model. *Financ. Res. Lett.* **44**(2022), 102107 (2021)
50. Lensberg, T., Schenk-Hoppé, K.R., Ladley, D.: Costs and benefits of financial regulation: short-selling bans and transaction taxes. *J. Bank. & Financ.* **51**, 103–118 (2015)
51. Varetto, F.: Genetic algorithms applications in the analysis of insolvency risk. *J. Bank. & Financ.* **22**(10–11), 1421–1439 (1998)
52. Chavez-Demoulin, V., McGill, J.A.: High-frequency financial data modeling using Hawkes processes. *J. Bank. & Financ.* **36**(12), 3415–3426 (2012)
53. Marshall, B.R., Young, M.R., Rose, L.C.: Candlestick technical trading strategies: can they create value for investors?. *J. Bank. & Financ.* **30**(8), 2303–2323 (2005)
54. Langfield, S., Liu, Z., Ota, T.: Mapping the UK interbank system. *J. Bank. & Financ.* **45**, 288–303 (2014)
55. Krink, T., Paterlini, S., Resti, A.: The optimal structure of PD buckets. *J. Bank. & Financ.* **32**(10), 2275–2286 (2008)
56. Leippold, M., Schärer, S.: Discrete-time option pricing with stochastic liquidity. *J. Bank. & Financ.* **75**, 1–16 (2017)
57. Jones, S., Johnstone, D., Wilson, R.: An empirical evaluation of the performance of binary classifiers in the prediction of credit ratings changes. *J. Bank. & Financ.* **56**, 72–85 (2015)
58. Chen, J., Jiang, G.J., Yuan, C., Zhu, D.: Breaking VIX at open: evidence of uncertainty creation and resolution. *J. Bank. & Financ.* **124**, 106060 (2021)
59. Sarantis, N.: On the short-term predictability of exchange rates: a BVAR time-varying parameters approach. *J. Bank. & Financ.* **30**(8), 2257–2279 (2005)
60. Khandani, A.E., Kim, A.J., Lo, A.W.: Consumer credit-risk models via machine-learning algorithms. *J. Bank. & Financ.* **34**(11), 2767–2787 (2010)
61. Fricke, D., Roukny, T.: Generalists and specialists in the credit market. *J. Bank. & Financ.* **112**, 105335 (2020)
62. Chiarella, C., Ladley, D.: Chasing trends at the micro-level: the effect of technical trading on order book dynamics. *J. Bank. & Financ.* **72**(Supplement), S119–S131 (2016)
63. Baule, R., Shkel, D.: Model risk and model choice in the case of barrier options and bonus certificates. *J. Bank. & Financ.* **133**, 106307 (2021)
64. Packham, N., Woebbeking, C.F.: A factor-model approach for correlation scenarios and correlation stress testing. *J. Bank. & Financ.* **101**, 92–103 (2019)
65. Chen, X.H., Maringer, D.: Detecting time-variation in corporate bond index returns: a smooth transition regression model. *J. Bank. & Financ.* **35**(1), 95–103 (2011)
66. Branger, N., Herold, M., Muck, M.: International stochastic discount factors and covariance risk. *J. Bank. & Financ.* **123**, 106018 (2021)
67. Dbouk, W., Jamali, I.: Predicting daily oil prices: linear and non-linear models. *Res. Int. Bus. Financ.* **46**, 149–165 (2018)
68. Aslam, F., Hunjra, A.I., Ftiti, Z., Louhichi, W., Shams, T.: Insurance fraud detection: evidence from artificial intelligence and machine learning. *Res. Int. Bus. Financ.* **62**, 101744 (2022)

69. Kyriazis, N., Papadamou, S., Corbet, S.: A systematic review of the bubble dynamics of cryptocurrency prices. *Res. Int. Bus. Financ.* **54**, 101254 (2020)
70. Chu, J., Chan, S., Zhang, Y.: High frequency momentum trading with cryptocurrencies. *Res. Int. Bus. Financ.* **52**, 101176 (2020)
71. Li, W., Mei, F.: Asset returns in deep learning methods: an empirical analysis on SSE 50 and CSI 300. *Res. Int. Bus. Financ.* **54**, 101291 (2020)
72. Chen, S.-H., Chang, C.-L., Tseng, Y.-H.: Social networks, social interaction and macroeconomic dynamics: how much could Ernst Ising help DSGE?. *Res. Int. Bus. Financ.* **30**(2014), 312–335 (2012)
73. Manahov, V., Urquhart, A.: The efficiency of Bitcoin: a strongly typed genetic programming approach to smart electronic Bitcoin markets. *Int. Rev. Financ. Anal.* **73**, 101629 (2021)
74. Zhang, Y.-J., Lin, J.-J.: Can the VAR model outperform MRS model for asset allocation in commodity market under different risk preferences of investors?. *Int. Rev. Financ. Anal.* **66**, 101395 (2019)
75. El Shazly, M.R., El Shazly, H.E.: Forecasting currency prices using a genetically evolved neural network architecture. *Int. Rev. Financ. Anal.* **8**(1), 67–82 (1999)
76. Kampouridis, M., Chen, S.-H., Tsang, E.: Market fraction hypothesis: a proposed test. *Int. Rev. Financ. Anal.* **23**, 41–54 (2012)
77. He, X.-Z., Li, Y., Zheng, M.: Heterogeneous agent models in financial markets: a nonlinear dynamics approach. *Int. Rev. Financ. Anal.* **62**, 135–149 (2019)
78. Zhang, X., Zhang, Q., Chen, D., Gu, J.: Financial integration, investor protection and imbalanced optimistically biased information timeliness in emerging markets. *Int. Rev. Financ. Anal.* **64**, 38–56 (2019)
79. Manahov, V.: A note on the relationship between high-frequency trading and latency arbitrage. *Int. Rev. Financ. Anal.* **47**, 281–296 (2016)
80. Diaz, E.M., Perez-Quiros, G.: GEA tracker: a daily indicator of global economic activity. *J. Int. Money Financ.* **115**, 102400 (2021)
81. Lim, G.C., McNelis, P.D.: Central bank learning, terms of trade shocks and currency risk: should only inflation matter for monetary policy?. *J. Int. Money Financ.* **26**(6), 865–886 (2007)
82. Marey, P.S.: Exchange rate expectations: controlled experiments with artificial traders. *J. Int. Money Financ.* **23**(2), 283–304 (2004)
83. Neely, C.J., Weller, P.A.: Intraday technical trading in the foreign exchange market. *J. Int. Money Financ.* **22**(2), 223–237 (2003)
84. Lai, J.T., McNelis, P.D., Yan, I.K.M.: Regional capital mobility in China: economic reform with limited financial integration. *J. Int. Money Financ.* **37**, 493–503 (2013)
85. Neely, C.J., Weller, P.A.: Technical trading rules in the European monetary system. *J. Int. Money Financ.* **18**(3), 429–458 (1999)
86. De Grauwe, P., Markiewicz, A.: Learning to forecast the exchange rate: two competing approaches. *J. Int. Money Financ.* **32**, 42–76 (2013)
87. Neely, C.J., Weller, P.A.: Technical analysis and central bank intervention. *J. Int. Money Financ.* **20**(7), 949–970 (2001)
88. Sant'Anna, L.R., Righi, M.B., Müller, F.M., Guedes, P.C.: Risk measure index tracking model. *Int. Rev. Econ. & Financ.* **80**, 361–383 (2022)
89. Fernandez-Perez, A., Fernández-Rodríguez, F., Sosvilla-Rivero, S.: The term structure of interest rates as predictor of stock returns: evidence for the IBEX 35 during a bear market. *Int. Rev. Econ. & Financ.* **31**, 21–33 (2013)
90. Neely, C.J.: Risk-adjusted, ex ante, optimal technical trading rules in equity markets. *Int. Rev. Econ. & Financ.* **12**(1), 69–87 (2003)
91. Wanke, P., Azad, M.A.K., Emrouznejad, A., Antunes, J.: A dynamic network DEA model for accounting and financial indicators: a case of efficiency in MENA banking. *Int. Rev. Econ. & Financ.* **61**, 52–68 (2019)
92. Heath, A., Kelly, G., Manning, M., Markose, S., Shaghaghi, A.R.: CCPs and network stability in OTC derivatives markets. *J. Financ. Stab.* **27**, 217–233 (2016)

93. Li, Chien-Kuo., Liang, Deron, Lin, Fengyi, Chen, Kwo-Liang.: The application of corporate governance indicators with XBRL technology to financial crisis prediction. *Emerg. Mark. Financ. Trade.* **51**(sup1), S58–S72 (2015)
94. Xu, X., Wang, C., Li, J., Shi, C.: Green transportation and information uncertainty in gasoline distribution: evidence from China. *Emerg. Mark. Financ. Trade.* **57**(11), 3101–3119 (2021)
95. Wang, T., Zhao, S., Wang, W., Yang, H.: How does exogenous shock change the structure of interbank network?: evidence from China under COVID-19. *Emerg. Mark. Financ. Trade* (2022)
96. Zhao, W., Lu, Y., Zhao, M., Zhang, P.: Fluctuations in the open economy of China: evidence from the ABNK model. *Emerg. Mark. Financ. Trade.* **56**(9), 2073–2092 (2020)
97. Tirapat, S., Visaltanachoti, N.: Opportunistic insider trading. *Pac. Basin Financ. J.* **21**(1), 1046–1061 (2012). ISSN 0927-538X
98. Wang, L., Su, Z.-Q., Fung, H.-G., Jin, H.-M., Xiao, Z.: Do CEOs with academic experience add value to firms? Evidence on bank loans from Chinese firms. *Pac. Basin Financ. J.* **67**, 101534 (2021)
99. Abraham, R., Samad, M.E., Bakhach, A.M., El-Chaarani, H., Sardouk, A., Nemar, S.E., Jaber, D.: Forecasting a stock trend using genetic algorithm and random forest. *J. Risk Financ. Manag.* **15**(5), 188 (2022)
100. Małecka-Ziemińska, E., Ziemiński, R.: Application of genetic algorithm to optimal income taxation. *J. Risk Financ. Manag.* **13**(11), 251 (2020)
101. Pisula, T.: An ensemble classifier-based scoring model for predicting bankruptcy of polish companies in the Podkarpackie Voivodeship. *J. Risk Financ. Manag.* **13**(2), 37 (2020)
102. Mba, J.C., Mai, M.M.: A particle swarm optimization copula-based approach with application to cryptocurrency portfolio optimisation. *J. Risk Financ. Manag.* **15**(7), 285 (2022)
103. Nazário, R.T.F., e Silva, L.J., Sobreiro, V.A., Kimura, H.: A literature review of technical analysis on stock markets. *Q. Rev. Econ. Financ.* **66**, 115–126 (2017)
104. Mansourfar, G., Mohamad, S., Hassan, T.: The behavior of MENA oil and non-oil producing countries in international portfolio optimization. *Q. Rev. Econ. Financ.* **50**(4), 415–423 (2010)
105. Jiang, M., Liu, J., Zhang, L.: An extended regularized Kalman filter based on Genetic Algorithm: Application to dynamic asset pricing models. *Q. Rev. Econ. Financ.* **79**, 28–44 (2021)
106. Zhuo, J., Li, X., Yu, C.: Parameter behavioral finance model of investor groups based on statistical approaches. *Q. Rev. Econ. Financ.* **80**, 74–79 (2021)
107. Manahov, V., Hudson, R., Gebka, B.: Does high frequency trading affect technical analysis and market efficiency? And if so, how?. *J. Int. Financ. Mark. Inst. Money.* **28**(2014), 131–157 (2013)
108. Sermpinis, G., Stasinakis, C., Dunis, C.: Stochastic and genetic neural network combinations in trading and hybrid time-varying leverage effects. *J. Int. Financ. Mark. Inst. Money.* **30**, 21–54 (2014)
109. Manahov, V., Hudson, R., Hoque, H.: Return predictability and the ‘wisdom of crowds’: Genetic Programming trading algorithms, the Marginal Trader Hypothesis and the Hayek Hypothesis. *J. Int. Financ. Mark. Inst. Money.* **37**, 85–98 (2015)
110. Manahov, V., Hudson, R., Linsley, P.: New evidence about the profitability of small and large stocks and the role of volume obtained using Strongly Typed Genetic Programming. *J. Int. Financ. Mark. Inst. Money.* **33**, 299–316 (2014)
111. Neely, C.J.: Forecasting foreign exchange volatility: Why is implied volatility biased and inefficient? And does it matter?. *J. Int. Financ. Mark. Inst. Money* **19**(1), 188–205 (2009)
112. Polyzos, S., Samitas, A., Kampouris, I.: Economic stimulus through bank regulation: Government responses to the COVID-19 crisis. *J. Int. Financ. Mark. Inst. Money.* **75**, 101444 (2021)
113. Aziz, S., Dowling, M., Hammami, H., Piepenbrink, A.: Machine learning in finance: a topic modeling approach. *Eur. Financ. Manag.* **28**, 744–770 (2022)
114. Bedendo, M., Cathcart, L., El-Jahel, L.: Market and model credit default swap spreads: mind the gap!. *Eur. Financ. Manag.* **17**, 655–678 (2011)

115. Franco, M., Vivo, J.-M.: Genetic algorithms for parameter estimation in modelling of index returns. *Eur. J. Financ.* **24**(13), 1088–1099 (2018)
116. Dunis, C.L., Laws, J., Karathanasopoulos, A.: GP algorithm versus hybrid and mixed neural networks. *Eur. J. Financ.* **19**(3), 180–205 (2013)
117. Drenovak, M., Ranković, V., Urošević, B., Jelic, R.: Bond portfolio management under Solvency II regulation. *Eur. J. Financ.* **27**(9), 857–879 (2021)
118. Geraskin, P., Fantazzini, D.: Everything you always wanted to know about log-periodic power laws for bubble modeling but were afraid to ask. *Eur. J. Financ.* **19**(5), 366–391 (2013)
119. Zhang, J., Wen, J., Chen, J.: Modeling market fluctuations under investor sentiment with a Hawkes-Contact process. *Eur. J. Financ.* (2021)
120. Gao, X., Ladley, D.: Noise trading and market stability. *Eur. J. Financ.* (2021)
121. Dunis, C.L., Laws, J., Middleton, P.W., Karathanasopoulos, A.: Trading and hedging the corn/ethanol crush spread using time-varying leverage and nonlinear models. *Eur. J. Financ.* **21**(4), 352–375 (2015)
122. Sermepinis, G., Laws, J., Dunis, C.L.: Modelling commodity value at risk with Psi Sigma neural networks using open-high-low-close data. *Eur. J. Financ.* **21**(4), 316–336 (2015)
123. Luo, J., Chen, L.: Volatility dependences of stock markets with structural breaks. *Eur. J. Financ.* **24**(17), 1727–1753 (2018)
124. Carr, P., Torricelli, L.: Additive logistic processes in option pricing. *Financ. Stoch.* **25**, 689–724 (2021)
125. Manahov, V.: Front-running scalping strategies and market manipulation: why does high-frequency trading need stricter regulation?. *Financ. Rev.* **51**, 363–402 (2016)
126. Shapiro, A.F.: A Hitchhiker's guide to the techniques of adaptive nonlinear models. *Insur. Math. Econ.* **26**(2–3), 119–132 (2000)
127. Gupta, P., Mittal, G., Mehlawat, M.K.: Expected value multiobjective portfolio rebalancing model with fuzzy parameters. *Insur. Math. Econ.* **52**(2), 190–203 (2013)
128. Shapiro, A.F.: Fuzzy logic in insurance. *Insur. Math. Econ.* **35**(2), 399–424 (2004)
129. Liu, Y.-J., Zhang, W.-G.: Fuzzy portfolio optimization model under real constraints. *Insur. Math. Econ.* **53**(3), 704–711 (2013)
130. Shapiro, A.F., Gorman, R.P.: Implementing adaptive nonlinear models. *Insur. Math. Econ.* **26**(2–3), 289–307 (2000)
131. Huang, X., Zhao, T.: Mean-chance model for portfolio selection based on uncertain measure. *Insur. Math. Econ.* **59**, 243–250 (2014)
132. Graf, S., Kling, A., Ruß, J.: Risk analysis and valuation of life insurance contracts: Combining actuarial and financial approaches. *Insur. Math. Econ.* **49**(1), 115–125 (2011)
133. Shapiro, A.F.: The merging of neural networks, fuzzy logic, and genetic algorithms. *Insur. Math. Econ.* **31**(1), 115–131 (2002)
134. Jevtić, P., Luciano, E., Vigna, E.: Mortality surface by means of continuous time cohort models. *Insur. Math. Econ.* **53**(1), 122–133 (2013)
135. Zemp, A.: Risk comparison of different bonus distribution approaches in participating life insurance. *Insur. Math. Econ.* **49**(2), 249–264 (2011)
136. Blostein, M., Miljkovic, T.: On modeling left-truncated loss data using mixtures of distributions. *Insur. Math. Econ.* **85**, 35–46 (2019)
137. Jevtić, P., Regis, L.: A continuous-time stochastic model for the mortality surface of multiple populations. *Insur. Math. Econ.* **88**, 181–195 (2019)
138. Zhou, H., Zhou, K.Q., Li, X.: Stochastic mortality dynamics driven by mixed fractional Brownian motion. *Insur. Math. Econ.* **106**, 218–238 (2022)
139. Beer, S., Braun, A., Marugg, A.: Pricing industry loss warranties in a Lévy-Frailty framework. *Insur. Math. Econ.* **89**, 171–181 (2019)
140. Manahov, V., Hudson, R., Urquhart, A.: High-frequency trading from an evolutionary perspective: financial markets as adaptive systems. *Int. J. Financ. Econ.* **2019**(24), 943–962 (2019)

141. Loukeris, N., Eleftheriadis, I.: Further Higher Moments in Portfolio Selection and A Priori Detection of Bankruptcy, Under Multi-layer Perceptron Neural Networks, Hybrid Neurogenetic MLPs, and the Voted Perceptron. *Int. J. Financ. Econ.* **20**, 341–361 (2015). <https://doi.org/10.1002/ijfe.1521>
142. Moradi, M., Jabbari Nooghabi, M., Rounaghi, M.M.: Investigation of fractal market hypothesis and forecasting time series stock returns for Tehran Stock Exchange and London Stock Exchange. *Int. J. Financ. Econ.* **26**, 662–678 (2021)
143. Mahmoudi, A., Hashemi, L., Jasemi, M., Pope, J.: A comparison on particle swarm optimization and genetic algorithm performances in deriving the efficient frontier of stocks portfolios based on a mean-lower partial moment model. *Int. J. Financ. Econ.* **26**, 5659–5665 (2021)
144. Yang, X., Zhang, C., Yang, Y., Wang, W., Wagan, Z.A.: A new risk measurement method for China's carbon market. *Int. J. Financ. Econ.* **27**, 1280–1290 (2022)
145. Xu, D., Zhang, X., Feng, H.: Generalized fuzzy soft sets theory-based novel hybrid ensemble credit scoring model. *Int. J. Financ. Econ.* **24**, 903–921 (2019)
146. Arreola Hernandez, J., Kang, S.H., Yoon, S.-M.: Interdependence and portfolio optimisation of bank equity returns from developed and emerging Europe. *Int. J. Financ. Econ.* **27**, 678–696 (2022)
147. Benink, H.A., Gordillo, J.L., Pardo, J.P., Stephens, C.R.: Market efficiency and learning in an artificial stock market: a perspective from Neo-Austrian economics. *J. Empir. Financ.* **17**(4), 668–688 (2010)
148. Ji, J., Wang, D., Xu, D., Xu, C.: Combining a self-exciting point process with the truncated generalized Pareto distribution: an extreme risk analysis under price limits. *J. Empir. Financ.* **57**, 52–70 (2020)
149. Visaltanachoti, N., Charoenwong, C., Ding, D.K.: Information asymmetry in warrants and their underlying stocks on the stock exchange of Thailand. *J. Empir. Financ.* **18**(3), 474–487 (2011)
150. Ballocchi, G., Dacorogna, M.M., Hopman, C.M., Müller, U.A., Olsen, R.B.: The intraday multivariate structure of the Eurofutures markets. *J. Empir. Financ.* **6**(5), 479–513 (1999)
151. Bauwens, L., De Backer, B., Dufays, A.: A Bayesian method of change-point estimation with recurrent regimes: application to GARCH models. *J. Empir. Financ.* **29**, 207–229 (2014)
152. Díaz-Hernández, A., Constantinou, N.: A multiple regime extension to the Heston-Nandi GARCH (1, 1) model. *J. Empir. Financ.* **53**, 162–180 (2019)
153. Kuester, K., Mittnik, S., Paolella, M.S.: Value-at-risk prediction: a comparison of alternative strategies. *J. Financ. Econ.* **4**(1), Winter, 53–89 (2006)
154. Gerlach, R., Wang, C.: Bayesian semi-parametric realized conditional autoregressive expectile models for tail risk forecasting. *J. Financ. Econ.* **20**(1), Winter, 105–138 (2022)
155. Kozhan, R., Salmon, M.: The information content of a limit order book: the case of an FX market. *J. Financ. Mark.* **15**(1), 1–28 (2012)
156. Arnold, T., Hilliard, J.E., Schwartz, A.: Short-maturity options and jump memory. *J. Financ. Res.* **30**, 437–454 (2007)
157. Herron, R.: How Much Does Your Banker's Target-Specific Experience Matter? Evidence from Target IPO Underwriters that Advise Acquirers. *J. Financ. Serv. Res.* **61**, 217–258 (2022)
158. Liu, Q., Luo, Q., Tse, Y., Xie, Y.: The market quality of commodity futures markets. *J. Futur. Mark.* **40**, 1751–1766 (2020)
159. Posselt, A.M.: Dynamics in the VIX complex. *J. Futur. Mark.* **42**, 1665–1687 (2022)
160. Roberts, M.C.: Technical analysis and genetic programming: constructing and testing a commodity portfolio. *J. Futur. Mark.* **25**, 643–660 (2005)
161. Wang, J.: Trading and hedging in S&P 500 spot and futures markets using genetic programming. *J. Futur. Mark.* **20**, 911–942 (2000)
162. Mamre, M.O., Sommervoll, D.E.: Coming of Age: Renovation Premiums in Housing Markets. *J. R. Estate Financ. Econ.* (2022)
163. Porth, L., Pai, J., Boyd, M.: A Portfolio Optimization Approach Using Combinatorics With a Genetic Algorithm for Developing a Reinsurance Model. *J. Risk Insur.* **82**, 687–713 (2015)

164. Owadally, I., Haberman, S., Gómez Hernández, D.: A Savings Plan With Targeted Contributions. *J. Risk Insur.* **80**, 975–1000 (2013)
165. Gatzert, N., Kellner, R.: The Effectiveness of Gap Insurance With Respect to Basis Risk in a Shareholder Value Maximization Setting. *J. Risk Insur.* **81**, 831–860 (2014)
166. Cont, R.: Model uncertainty and its impact on the pricing of derivative instruments. *Math. Financ.* **16**, 519–547 (2006)
167. Karathanasopoulos, A., Dunis, C., Khalil, S.: Modelling, forecasting and trading with a new sliding window approach: the crack spread example. *Quant. Financ.* **16**(12), 1875–1886 (2016)
168. Brabazon, A.: The new ‘brew’ on the Liffey: How fmc2 is adding the yeast. *Quant. Financ.* **10**(3), 241–245 (2010)
169. LeBaron, B.: A builder’s guide to agent-based financial markets. *Quant. Financ.* **1**(2), 254–261 (2001)
170. Horvath, B., Muguruza, A., Tomas, M.: Deep learning volatility: a deep neural network perspective on pricing and calibration in (rough) volatility models. *Quant. Financ.* **21**(1), 11–27 (2021)
171. Payne, B.C., Tresl, J.: Hedge fund replication with a genetic algorithm: breeding a usable mousetrap. *Quant. Financ.* **15**(10), 1705–1726 (2015)
172. Wang, C., Chen, Q., Gerlach, R.: Bayesian realized-GARCH models for financial tail risk forecasting incorporating the two-sided Weibull distribution. *Quant. Financ.* **19**(6), 1017–1042 (2019)
173. Cheng, C.-H., Wang, S.-H.: A quarterly time-series classifier based on a reduced-dimension generated rules method for identifying financial distress. *Quant. Financ.* **15**(12), 1979–1994 (2015)
174. Oesch, C., Maringer, D.: Low-latency liquidity inefficiency strategies. *Quant. Financ.* **17**(5), 717–727 (2017)
175. Tsao, C.-Y.: Portfolio selection based on the mean-VaR efficient frontier. *Quant. Financ.* **10**(8), 931–945 (2010)
176. Hendricks, D., Gebbie, T., Wilcox, D.: Detecting intraday financial market states using temporal clustering. *Quant. Financ.* **16**(11), 1657–1678 (2016)
177. Lien, D., Tse, Y.K., Zhang, X.: Structural change and lead-lag relationship between the Nikkei spot index and futures price: a genetic programming approach. *Quant. Financ.* **3**(2), 136–144 (2003)
178. Tafin Djoko, D., Tillé, Y.: Selection of balanced portfolios to track the main properties of a large market. *Quantitative Finance* **15**(2), 359–370 (2015)
179. Wang, D., Tu, J., Chang, X., Li, S.: The lead-lag relationship between the spot and futures markets in China. *Quant. Financ.* **17**(9), 1447–1456 (2017)
180. Sariev, E., Germano, G.: Bayesian regularized artificial neural networks for the estimation of the probability of default. *Quant. Financ.* **20**(2), 311–328 (2020)
181. Acosta-González, E., Armas-Herrera, R., Fernández-Rodríguez, F.: On the index tracking and the statistical arbitrage choosing the stocks by means of cointegration: the role of stock picking. *Quant. Financ.* **15**(6), 1075–1091 (2015)
182. Ma, F., Liang, C., Zeng, Q., Li, H.: Jumps and oil futures volatility forecasting: a new insight. *Quant. Financ.* **21**(5), 853–863 (2021)
183. Creamer, G., Freund, Y.: Automated trading with boosting and expert weighting. *Quant. Financ.* **10**(4), 401–420 (2010)
184. Creamer, G.: Model calibration and automated trading agent for euro futures. *Quant. Financ.* **12**(4), 531–545 (2012)
185. Zumbach, G.: Volatility conditional on price trends. *Quant. Financ.* **10**(4), 431–442 (2010)
186. Funahashi, H.: Artificial neural network for option pricing with and without asymptotic correction. *Quant. Financ.* **21**(4), 575–592 (2021)
187. Kim, H., Jun, S., Moon, K.-S.: Stock market prediction based on adaptive training algorithm in machine learning. *Quant. Financ.* **22**(6), 1133–1152 (2022)
188. How, J., Ling, M., Verhoeven, P.: Does size matter? a genetic programming approach to technical trading. *Quant. Financ.* **10**(2), 131–140 (2010)

189. Fang, J., Lin, J., Xia, S., Xia, Z., Hu, S., Liu, X., Yong, J.: Neural network-based automatic factor construction. *Quant. Financ.* **20**(12), 2101–2114 (2020)
190. Ji, J.R., Wang, D., Tu, J.Q.: Modifying a simple agent-based model to disentangle the microstructure of Chinese and us stock markets. *Quant. Financ.* **18**(12), 2067–2083 (2018)
191. Yaros, J.R., Imielinski, T.: Data-driven methods for equity similarity prediction. *Quant. Financ.* **15**(10), 1657–1681 (2015)
192. Shu, L., Shi, F., Tian, G.: High-dimensional index tracking based on the adaptive elastic net. *Quant. Financ.* **20**(9), 1513–1530 (2020)
193. Pfeuffer, M., Möstel, L., Fischer, M.: An extended likelihood framework for modelling discretely observed credit rating transitions. *Quant. Financ.* **19**(1), 93–104 (2019)
194. Dempster, M.A.H., Jones, C.M.: A real-time adaptive trading system using genetic programming. *Quant. Financ.* **1**(4), 397–413 (2001)
195. Austin, M.P., Bates, G., Dempster, M.A.H., Leemans, V., Williams, S.N.: Adaptive systems for foreign exchange trading. *Quant. Financ.* **4**(4), 37–45 (2004)
196. Azzone, M., Bavieria, R.: Additive normal tempered stable processes for equity derivatives and power-law scaling. *Quant. Financ.* **22**(3), 501–518 (2022)
197. Siikanen, M., Nögel, U., Kannainen, J.: Trading too expensively in the FX market?. *Quant. Financ.* **19**(12), 1933–1944 (2019)
198. Chen, M.-Y.: Using a hybrid evolution approach to forecast financial failures for Taiwan-listed companies. *Quant. Financ.* **14**(6), 1047–1058 (2014)
199. Grishina, N., Lucas, C.A., Date, P.: Prospect theory-based portfolio optimization: an empirical study and analysis using intelligent algorithms. *Quant. Financ.* **17**(3), 353–367 (2017)
200. Murphy, N.J., Gebbie, T.J.: Learning the dynamics of technical trading strategies. *Quant. Financ.* **21**(8), 1325–1349 (2021)
201. Meade, N., Beasley, J.E.: Detection of momentum effects using an index out-performance strategy. *Quant. Financ.* **11**(2), 313–326 (2011)
202. Jobst, N.J., Horniman, M.D., Lucas, C.A., Mitra, G.: Computational aspects of alternative portfolio selection models in the presence of discrete asset choice constraints. *Quant. Financ.* **1**(5), 489–501 (2001)
203. Gibson Brandon, R., Gyger, S.: Optimal hedge fund portfolios under liquidation risk. *Quant. Financ.* **11**(1), 53–67 (2011)
204. Oeuvray, R., Junod, P.: A practical approach to semideviation and its time scaling in a jump-diffusion process. *Quant. Financ.* **15**(5), 809–827 (2015)
205. Baule, R., Entrop, O., Wessels, S.: Performance measurement for option portfolios in a stochastic volatility framework. *Quant. Financ.* **22**(3), 519–539 (2022)
206. Rebonato, R.: Probably approximately correct. *Quant. Financ.* **16**(3), 349–353 (2016)
207. Luss, R., D'Aspremont, A.: Predicting abnormal returns from news using text classification. *Quant. Financ.* **15**(6), 999–1012 (2015)
208. Yamamoto, R.: Trading profitability from learning and adaptation on the Tokyo Stock Exchange. *Quant. Financ.* **16**(6), 969–996 (2016)
209. Zhang, S.M., Feng, Y.: American option pricing under the double Heston model based on asymptotic expansion. *Quant. Financ.* **19**(2), 211–226 (2019)
210. Kim, S., Kim, S.: Index tracking through deep latent representation learning. *Quant. Financ.* **20**(4), 639–652 (2020)
211. Satpathy, T., Shah, R.: Sparse index tracking using sequential Monte Carlo. *Quant. Financ.* **22**(9), 1579–1592 (2022)
212. Li, W., Paraschiv, F., Serpinis, G.: A data-driven explainable case-based reasoning approach for financial risk detection. *Quant. Financ.* (2022)
213. Lu, X., Abergel, F.: High-dimensional Hawkes processes for limit order books: modelling, empirical analysis and numerical calibration. *Quant. Financ.* **18**(2), 249–264 (2018)
214. Zhao, Z., Xu, F., Du, D., Meihua, W.: Robust portfolio rebalancing with cardinality and diversification constraints. *Quant. Financ.* **21**(10), 1707–1721 (2021)

Chapter 25

Evolutionary Machine Learning and Games



**Julian Togelius, Ahmed Khalifa, Sam Earle, Michael Cerny Green,
and Lisa Soros**

Abstract Evolutionary machine learning (EML) has been applied to games in multiple ways, and for multiple different purposes. Importantly, AI research in games is not only about playing games; it is also about generating game content, modeling players, and many other applications. Many of these applications pose interesting problems for EML. We will structure this chapter on EML for games based on whether evolution is used to augment machine learning (ML) or ML is used to augment evolution. For completeness, we also briefly discuss the usage of ML and evolution separately in games.

25.1 Introduction

Games of all sorts (including card games, board games, and video games) provide a rich domain for exploring computational intelligence. In many ways, games reflect the parts of the real world that we as humans find interesting, isolating key facets of our experience and encapsulating them within a tractable and interactive medium.

Beyond mere entertainment, games provide a unique domain for exploring and evaluating AI. Historically, the intersection of AI and games has focused on agents that play specific games *well*. In this way, games complement the litany of task environments for AI such as embodied agent control. However, games offer additional

J. Togelius (✉) · S. Earle · M. C. Green
Computer Science and Engineering, New York University, 370 Jay Street, Brooklyn, NY 07960,
USA
e-mail: julian.togelius@gmail.com

J. Togelius
Odl.ai, Nørrebrogade 184, 2200 Copenhagen, Denmark

A. Khalifa
Institute of Digital Games, University of Malta, 20 TriqL-Esperanto, Msida MSD2080, Malta

L. Soros
Computer Science, Barnard College, 3009 Broadway, New York NY 10027, USA

challenges beyond reward maximization such as effecting spirited play or emulating the style of particular humans.

In addition to playing games, there are challenges including generating content (such as levels, quests, textures, and characters), modeling players, matching players, and adapting interfaces. Content generation in particular requires deeply creative algorithms capable of understanding the essence of a domain and conjuring up new artifacts. In this way, games also provide an opportunity for exploring concepts such as algorithmic innovation and open-endedness.

In this chapter, we survey the application of EML to games. We take an inclusive view of EML, focusing on cases where a ML model is evolved, but including examples of all kinds of interaction between evolution and ML. We also give brief overviews of the use of non-evolutionary ML and non-ML evolution in games, but given the breadth of the topic, those sections are mere sketches.

EML can, in one form or another, be applied to almost any AI challenge in games. However, this family of methods has seen much more application in some areas rather than others. Reflecting on this, a relatively large number of examples in this chapter will be from game content generation. But there will also be plenty of examples of game-playing evolutionary ML.

25.2 Machine Learning in Games

Some of the earliest advancements in ML are due to research on game-playing. In particular, Samuel’s Checkers player from 1959 [81] was the first example of what we now call reinforcement learning. While programs for Chess [12], Checkers [82], and Go initially built (and usually still build) on some form of tree search, machine-learned board value functions were introduced at an early stage and became crucial to any advanced efforts to play classical board games. These value functions could be learned through supervised learning, reinforcement learning, or some combination. AlphaGo [89] and AlphaZero [90] represent very successful combinations of tree search with ML that originated in research on Go playing but have found applications in a wide variety of fields.

At the outset of the deep learning era, certain video games came to play important roles as testbeds and benchmarks of deep reinforcement learning algorithms. Deep Q-networks were introduced in a landmark Nature paper in 2015 [67], where they were trained on games from the Atari 2600 console via the Arcade Learning Environment [6]. Since then, Atari games have been a crucial benchmark for deep reinforcement learning, and new algorithms are often tested on Atari games first [49]. Several other video games have also become important as benchmarks for RL algorithms, notably Doom [52] and MineCraft [40]. To a lesser extent, supervised learning has been applied to these games to learn game-playing from large sets of human playtraces [3].

Although a much smaller research area, ML algorithms of various kinds have also been applied to generating game content [100]. Most of these applications fall into self-supervised learning, such as using generative adversarial networks or recurrent neural networks for generating levels for Super Mario Bros [2, 99, 112] or cards for Magic: The Gathering [98]. However, there have also been attempts to use reinforcement learning for level generation more recently [53].

Other AI challenges in games for which ML has been used prominently include player modeling [91], cheat detection [47], and matchmaking [66]. In player modeling, preference learning has been successfully applied to predicting player affect in response to game levels, game situations, or behavior. For cheat detection, explainable ML algorithms (such as decision trees) are most commonly used as the AI cannot act by itself and block players. It usually needs the moderator's input to confirm if the flagged player is a cheater before banning the player from the game. Microsoft's TrueMatch system, widely used in XBox games, uses ML on top of a ranking system to learn to match players well.

25.3 Evolution in Games

The robustness and wide applicability of evolutionary search means that evolutionary algorithms have been applied rather extensively to games. Many, perhaps most, examples of applications of evolution to games are in the form of neuroevolution, which counts as EML. However, there are also “pure” applications of evolutionary computation in games. Here we will survey two types: evolutionary planning and search-based procedural content generation.

25.3.1 *Evolutionary Planning*

For playing a game that has a fast-forward model, one preferably uses some kind of search algorithm for planning. Traditionally, this would mean a tree search algorithm, such as a Minimax variation for two-player games or some form of A* for single-player games [79]. For games with a nontrivial search depth, one would need to cut off the search at some specified depth and use a board or state evaluation function to evaluate the deepest search node.

The plan that is generated by the tree search algorithm is simply a list of actions, with a numeric value (obtained from the state evaluation function) that signifies the projected value of that sequence of actions. Recognizing this, one might just as well use an evolutionary algorithm to evolve the plan, using the state evaluation function as a fitness function. In an approach called rolling horizon evolution [71], several groups have done this for single-player games [31], with results that are generally competitive with state-of-the-art tree search algorithms such as Monte Carlo Tree Search [11].

The advantages of evolution over classic tree search algorithms become more clear in games with a very large branching factor [84]. Games that feature control over multiple units, including many real-time or turn-based strategy games, can have branching factors in millions or even billions. This chokes most tree search algorithms as the search never gets past the first few moves. Evolutionary algorithms, operating on the level of the whole plan as a sequence, are not affected by this limitation. Results for several strategy games [50] show that evolutionary planning can out-compete tree search by a large margin.

25.3.2 *Search-Based PCG*

After game playing, procedural content generation (PCG) is probably the topic in games that has received the most interest from the AI research community [86]. Within academic research on PCG, evolutionary computation is currently the dominant approach because of the natural fit between the method and the problem. In search-based PCG, as the application of evolution to PCG problem is called, individual pieces of game content are evolved with a fitness function that reflects some measure of content quality [109]. This approach has been applied to a large number of types of game content, including quests [20], music [26], textures [114], rules [17], and levels for platform games [85], first-person shooters [13], and real-time strategy games [62].

Finding a fitness function that accurately reflects the quality of a candidate content artifact is typically nontrivial. Judging the quality of, for example, a game level is inherently hard and often requires playing the game. Writing code that performs this quality judgment is often harder still. Therefore, many search-based PCG implementations use simulation-based evaluation, where an agent of some kind plays part of the game that incorporates the content under evaluation. For example, the agent could play a level to see if it is playable under certain conditions [54]. Such agents may be based on ML [108]. In other cases, the fitness function is model-based and relies on some kind of learned model to do the evaluation [51]. In yet other works, fitness is assigned interactively by a human player [32].

25.4 Evolutionary Machine Learning in Games

In this section, we overview the use of EML in games research. While one way of categorizing this body of work is to divide game application (e.g. planning for non-player character actions, and PCG), another is to tear apart EML, considering the ways in which evolution and ML interact. We take the latter approach in the context of this book, as the former is better suited to works on AI and Games [116]. We survey research in games using EML (as shown in Fig. 25.1) by

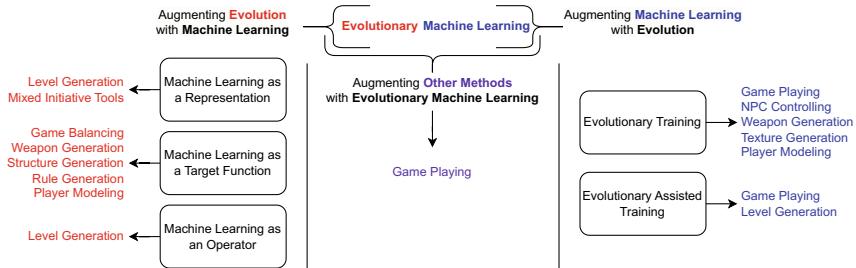


Fig. 25.1 Taxonomy of EML in Games based on the dynamics between evolution and ML with examples of the usage of these techniques in games

- Augmenting Machine Learning with Evolution.
- Augmenting Evolution with Machine Learning.
- Augmenting other methods with Evolutionary Machine Learning.

The way we acquired this division is by looking into what is the end goal of the work that uses EML and analyze what is the core algorithm that is being used. For example, if the end goal is to have a neural network agent that controls a player in a video game, then evolution in that case is helping the ML (neural network) to achieve its goal. On the other hand, if the goal is to use evolution to find a certain content, then the search process is what is important and the ML is helping it to achieve its goal. We also found some other cases where the goal algorithm is neither an evolutionary algorithm nor ML, but where EML nonetheless plays an important supporting role.

25.5 Augmenting Machine Learning with Evolution

The easiest way to incorporate the evolutionary algorithm with ML is to utilize evolutionary optimization power to help the ML algorithm. Thinking about it from this direction allows us to see two ways where evolution can help. We can either use an evolutionary algorithm as the training process for the ML or the evolutionary algorithm for adjusting and helping the training process of ML, or both.

25.5.1 Evolutionary Training

Evolution can be used to directly train neural networks, either the weights of the network, its topology, or both. This is called *neuroevolution* and has a long history in artificial life, robotics [68], and video game research [76]. It should be noted that some recent neuroevolution work, specifically focused on finding structures for deep networks, is alternatively billed as “neural architecture search” [27].

25.5.1.1 Neuroevolution for Gameplay Tasks

The probably most common use for neuroevolution in games is as a form of reinforcement learning. Here, the fitness evaluation of a chromosome is calculated by converting the genotype to a neural network (which might be as simple as setting the weights of a fixed topology) and using it to try to play the game, and basing the fitness on how well the network played the game. Compared to other forms of reinforcement learning, such as those based on variations of temporal difference learning or policy gradients, neuroevolutionary reinforcement learning tends to be more reliable but has issues with scaling to very large networks [76, 107].

In board games, the most common (and probably most sensible) version of neuroevolution is to evolve a board value function used together with some version of the Minimax algorithm. Successful applications of this method go back a few decades [28]; see also Sect. 25.7.

In video games, where fast and deterministic forward models are rare, it is typically more appropriate to use neuroevolution to generate a network that directly selects what action to take based on an observation of the environment. Early examples of this type of neuroevolution include work on 2D car racing games [105, 106]. In those examples, the neural networks were fed simulated rangefinders representing the track geometry in front of the car, and outputted accelerator/brake and steering commands. Another relatively early example of this type of neuroevolution is the evolution of neural nets to play Super Mario Bros based on discrete grid sensor inputs [104].

As the ALE benchmark suite, featuring games from the old Atari 2600 video game console, became popular with the rise of deep reinforcement learning, evolutionary algorithms were also applied to learn to play Atari games from pixel inputs. An influential paper by a team at OpenAI showed that a simple evolution strategy can be competitive with standard DQN, and in particular that evolution scales effortlessly compared to other types of reinforcement learning [80]. Follow-up work by other authors showed that the slightly more sophisticated Canonical Evolution Strategy can perform much better [16]. However, the more recent and sophisticated gradient-based RL methods outperform existing evolutionary methods on Atari games. In general, while standard neuroevolution remains competitive with gradient-based RL when the network size is small, it tends to not scale very well to very large networks. This may be because the single fitness measure imparts less information than the dense rewards that can be used with some RL methods, or because of the lack of directional information in the random mutations in a high-dimensional space.

Various attempts have been made to overcome this limitation of neuroevolutionary reinforcement learning for games with high-dimensional (such as visual) input space. One idea is to separate visual processing from the policy. This way, a smaller network that receives a lower dimensional encoding of the observation can be trained effectively by evolution. One idea is to use an autoencoder to compress the visual input, feeding the smaller policy network of the bottleneck layer in the autoencoder. This was tested successfully on the classic FPS game Doom, in a setup where the autoencoder was continually re-trained as new sections of the level were discovered [1]. Another idea is to represent the state using a dictionary of centroids in state

space, feeding a sparse encoding to the policy network. This setup can be used to evolve surprisingly tiny networks, with as few as six neurons, to play Atari games well [18].

One of the areas where neuroevolution can do things that regular reinforcement arguably cannot is in topology and weight evolution, where both the topology and the weights of the network are evolved. The pre-eminent algorithm here is NEAT by Stanley and Miikkulainen, which has been applied to learn to play various games and performs very well on small to moderate-size networks [96]. Later, it has been applied to the General Video Game framework with decent results [72]. HyperNEAT is an indirect encoding that can in principle handle much larger input spaces [95], and which has been applied to playing Atari games with some success [14].

25.5.1.2 Neuroevolution for Non-gameplay Tasks

Neuroevolution can also be applied to other tasks in games besides game playing. In particular, there are several prominent examples of neuroevolution for procedural content generation. This includes the Galactic Arms Race game [43], where players collaboratively evolve weapons, and the Petalz social network game where players evolve flowers [75]. One can even use neuroevolution as a form of meta-content generator, evolving a population of networks that can generate a large range of different levels [24]. An interesting use of neuroevolution for non-player characters is in the NERO game, where players train populations of agents to fight for them [94]. Even earlier, the cult video game Creatures uses a form of neuroevolution as a game mechanic.

Neuroevolution has also been used to model a certain playstyle [45, 46]. Holmgaard et al. used a $\mu + \lambda$ evolutionary strategy to evolve a single-layer neural network that decides on the next target in MiniDungeons. The agent then uses the A* algorithm to navigate toward the selected target. The agents were evolved using action agreement ratio [45], tactical agreement ratio, and strategic agreement ratio [46] to mimic different playstyles from a corpus of collected human data. The final evolved agents were not only able to finish unseen levels but also navigate them in the same vein as the target playstyle.

25.5.2 Evolutionary Assisted Training

In the work described in Sect. 25.5.1 above, the evolutionary process serves to learn more sophisticated policies—e.g. for game playing or design, whether by the modification of action sequences, game assets, or of neural network weights or architectures. In this section, on the other hand, the learning or training process is implemented by some non-evolutionary algorithm (typically gradient descent), while evolution exists as an auxiliary process that renders learning more capable or efficient. This section focuses primarily on a particularly prominent role taken on by such an

assistive evolutionary process, which is creating or curating a “curriculum” for a learning agent.

25.5.2.1 Curriculum Design in Machine Learning

Curriculum learning [7] takes inspiration from human developmental psychology, where, by way of example, children are taught complex subjects (e.g. mathematics) incrementally over time, by first familiarizing them with more elementary and fundamental concepts (such as basic arithmetic) before moving on to more sophisticated techniques (such as calculus or linear algebra). In the context of ML, a curriculum may expose a learning agent to increasingly complex training examples, switching from simpler to more complex ones once the agent’s abilities reach a certain level.

For example, a neural network-based agent tasked with navigating through a maze to a goal tile will in all likelihood be incapable of navigating a complex maze by sheer luck at the very beginning of training (i.e. with randomly initialized weights resulting in it taking effectively random actions at each step). It would thus be wasteful to expose it to such complex mazes at the beginning of training. It has a much better chance, on the other hand, of stumbling upon the goal tile in simpler mazes, where the goal is close to the starting position. A curriculum learning approach could thus order mazes by their complexity (or the distance between start and goal positions) only introducing more complex mazes into the pool of training data once the agent has learned to reliably solve a set of simpler mazes.

In ML, curriculum learning can not only make training more efficient (avoiding training on infeasible examples early on) but also lead to improved generalization, by allowing the model to incrementally learn more meaningful representations of the problem at hand. The importance of curriculum learning for training generally capable strategies over a large space of tasks has been demonstrated in XLand [102, 103], where a large set of tasks are generated in a 3D environment, where embodied agents take visual input and must navigate a procedurally generated terrain to achieve certain goal states, which involve manipulating and re-combining various primitive objects. In XLand, a massive space of tasks (involving different rules and initial map conditions) are generated before learning, then curated during agent training so as to produce a curriculum of increasing complexity. This curation is aided by metrics using agent play-through to measure whether a given task is both nontrivial and learnable (i.e. whether it is on the “frontier” of agent abilities). The resulting player agents are able to generalize to new tasks—Involving different object-recombination mechanics and goal states—that were not seen during training.

25.5.2.2 Evolutionary Curriculum Design

Other work generates training environments on the fly, then curates or filters them to ensure learnability, resulting in a series of environments of increasing complexity over the course of agent training player agents. PAIRED [21] introduces this notion

of “Unsupervised Environment Design”, using an RL agent to generate level layouts of training environments so as to maximize learnability (formulated as approximate regret). More recently, ACCEL [70] effectively evolves these environments, applying small mutations to level layouts and filtering them for regret. The domains considered include tile-based maze-like environments in which the player must navigate to a goal (while sometimes avoiding lava) and the 2D physics-based “bipedal walker” environment. The mutations involve changing the state of given tiles in the former case and changing the height of the terrain in the latter. The fitness criterion of a mutated level is the regret it induces in the player agent. The authors show that the evolution of increasingly complex environments (lengthy and obstacle-heavy mazes, and rough or steep terrain) coincide with the development of a robust player agent capable of generalizing to unseen environments (e.g. capable of solving a maze in a grid much larger than those evolved during training).

POET [113] similarly evolves environments online while training player policies in the bipedal walker environment, though it trains multiple player agents concurrently, pairing these with particular environments (and occasionally transferring players to different environments mid-training in a paradigm reminiscent of transfer learning [110]). Here, terrain is represented by a Compositional Pattern Producing Network (CPPN [93]), and evolved using Neuroevolution of Augmenting Topologies (NEAT [96]). Agent policies are optimized via an Evolutionary Strategy [80]. Mutated environments are added to the pool of environments if they are neither too hard nor too easy for existing agents, and if they are sufficiently novel with respect to environments already in the pool, thus encouraging diversity among the problems generated (and solved) by the algorithm. PINSKY [22] applies the framework introduced by POET to 2D tile-based games.

In Go-Explore [25], a variant of novelty search [60] is used to incentivize exploration by an RL agent, by effectively maintaining an archive of behaviors that result in novel states in a sparse reward problem. The domain tackled in this work is Montezuma’s Revenge, a side-scrolling adventure game from the Atari 2600 suite, which was formerly unsolved by methods not relying on expert demonstrations. Go-Explore facilitates exploration by looking at the (x, y) coordinate location of the player avatar on the screen, and, whenever a new position is reached by the player, storing in memory the sequence of behaviors that led to this novel state. On subsequent play-throughs, the agent can then reliably navigate back to an (x, y) position on the frontier of that which it has already explored, so that it will tend to explore further and further out from its starting location, and so that once a tricky segment has been solved—such as jumping carefully across a dangerous chasm—it can be reliably solved on every subsequent play-through without further trial and error. When the goal state is reached, they train a game-playing agent on the successful trajectories from the archive.

25.6 Augmenting Evolution with Machine Learning

The goal in augmenting the evolutionary process is to mimic the output of the evolution process by using ML to create an easier landscape to navigate, better operators, faster fitness function, etc.

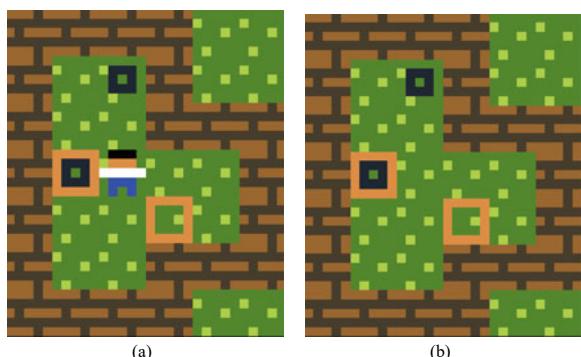
25.6.1 Machine Learning as a Representation

Evolutionary algorithms are sensitive to the input representation [35]. There is a body of research on understanding and exploring the different representations and their effects [78]. In most of these works, it is agreed that a good representation provides us with a smooth fitness landscape. This means that individuals that are close to each other in representation space have similar fitness. In many game domains such as level generation, a direct representation (2D matrix of tiles) is usually not the best as it is too sensitive to small changes. For example, removing the player tile from a level will make the level unplayable immediately (Fig. 25.2). Instead of utilizing domain knowledge to figure out a good representation, ML models can be used to learn this representation. In that case, ML acts as a genotype-to-phenotype mapper for the evolutionary algorithm. The evolutionary algorithm can just work on the provided representation (called latent variables) and modify it using a mutation function, then the ML transforms it to the phenotype where it gets evaluated using a fitness function.

25.6.1.1 Super Mario Bros

Although there are many ML methods that can be used to learn a good representation, most of the work focuses on either AutoEncoders [57] or Generative Adversarial Networks (GANs) [36]. For example, Volz et al. [112] trained a GAN on the levels of Super Mario Bros (Nintendo, 1985). The levels are divided into single scenes of

Fig. 25.2 An example of a Sokoban level with and without the human player



size (28×14) using a sliding window. After training the network on generating new scenes, they used CMA-ES [42] algorithm to search the latent space of the GAN for playable Mario levels with different game features such as maximizing/minimizing the number of jumps. The fitness was calculated using an A* agent that measures the playability percentage of the generated levels and the number of jumps that have been performed. The generated levels mostly follow the structure of original Mario levels without the need for specifying that in the fitness function or the representation [19, 85]. Following the previous work, Fontaine et al. [29] used a quality diversity algorithm called Covariance Matrix Adaptation MAP-Elites (CMA-ME) [30] to discover new and diverse playable levels. Utilizing CMA-ME helped the authors overcome the repetition of the generated levels using normal CMA-ES and find new levels that are hard to find using normal CMA-ES such as levels that combine two different level styles (Fig. 25.3). Another advantage is at the end of the run, the algorithm manages to produce a corpus of playable levels with different features instead of a single playable level as in the case of CMA-ES.

So far, the previously discussed work only focused on generating a small scene of Super Mario levels of size (28×14); creating a full level by concatenating the generated scenes sometimes might end in an unplayable level (see Fig. 25.4). This is because evolution does not have any context about the previously generated scenes. This can be easily solved by modifying the fitness to incorporate all the previous scenes. The problem is that evolution cannot change anything from the previously generated scenes. This might cause evolution to repeat certain patterns to solve the problem. Another simple solution is to evolve the whole level all at once and use the GAN to transform each sequence independently. The problem with that is for very long levels; this could be hard as the representation might be big and not easy to evolve due to conflict in fitness between different parts of the chromosome. To solve this issue, Schrum et al. [83] proposed encoding the big level using a compressed representation which is encoded in the form of a CPPN [93]. The CPPN takes the location of the scene, and it spits out the latent variable for that area. The NEAT [96] algorithm was used to create the full level. The output of this experiment shows that

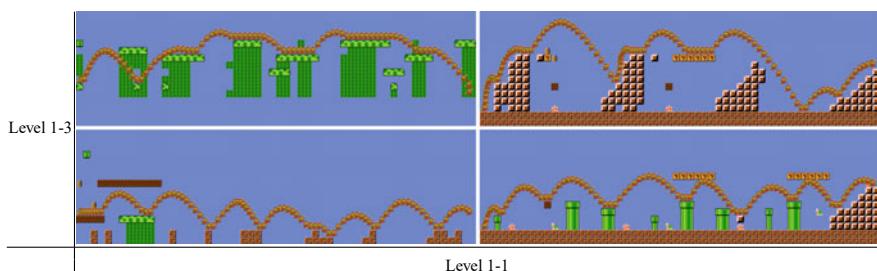


Fig. 25.3 Generated Super Mario scenes using CMA-ME that are similar to either sky levels or ground levels

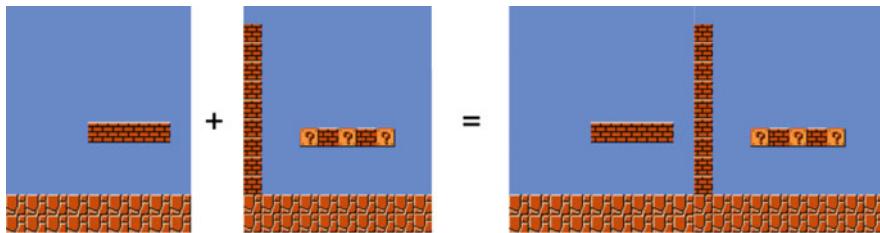


Fig. 25.4 These Mario scenes are playable on their own, but combining them in that order will not be playable

using CPPNs to evolve the structure of the level followed by normal evolution on the concatenated latent variables produced better results than any part alone.

25.6.1.2 Other Applications

Tanabe et al. [101] used latent variable evolution with Variational AutoEncoder (VAE) [74] to generate levels for Angry Birds (Rovio, 2009). They treated the generated levels as a sequence instead of being a tile map to overcome the problem of having variable size objects and having objects overlayed each other. They searched the latent space of VAE using CMA-ES algorithm to find levels that minimize the usage of birds, maximize the number of pigs, etc. search. Another avenue is to use interactive evolution to create a mixed initiative tool [87] combined with the power of ML. This is usually a common avenue in subjective domains that are hard to measure its success. There is a substantial breadth of work in this area, and a full review is outside the scope of this book chapter. However, some recent works generate art [48], human faces [117], backgrounds, and shoes [10]. However, as far as we know, these tools have not been used in texture generation or game art. This could be an interesting avenue to explore with the rise of large generative models such as Stable Diffusion [77].

25.6.2 Machine Learning as a Target Function

Usually, the bottleneck for evolution is calculating the fitness function [92], which motivated people to seek methods to approximate its value to speed up the evolutionary process. This approach involves surrogate models [92]. Surrogate models have been modeled using different ML methods such as linear regression [37], Gaussian mixture models [63], and neural networks [51]. In this section, we decided to call it a target function instead of a fitness function as ML could be used to approximate not only the fitness function but also features of the phenotype such as time to finish the level, and number of jumps in a playtrace. These features can be used either as

behavior characteristics in quality diversity algorithms [8, 118] or a diversity score for divergent search algorithms [5, 61].

We can divide surrogate models based on how they are trained: online and offline methods. The online method focuses on training the surrogate model as a part of the evolutionary process. The evolution helps to create the dataset for training the ML model. On the other hand, the offline method focuses on training the ML model beforehand. This makes surrogate models similar to reinforcement learning where the online method is similar to off-policy reinforcement learning and the offline method is similar to offline reinforcement learning where we collect the data before training the model.

25.6.2.1 Online Methods

In the games domain, most of the work utilizes online training due to the small number of data that can be found to train ML models. Volz et al. [111] used multi-objective optimization to create balanced cards for the Top Trumps card game (Dubreq, 1978). The goal is to generate a group of unique balanced decks such that there is no dominant strategy to win the game. The surrogate model here is a statistical model that predicts the minimum number of simulation that is needed to have an accurate estimation of the win rate. The results showed that generated decks from surrogate models are as good as normal simulations and require a lot less computational power. In a similar vein, Zhang et al. [118] used Deep Surrogate Assisted MAP-Elites to generate a deck of cards for Hearthstone (Blizzard, 2014). They not only used the surrogate model to calculate the fitness (the average difference of health between both players) but also the behavior characteristics (the average number of cards in hand and the number of turns till the end of the match) needed for the MAP-Elites Archive. The model is trained online after a fixed number of iterations to make sure the output is correct and up to date. Bhatt et al. [8] generalized the system by adding another surrogate model that predicts the agent playtrace instead of just the final metrics and tested it on generating mazes and Super Mario Bros levels. The new system works better in comparison with the previous one introduced by Zhang et al. [118] as the introduced agent prediction helps improve the results.

Some quality diversity and divergent search algorithms need ML as their core element such as Surprise Search [37]. Surprise Search abandons objectivity for the sake of surprise. The surprise score needs to be modeled using a ML algorithm so the evolution can predict new elements. The authors modeled the surprise score using linear regression to predict the genotype of the next generation given the previous generation. In this case, online models are being used as the model gets updated after each generation. Later, Gravina et al. [38] adjusted the surprise search algorithm to maintain quality and not only focus on surprise. They utilized the new algorithm (Constrained Surprise Search) to generate a diverse set of balanced weapons for FPS shooter games. They also showcased that it can be used to generate robot controllers, maze solutions, and new mazes [39]. Liapis et al. [61] and Barthet et al. [5] used a denoising autoencoder to help Constrained Novelty search [59] to find innovative

spaceship designs and MineCraft (Mojang Studios, 2009) buildings, respectively. The denoising autoencoder is trained online from the found data and then the compressed representation is used to measure the novelty of the generated content.

25.6.2.2 Offline Methods

On the offline side, Karavalos et al. [51] trained a neural network to predict the win rate and time to finish an FPS match of a given level and the player classes. They trained the model offline on tons of random matches using random levels and random player classes, then later used the trained model as a surrogate model to evolve different player classes. Migkotzidis and Liapis [65] used the same model as part of a mixed-initiative tool [115] that can be used to design balanced levels for FPS games. On the side of affective computing, Barthet et al. [4] use a variant of Go-Explore called Go-Blend (a quality diversity algorithm that keeps track of the best solutions) to explore game trajectories that can mimic different human play styles and arousal levels. They used a simple K-NN algorithm over the AGAIN dataset [64] to predict the arousal levels during playing a car racing game. Similarly, Shaker et al. [88] trained an offline surrogate model on human preference over Mario levels and used it to generate new levels for Super Mario Bros (Nintendo, 1985). Since the search space was not huge, an exhaustive search was used instead of evolution. We included this research as the exhaustive search can be easily replaced with evolution in more complex games.

A different way to use the ML model is to use it to play games instead of directly measuring playability or the attributes and then extracting the needed statistics from the playthrough. For example, Togelius and Schmidhuber [108] evolved arcade game rules using $\mu + \lambda$ evolution strategy to evolve the game rules such that the evolved games are fun to play by humans. Togelius and Schmidhuber use Koster's theory of fun [56] to estimate how learnable these games are. The learnability is approximated by measuring the improvement of a neural network agent that is trained using reinforcement learning. The generated games from that experiment were interesting but not fun for humans. This is due to the fact that AI agents play differently than humans, and we need models that can model the human experience.

Finally, surrogate models can also be used to assist game-playing agents. Olesen et al. [69] tried to use EML to control the player in a car racing experiment. ML was used to learn a forward model which predicts the next state given a certain state and action. This process is called a World Model [41], where we try to learn an approximate forward model of a specific environment. In their work, Olesen et al. used a VAE followed by Mixture Density Network [9] to learn that forward model. Later, they used an evolutionary planning algorithm (Random Mutation Hill Climber (RMHC)) to control the player car to play the game efficiently. The system first gets trained using collected data from random policy, then later it gets improved using frames and output from the RMHC algorithm. Similarly, Dockhorn et al. [23] used hashmaps and decision trees to learn a local forward model for Sokoban. A local forward model is a model that only cares about the local observation around the

player character. They use RHEA to play the game using the local forward model. Although the local forward model has high accuracy, the local model propagates errors over time which causes the planning agent to not achieve very high scores in playing the games.

25.6.3 *Machine Learning as an Operator*

Models trained via ML can be used as mutation or crossover operators during evolution. This is particularly useful when the space of mutations is very large, making it unlikely that random perturbations will lead to meaningful changes in the genome. In such a scenario, a ML-based model can be trained beforehand so as to learn useful priors over the space of possible mutations. This is particularly appealing when a large dataset of human examples is available.

As an example, ML models can be trained on mutation trajectories [55, 58], i.e. the modifications made to an entity over the course of evolution. The end result is a model that can generate levels by modifying levels similar to an evolutionary/search-based generator.

In another example, the problem of genetic programming is highly complex. A very naive approach could be to try mutating a piece of code at the string or character level, but this would result in a combinatorially explosive action space for the mutation operator. For this reason, most genetic programming approaches will “bake in” some useful priors into the action space afforded to the operator—for example, by providing a higher level set of available actions that may involve adding functional blocks of code such as if/else statements instead of individual characters or words, while treating the code as a graph or tree structure instead of merely a string.

Large Language Models (LLMs) have shown impressive performance on next-token prediction tasks when trained auto-regressively on massive corpora of human text scraped from the Internet. Despite their simple training scheme, they can be observed during inference to generate text that is coherent at a high level, and with the right prompting, can do things like generating rhyming poems about a particular subject matter in a particular style and answering questions while maintaining long-range dependencies. They have also been fine-tuned on corpora of code, and incorporated into tools such as Github Copilot.

In Evolution through Large Models (ELM, Sect. 10.10), the authors take advantage of these learned priors, and prompt LLMs to produce embodied agents capable of traversing the terrain in the SodaWorld environment. The LLM used is a diff model, which is trained on a dataset of code changes and their corresponding natural language descriptions in commit messages. ELM uses this diff model as a mutation operator by randomly selecting from a fixed set of generic commit messages (i.e. “made changes to the code”). This operator is then used inside a quality diversity (QD) loop that searches for a diverse group of ambulating soft robots in SodaWorld. The LLM can

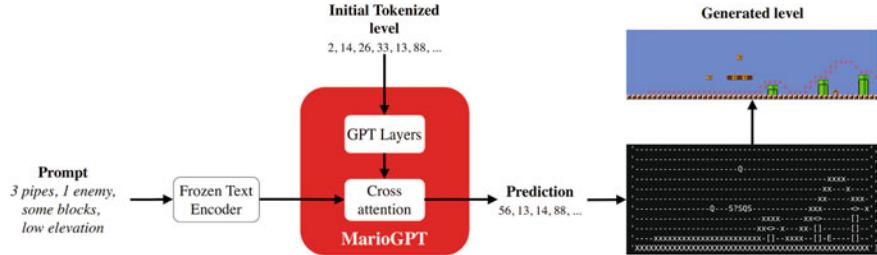


Fig. 25.5 The MarioGPT level-generation pipeline

be additionally fine-tuned on the series of mutations accepted to the archive of elites in QD (an approach similarly applied to environment generation in [55]).

An ELM type approach has also been applied to game levels in MarioGPT [97], where GPT-based language models are fine-tuned on a small dataset of Super Mario Bros (Nintendo, 1985) levels (Fig. 25.5). After fine-tuning the language model, novelty search is used as an outer evolutionary loop to search for a diverse set of novel levels generated by the model. While QD fills an archive spanning over multiple behavior characteristics or measures of interest, novelty search simply looks for sufficiently “different” individuals, with respect to those already in the archive of elites, by computing the mean distance using some distance function in phenotype space. In MarioGPT, the authors consider the mean distance between the paths corresponding to (tree search-generated) solutions for generated levels, where paths are lines over the 2D plane corresponding to the shape of generated levels.

The previous examples are the only ones that we could find specifically on games. It is clear that this area provides ample opportunities for future exploration. For example, one might consider using LLMs for Neural Architecture Search for game-playing agents similar to the work by Chen et al. [15]. Chen et al. [15] represented the neural network architecture as a piece of code that defines layers, skip connections, activations, etc. They used the diff model as a mutation operator to produce new networks. These nets are then evaluated by training them on a labeled dataset (e.g. image classification) and evaluating their test accuracy.

25.7 Augmenting Other Methods with Evolutionary Machine Learning

Finally, EML can be used to support other algorithms like tree search and reinforcement learning. Most of the known work focused on using evolution to find/train neural networks that can support tree search algorithms. Blondie24 [28] is an AI agent that can play checkers very efficiently. It was able to beat 99.61% of the matches that it played against 165 human players. The algorithm uses the Min-Max tree search algorithm [79] with the support of a neural network as a state evaluator. The neural network was trained using an evolutionary algorithm. In a similar manner, Reisinger

et al. [73] not only evolved the network weights but also the architecture using NEAT algorithm [96]. The evolved network was used as a state evaluator for the alpha-beta pruning algorithm [79] to play different board games from the General Game Playing Framework [34]. The best-evolved agent was able to beat the random agent in 5 different games from the General Game Playing framework. Finally, Gauci and Stanely [33] used the HyperNEAT algorithm [95] to prune some branches for the Min-Max algorithm [79] besides the alpha-beta pruning. The final agent was able to have a higher win rate and more ties compared to the default alpha-beta pruning algorithm.

We can notice that most of the work in that area is older than 10 years ago, and we could not find any new work that combines EML with other search algorithms. We think that the boom in computation power and the dependence on the backpropagation algorithm is the main cause of that. For example, the Alpha-Go algorithm [89] uses a neural network as a state evaluator for MCTS agent [11]. This is similar to the Blondie24 agent with the difference of having more computation power to train a big network using backpropagation [44]. This does not mean that we should abandon EML for the sake of backpropagation, but it should push us toward using EML in a smarter and different way that backpropagation cannot do. For example, we could try evolving a network that compresses the game state space such that a dynamic programming agent can play the game efficiently.

25.8 Conclusion

As demonstrated by the myriad examples in this chapter, game research has been enhanced greatly by evolution and ML. The combination of the two approaches provides particularly compelling tools for generating game agents and content; the two types of methods naturally complement each other's strengths and weaknesses. The future of this research area is exciting, as rapid advances in ML technologies will allow us to not only apply new algorithms to existing games but perhaps even to create new kinds of game-based challenges as well (which will, in turn, provide new domains for evaluating new kinds of EML systems).

References

1. Alvernaz, S., Togelius, J.: Autoencoder-augmented neuroevolution for visual doom playing. In: 2017 IEEE Conference on Computational Intelligence and Games (CIG), pp. 1–8. IEEE, MIT Press (2017)
2. Awiszus, M., Schubert, F., Rosenhahn, B.: Toad-gan: coherent style level generation from a single example. Proc. AAAI Conf. Artif. Intell. Interact. Digit. Entertain. **16**, 10–16 (2020)
3. Baker, B., Akkaya, I., Zhokov, P., Huizinga, J., Tang, J., Ecoffet, A., Houghton, B., Sampedro, R., Clune, J.: Video pretraining (vpt): learning to act by watching unlabeled online videos. In: Advances in Neural Information Processing Systems (2023)

4. Barhet, M., Khalifa, A., Liapis, A., Yannakakis, G.: Generative personas that behave and experience like humans. In: Proceedings of the 17th International Conference on the Foundations of Digital Games, pp. 1–10 (2022)
5. Barhet, M., Liapis, A., Yannakakis, G.N.: Open-ended evolution for minecraft building generation. In: IEEE Transactions on Games (2022)
6. Bellemare, M.G., Naddaf, Y., Veness, J., Bowling, M.: The arcade learning environment: an evaluation platform for general agents. *J. Artif. Intell. Res.* **47**, 253–279 (2013)
7. Bengio, Y., Louradour, J., Collobert, R., Weston, J.: Curriculum learning. In: Proceedings of the 26th Annual International Conference on Machine Learning, pp. 41–48 (2009)
8. Bhatt, V., Tjanaka, B., Fontaine, M.C., Nikolaidis, S.: Deep surrogate assisted generation of environments (2022). [arXiv:2206.04199](https://arxiv.org/abs/2206.04199)
9. Bishop, C.M.: Mixture density networks. In: Neural Computing Research Group Report (1994)
10. Bontrager, P., Lin, W., Togelius, J., Risi, S.: Deep interactive evolution. In: Computational Intelligence in Music, Sound, Art and Design: 7th International Conference, EvoMUSART 2018, Parma, Italy, April 4–6, 2018, Proceedings, pp. 267–282. Springer (2018)
11. Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfsagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of monte carlo tree search methods. *IEEE Trans. Comput. Intell. AI Games* **4**(1), 1–43 (2012)
12. Campbell, M., Joseph Hoane, A., Jr., Hsu, F.: Deep blue. *Artif. Intell.* **134**(1–2), 57–83 (2002)
13. Cardamone, L., Yannakakis, G.N., Togelius, J., Luca Lanzi, P.: Evolving interesting maps for a first person shooter. In: Applications of Evolutionary Computation: EvoApplications, pp. 63–72. Springer (2011)
14. Carvelli, C., Grbic, D., Risi, S.: Evolving hypernetworks for game-playing agents. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion, pp. 71–72 (2020)
15. Chen, A., Dohan, D.M., So, D.R.: Evoprompting: language models for code-level neural architecture search (2023). [arXiv:2302.14838](https://arxiv.org/abs/2302.14838)
16. Chrabaszcz, P., Loshchilov, I., Hutter, F.: Back to basics: benchmarking canonical evolution strategies for playing atari. In: IJCAI (2018)
17. Cook, M., Colton, S., Raad, A., Gow, J.: Mechanic miner: reflection-driven game mechanic discovery and level design. In: Applications of Evolutionary Computation: 16th European Conference, EvoApplications 2013, Vienna, Austria, April 3–5, 2013. Proceedings 16, pp. 284–293. Springer (2013)
18. Cuccu, G., Togelius, J., Cudré-Mauroux, P.: Playing atari with six neurons (2018). [arXiv:1806.01363](https://arxiv.org/abs/1806.01363)
19. Dahlskog, S., Togelius, J.: Patterns as objectives for level generation. In: Design Patterns in Games (DPG), Chania, Crete, Greece (2013). ACM Digital Library (2013)
20. Soares de Lima, E., Feijó, B., Furtado, A.L.: Procedural generation of quests for games using genetic algorithms and automated planning. In: SBGames, pp. 144–153 (2019)
21. Dennis, M., Jaques, N., Vinitsky, E., Bayen, A., Russell, S., Critch, A., Levine, S.: Emergent complexity and zero-shot transfer via unsupervised environment design. *Adv. Neural Inf. Process. Syst.* **33**, 13049–13061 (2020)
22. Dharna, A., Togelius, J., Soros, L.B.: Co-generation of game levels and game-playing agents. In: Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, vol. 16, pp. 203–209 (2020)
23. Dockhorn, A., Lucas, S.M., Volz, V., Bravi, I., Gaina, R.D., Perez-Liebana, D.: Learning local forward models on unforgiving games. In: 2019 IEEE Conference on Games (CoG), pp. 1–4. IEEE (2019)
24. Earle, S., Snider, J., Fontaine, M.C., Nikolaidis, S., Togelius, J.: Illuminating diverse neural cellular automata for level generation. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 68–76 (2022)
25. Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K.O., Clune, J.: First return, then explore. *Nature* **590**(7847), 580–586 (2021)

26. Eigenfeldt, A.: Corpus-based recombinant composition using a genetic algorithm. *Soft Comput.* **16**, 2049–2056 (2012)
27. Elsken, T., Hendrik Metzen, J., Hutter, F.: Neural architecture search: a survey. *J Mach. Learn. Res.* **20**(1), 1997–2017 (2019)
28. Fogel, D.B.: *Blondie24: Playing at the Edge of AI*. Morgan Kaufmann (2002)
29. Fontaine, M.C., Liu, R., Khalifa, A., Modi, J., Togelius, J., Hoover, A.K., Nikolaidis, S.: Illuminating mario scenes in the latent space of a generative adversarial network. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, pp. 5922–5930 (2021)
30. Fontaine, M.C., Togelius, J., Nikolaidis, S., Hoover, A.K.: Covariance matrix adaptation for the rapid illumination of behavior space. In: Proceedings of the 2020 Genetic and Evolutionary Computation Conference, pp. 94–102 (2020)
31. Gaina, R.D., Lucas, S.M., Perez-Liebana, D.: Rolling horizon evolution enhancements in general video game playing. In: 2017 IEEE Conference on Computational Intelligence and Games (CIG), pp. 88–95. IEEE (2017)
32. Gallotta, R., Arulkumaran, K., Soros, L.B.: Preference-learning emitters for mixed-initiative quality-diversity algorithms. In: IEEE Transactions on Games, pp. 1–14 (2023)
33. Gauci, J., Stanley, K.O.: Evolving neural networks for geometric game-tree pruning. In: Proceedings of the 13th annual conference on Genetic and evolutionary computation, pp. 379–386 (2011)
34. Genesereth, M., Love, N., Pell, B.: General game playing: overview of the aaai competition. *AI Mag.* **26**(2), 62–62 (2005)
35. Goldberg, D.E.: *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison Wesley (1989)
36. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Bing, X., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial networks. *Commun. ACM* **63**(11), 139–144 (2020)
37. Gravina, D., Liapis, A., Yannakakis, G.: Surprise search: beyond objectives and novelty. *Proc. Genet. Evol. Comput. Conf.* **2016**, 677–684 (2016)
38. Gravina, D., Liapis, A., Yannakakis, G.N.: Constrained surprise search for content generation. In: 2016 IEEE Conference on Computational Intelligence and Games (CIG), pp. 1–8. IEEE (2016)
39. Gravina, D., Liapis, A., Yannakakis, G.N.: Quality diversity through surprise. *IEEE Trans. Evol. Comput.* **23**(4), 603–616 (2018)
40. Guss, W.H., Houghton, B., Topin, N., Wang, P., Codel, C., Veloso, M., Salakhutdinov, R.: Minerl: a large-scale dataset of minecraft demonstrations (2019). [arXiv:1907.13440](https://arxiv.org/abs/1907.13440)
41. Ha, D., Schmidhuber, J.: World models (2018). [arXiv:1803.10122](https://arxiv.org/abs/1803.10122)
42. Hansen, N., Müller, S.D., Koumoutsakos, P.: Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evol. Comput.* **11**(1), 1–18 (2003)
43. Jonathan Hastings, E., Guha, R.K., Stanley, K.O.: Automatic content generation in the galactic arms race video game. *IEEE Trans. Comput. Intell. AI Games* **1**(4), 245–263 (2009)
44. Hecht-Nielsen, R.: Theory of the backpropagation neural network. In: Neural networks for perception, pp. 65–93. Elsevier (1992)
45. Holmgård, C., Liapis, A., Togelius, J., Yannakakis, G.N.: Evolving personas for player decision modeling. In: 2014 IEEE Conference on Computational Intelligence and Games, pp. 1–8. IEEE (2014)
46. Holmgård, C., Liapis, A., Togelius, J., Yannakakis G.N.: Evolving models of player decision making: personas versus clones. *Entertain. Comput.* **16**, 95–104 (2016)
47. Hong, S.-W., Kim, J.-T., Kim, H.-I.: Identification of auto programs by using decision tree learning for mmorpg. *J. Korea Multimed. Soc.* **9**(7), 927–937 (2006)
48. Simon, J.: Artbreeder (2018). <https://www.artbreeder.com/>. Accessed: March 17, 2023
49. Justesen, N., Bontrager, P., Togelius, J., Risi, S.: Deep learning for video game playing. *IEEE Trans. Games* **12**(1), 1–20 (2019)

50. Justesen, N., Mahlmann, T., Togelius, J.: Online evolution for multi-action adversarial games. In: Applications of Evolutionary Computation: 19th European Conference, EvoApplications 2016, Porto, Portugal, March 30–April 1, 2016, Proceedings, Part I 19, pp. 590–603. Springer (2016)
51. Karavolos, D., Liapis, A., Yannakakis, G.N.: Using a surrogate model of gameplay for automated level design. In: 2018 IEEE Conference on Computational Intelligence and Games (CIG), pp. 1–8. IEEE (2018)
52. Kempka, M., Wydmuch, M., Runc, G., Toczek, J., Jaśkowski, W.: Vizdoom: a doom-based ai research platform for visual reinforcement learning. In: 2016 IEEE Conference on Computational Intelligence and Games (CIG), pp. 1–8. IEEE (2016)
53. Khalifa, A., Bontrager, P., Earle, S., Togelius, J.: Pgcrl: procedural content generation via reinforcement learning. Proc. AAAI Conf. Artif. Intell. Interact. Digit. Entertain. **16**, 95–101 (2020)
54. Khalifa, A., Cerny Green, M., Barros, G., Togelius, J.: Intentional computational level design. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 796–803 (2019)
55. Khalifa, A., Togelius, J., Cerny Green, M.: Mutation models: learning to generate levels by imitating evolution. In: Proceedings of the 17th International Conference on the Foundations of Digital Games, pp. 1–9 (2022)
56. Koster, R.: Theory of Fun for Game Design. O'Reilly Media, Inc. (2013)
57. Kramer, M.A.: Nonlinear principal component analysis using autoassociative neural networks. AIChE J **37**(2), 233–243 (1991)
58. Lehman, J., Gordon, J., Jain, S., Ndousse, K., Yeh, C., Stanley, K.O.: Evolution through large models (2022). [arXiv:2206.08896](https://arxiv.org/abs/2206.08896)
59. Lehman, J., Stanley, K.O.: Abandoning objectives: evolution through the search for novelty alone. Evolut. Comput. **19**(2), 189–223 (2011)
60. Lehman, J., Stanley, K.O.: Novelty search and the problem with objectives. In: Genetic Programming Theory and Practice IX, pp. 37–56 (2011)
61. Liapis, A., Martínez, H.P., Togelius, J., Yannakakis, G.N.: Transforming exploratory creativity with delenox (2021). [arXiv:2103.11715](https://arxiv.org/abs/2103.11715)
62. Liapis, A., Yannakakis, G.N., Togelius, J.: Generating map sketches for strategy games. In: Applications of Evolutionary Computation: 16th European Conference, EvoApplications 2013, Vienna, Austria, April 3–5, 2013. Proceedings 16, pp. 264–273. Springer (2013)
63. Liu, B., Zhang, Q., GE Gielen, G.: A gaussian process surrogate model assisted evolutionary algorithm for medium scale expensive optimization problems. IEEE Trans. Evolut. Comput. **18**(2), 180–192 (2013)
64. Melhart, D., Liapis, A., Yannakakis, G.N.: The affect game annotation (again) dataset, pp. arXiv–2104 (2021)
65. Migkotzidis, P., Liapis, A.: Susketch: surrogate models of gameplay as a design assistant. IEEE Trans. Games **14**(2), 273–283 (2021)
66. Minka, T., Cleven, R., Menke, R.: Machine learning for optimal matchmaking (2020). <https://www.youtube.com/watch?v=Q8BX0nXfPjY>
67. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
68. Nolfi, S., Floreano, D.: Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines. MIT press (2000)
69. VAN Olesen, T., Nguyen, D.T.T., Palm, R.B., Risi, S.: Evolutionary planning in latent space. In: Applications of Evolutionary Computation: 24th International Conference, EvoApplications 2021, Held as Part of EvoStar 2021, Virtual Event, April 7–9, 2021, Proceedings 24, pp. 522–536. Springer (2021)
70. Parker-Holder, J., Jiang, M., Dennis, M., Samvelyan, M., Foerster, J., Grefenstette, E., Rocktäschel, T.: Evolving curricula with regret-based environment design. In: International Conference on Machine Learning, pp. 17473–17498. PMLR (2022)

71. Perez, D., Samothrakis, S., Lucas, S., ohlfshagen, P.: Rolling horizon evolution versus tree search for navigation in single-player real-time games. In: Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation, pp. 351–358 (2013)
72. Perez-Liebana, D., Sajid Alam, M., Gaina, R.D.: Rolling horizon neat for general video game playing. In: 2020 IEEE Conference on Games (CoG), pp. 375–382. IEEE (2020)
73. Reisinger, J., Bahceci, E., Karpov, I., Miikkulainen, R.: Coevolving strategies for general game playing. In: 2007 IEEE Symposium on Computational Intelligence and Games, pp. 320–327. IEEE (2007)
74. Jimenez Rezende, D., Mohamed, S., Wierstra, D.: Stochastic backpropagation and approximate inference in deep generative models. In: International Conference on Machine Learning, pp. 1278–1286. PMLR (2014)
75. Risi, S., Lehman, J., D’Ambrosio, D., Hall, R., Stanley, K.: Combining search-based procedural content generation and social gaming in the petalz video game. Proc. AAAI Conf. Artif. Intell. Interact. Digit. Entertain. **8**, 63–68 (2012)
76. Risi, S., Togelius, J.: Neuroevolution in games: state of the art and open challenges. IEEE Trans. Comput. Intell. AI Games **9**(1), 25–41 (2015)
77. Rombach, R., Blattmann, A., Lorenz, D., Esser, P., Ommer, B.: High-resolution image synthesis with latent diffusion models (2021)
78. Rothlauf, F., Rothlauf, F.: Representations for Genetic and Evolutionary Algorithms. Springer (2006)
79. Russell, S.J.: Artificial Intelligence a Modern Approach. Pearson Education, Inc. (2010)
80. Salimans, T., Ho, J., Chen, X., Sidor, S., Sutskever, I.: Evolution strategies as a scalable alternative to reinforcement learning (2017). [arXiv:1703.03864](https://arxiv.org/abs/1703.03864)
81. Samuel, A.: Some studies in machine learning using the game of checkers. Reprinted in Feigenbaum, E.A., Feldman, J. (eds.) (1963). Computers and thought (1959)
82. Schaeffer, J., Burch, N., Bjornsson, Y., Kishimoto, A., Muller, M., Lake, R., Paul, L., Sutphen, S.: Checkers is solved. Science **317**(5844), 1518–1522 (2007)
83. Schrum, J., Capps, B., Steckel, K., Volz, V., Risi, S.: Hybrid encoding for generating large scale game level patterns with local variations. In: IEEE Transactions on Games (2022)
84. Sfikas, K., Liapis, A.: Playing against the board: rolling horizon evolutionary algorithms against pandemic. IEEE Trans. Games **14**(3), 339–349 (2021)
85. Shaker, N., Nicolau, M., Yannakakis, G.N., Togelius, J., O’neill, M.: Evolving levels for super mario bros using grammatical evolution. In: 2012 IEEE Conference on Computational Intelligence and Games (CIG), pp. 304–311. IEEE (2012)
86. Shaker, N., Togelius, J., Nelson, M.J.: Procedural Content Generation in Games. Springer (2016)
87. Shaker, N., Togelius, J., Nelson, M.J., Liapis, A., Smith, G., Shaker, N.: Mixed-initiative content creation. In: Procedural Content Generation in Games, pp. 195–214 (2016)
88. Shaker, N., Yannakakis, G., Togelius, J.: Towards automatic personalized content generation for platform games. Proc. AAAI Conf. Artif. Intell. Interact. Digit. Entertain. **6**, 63–68 (2010)
89. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the game of go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016)
90. Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al.: Mastering the game of go without human knowledge. Nature **550**(7676), 354–359 (2017)
91. Smith, A.M., Lewis, C., Hullet, K., Smith, G., Sullivan, A.: An inclusive view of player modeling. In: Proceedings of the 6th International Conference on Foundations of Digital Games, pp. 301–303 (2011)
92. Sobester, A., Forrester, A., Keane, A.: Engineering Design via Surrogate Modelling: a Practical Guide. Wiley (2008)
93. K.O Stanley. Compositional pattern producing networks: a novel abstraction of development. *Genet. Program. Evolvable Mach.* **8**, 131–162 (2007)

94. Stanley, K.O., Bryant, B.D., Miikkulainen, R.: Real-time neuroevolution in the nero video game. *IEEE Trans. Evolut. Comput.* **9**(6), 653–668 (2005)
95. Stanley, K.O., D’Ambrosio, D.B., Gauci, J.: A hypercube-based encoding for evolving large-scale neural networks. *Artif. Life* **15**(2), 185–212 (2009)
96. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. *Evolut. Comput.* **10**(2), 99–127 (2002)
97. Sudhakaran, S., González-Duque, M., Gianois, C., Freiberger, M., Najarro, E., Risi, S.: Mariogpt: open-ended text2level generation through large language models (2023). [arXiv:2302.05981](https://arxiv.org/abs/2302.05981)
98. Summerville, A., Mateas, M.: Mystical tutor: a magic: the gathering design assistant via denoising sequence-to-sequence learning. *Proc. AAAI Conf. Artif. Intell. Interact. Digit. Entertain.* **12**, 86–92 (2016)
99. Summerville, A., Mateas, M.: Super mario as a string: platformer level generation via lstms (2016). [arXiv:1603.00930](https://arxiv.org/abs/1603.00930)
100. Summerville, A., Snodgrass, S., Guzdial, M., Holmgård, C., Hoover, A.K., Isaksen, A., Nealen, A., Togelius, J.: Procedural content generation via machine learning (pcgml). *IEEE Trans. Games* **10**(3), 257–270 (2018)
101. Tanabe, T., Fukuchi, K., Sakuma, J., Akimoto, Y.: Level generation for angry birds with sequential vae and latent variable evolution. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 1052–1060 (2021)
102. Adaptive Agent Team, Bauer, J., Baumli, K., Baveja, S., Behbahani, F., Bhoopchand, A., Bradley-Schmiege, N., Chang, M., Clay, N., Collister, A., et al.: Human-timescale adaptation in an open-ended task space (2023). [arXiv:2301.07608](https://arxiv.org/abs/2301.07608)
103. Open Ended Learning Team, Stooke, A., Mahajan, A., Barros, C., Deck, C., Bauer, J., Sygnowski, J., Trebacz, M., Jaderberg, M., Mathieu, M., et al.: Open-ended learning leads to generally capable agents (2021). [arXiv:2107.12808](https://arxiv.org/abs/2107.12808)
104. Togelius, J., Karakovskiy, S., Koutnřík, J., Schmidhuber, J.: Super mario evolution. In: 2009 Ieee Symposium on Computational Intelligence and Games, pp. 156–161. IEEE (2009)
105. Togelius, J., Lucas, S.M.: Evolving controllers for simulated car racing. In: 2005 IEEE Congress on Evolutionary Computation, vol. 2, pp. 1906–1913. IEEE (2005)
106. Togelius, J., Lucas, S.M.: Evolving robust and specialized car racing skills. In: 2006 IEEE International Conference on Evolutionary Computation, pp. 1187–1194. IEEE (2006)
107. Togelius, J., Schaul, T., Wierstra, D., Igel, C., Gomez, F., Schmidhuber, J.: Ontogenetic and phylogenetic reinforcement learning. *Künstliche Intelligenz* **23**(3), 30–33 (2009)
108. Togelius, J., Schmidhuber, J.: An experiment in automatic game design. In: 2008 IEEE Symposium On Computational Intelligence and Games, pp. 111–118. IEEE (2008)
109. Togelius, J., Yannakakis, G.N., Stanley, K.O., Browne, C.: Search-based procedural content generation: a taxonomy and survey. *IEEE Trans. Comput. Intell. AI Games* **3**(3), 172–186 (2011)
110. Torrey, L., Shavlik, J.: Transfer learning. In: Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques, pp. 242–264. IGI global (2009)
111. Volz, V., Rudolph, G., Naujoks, B.: Demonstrating the feasibility of automatic game balancing. *Proc. Genet. Evolut. Comput. Conf.* **2016**, 269–276 (2016)
112. Volz, V., Schrum, J., Liu, J., Lucas, S.M., Smith, A., Risi, S.: Evolving mario levels in the latent space of a deep convolutional generative adversarial network. In: Proceedings of the Genetic and Evolutionary Computation Conference, pp. 221–228 (2018)
113. Wang, R., Lehman, J., Clune, J., Stanley, K.O.: Paired open-ended trailblazer (poet): endlessly generating increasingly complex and diverse learning environments and their solutions (2019). [arXiv:1901.01753](https://arxiv.org/abs/1901.01753)
114. Wiens, A.L., Ross, B.J.: Gentropy: evolving 2d textures. *Comput. Graph.* **26**(1), 75–88 (2002)
115. Yannakakis, G.N., Liapis, A., Alexopoulos, C.: Mixed-initiative co-creativity. In: Foundations of Digital Games. ACM (2014)
116. Yannakakis, G.N., Togelius, J.: Artificial Intelligence and Games. Springer (2018)

117. Zaltron, N., Zurlo, L., Risi, S.: Cg-gan: an interactive evolutionary gan-based approach for facial composite generation. *Proc. AAAI Conf. Artif. Intell.* **34**, 2544–2551 (2020)
118. Zhang, Y., Fontaine, M.C., Hoover, A.K., Nikolaidis, S.: Deep surrogate assisted map-elites for automated hearthstone deckbuilding. In: *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 158–167 (2022)

Chapter 26

Evolutionary Machine Learning in the Arts



Jon McCormack

Abstract This chapter looks at artistic and creative applications of evolutionary machine learning. While both evolutionary computing and machine learning techniques have been applied to all kinds of creative and artistic projects, it is more rare to see them used in combination. The chapter will examine the origins and uses of evolution in the arts, before presenting a case study of an evolutionary machine learning artwork. The discussion presents the technical, conceptual, and creative aspects of developing an artwork. The chapter concludes with a discussion on the rise of generative AI and how evolution might contribute to the next wave of artistic possibilities for evolutionary machine learning.

26.1 Introduction

This chapter looks at the applications of Evolutionary Machine Learning (EML) to the creative arts. While evolutionary methods have been used for creative applications independently for well over 30 years, and ML and AI methods for perhaps even longer, it's more difficult to find specific applications of artistic systems that combine both evolution *and* machine learning. In researching this field, it was difficult to find many artistic examples beyond just proof-of-concept ideas or research experiments, so this chapter should not be considered a broad survey of EML artistic or creative systems. Rather, the focus is to examine the conceptual issues and practical implications for using EML in the arts, with a view to how they might be used in the future. Along the way, I'll use one particular system (my own) as an example, not because it is necessarily the best example, but it is one that I'm highly familiar with.

While evolutionary methods have been well explored in the arts for many decades, at the time of writing, large 'foundation models' are what everyone is currently talking about [5]. The rapid rise of deep learning and generative AI has captured the

J. McCormack (✉)
Monash University, Melbourne, Australia
e-mail: jon.mccormack@monash.edu

imagination of many artists and designers. Moreover, the accessibility and power of these technologies has also rapidly increased, making it trivially easy for anyone to use and explore, for example, prompt-based systems that transform a high-level text description of a scene into a photo-realistic image, a video or even 3D scene. In contrast, designing and using an evolutionary system for creative applications still require significant effort, both in terms of the conceptual design and technical realisation. While there are many useful and accessible evolutionary software frameworks, they can't compete with the integrated use of generative text-to-image tools like Stable Diffusion and MidJourney or large language models like OpenAI's ChatGPT or Google's Bard,¹ all of which use simple text interfaces and produce impressive results in seconds. Such ubiquity, ease of use, and minimal effort required by any user of these systems has led some to question if, for example, prompt writing can be considered an artistic practice [24].

The chapter begins with a brief review of the use of evolution in the arts and also a few, select, examples of how machine learning has been used. Next, I'll use my own artistic system, *Eden* as an example of how EML can be used as the basis for an interactive artwork. Following that, I'll move on to looking at the implications of current large generative models and discuss ways in which evolutionary methods might help us use generative models creatively.

26.2 Evolution and Creativity

When considering the creativity we observe in the world, we're confronted by two distinct sources: the creative ability of evolutionary processes, which have 'discovered' things such as flight, vision, language and intelligence, all via a non-teleological process of evolution by natural selection. The second is human intelligence, which has built cities, great works of art and nuclear weapons via the evolved collective intelligence and creativity of our species and our environment. In a basic sense, these two different sources of creativity mirror the approaches taken by researchers in developing artificial creative systems. The first is a 'bottom-up' process of adaptive and emergent complexity, the second is a 'top down' approach by training complex neural networks on large datasets of digitised human culture.

It could be argued that evolution is the more powerful process, as it is responsible for all life on earth, including our own species and our highly capable brains and bodies. In this section, I'll look at why evolutionary methods might remain the best hope for creating *new* creative systems and forms of artistic creativity.

The application of evolutionary computing methods to creative art and design began in 1986 [17], when evolutionary biologist Richard Dawkins' book, *The Blind Watchmaker* included a description of software that evolved simple, two-dimensional

¹ As preparation for writing this chapter I asked both ChatGPT and Bard to provide an academic report on the applications of EML to the arts. Unfortunately, neither could provide any useful nor accurate information on this topic.

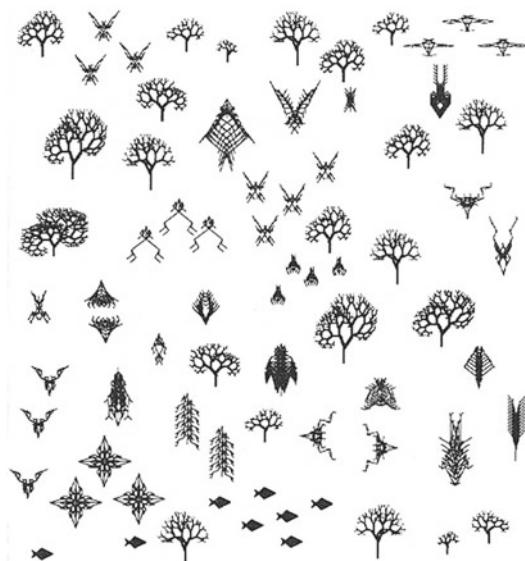
stick-like shapes drawn by the computer, known as ‘Biomorphs’ [10]. The previous evolutionary computing methods required a computer-representable function that could evaluate the suitability or *fitness* of individuals in a population, allowing them to be ranked, then the highest ranked (most fit) individuals used for breeding. Attempting to craft a computer function that measures concepts like aesthetics, beauty or even ‘plant or animal likeness in stick figure drawings’ was (and, in general, remains) an exceedingly difficult problem [23]. Dawkins circumvented the problem of devising such a machine-representable fitness function by getting the human user to perform fitness evaluation for each design at each generation in the evolutionary loop. His goal was to demonstrate the power of evolution as a design algorithm, one that could design complexity without the need for an explicit designer (the human user simply evaluates the suitability of individuals at each generation). The results surprised even Dawkins’ himself, far surpassing his expectations for what such a simple system could produce:

Nothing in my biologist’s intuition, nothing in my 20 years’ experience of programming computers, and nothing in my wildest dreams, prepared me for what actually emerged on the screen...

— Richard Dawkins [10]

What surprised Dawkins was the variety of forms he was able to evolve, many reminiscent of real biology, at least as much as could be conveyed in a two-dimensional stick representation (Fig. 26.1). These forms resembled trees, insects, fish and birds. The fact that such a variety of natural-looking forms could be evolved was due,

Fig. 26.1 Some example forms evolved by Dawkins using his *Biomorph* software



in part, to Dawkins' experience as a Biologist – he was very familiar with the morphology of a wide variety of species, recognising traits like lateral symmetry and self-similarity, and built these into the Biomorph generative system that evolution acted upon.

It did not take long for people interested in creativity and art to grasp the significance of this idea, and how it might be used to create a new kind of art and design: one that was evolved rather than directly created by human hands and minds. In theory and in practice, this allowed a human designer to be freed from the limits of their own imagination, permitting the generation of designs that could not be directly conceived in the mind or created by the artist alone. Evolutionary computing finally broke the so-called ‘dictum of Descartes’: that no designer can create a system that outperforms the knowledge and imagination of its designer [2].

The early adopters of Dawkins’ ‘Blind Watchmaker’ process (now known as the Interactive Genetic Algorithm or IGA) were Karl Sims in the USA [30] and William Latham and Stephen Todd in the UK [36, 37]. Latham and Todd developed new computer-aided design software, called ‘Mutator’, based on ideas Latham explored while studying at the Royal College of Art in London where he manually executed an evolutionary process to create a complex evolutionary tree of hand-drawn 3D form (Fig. 26.2). Mutator was used to evolve 3D forms via human-evaluated aesthetic selection in the computer. In the early 1990s, Latham created a series of other-worldly, organic, surreal virtual sculptures and animated films based on the idea of evolving form. At the same time, Sims used Dakwins’ technique to evolve dynamic systems, Lisp expression images and exotic plant-like structures. Along with Latham,

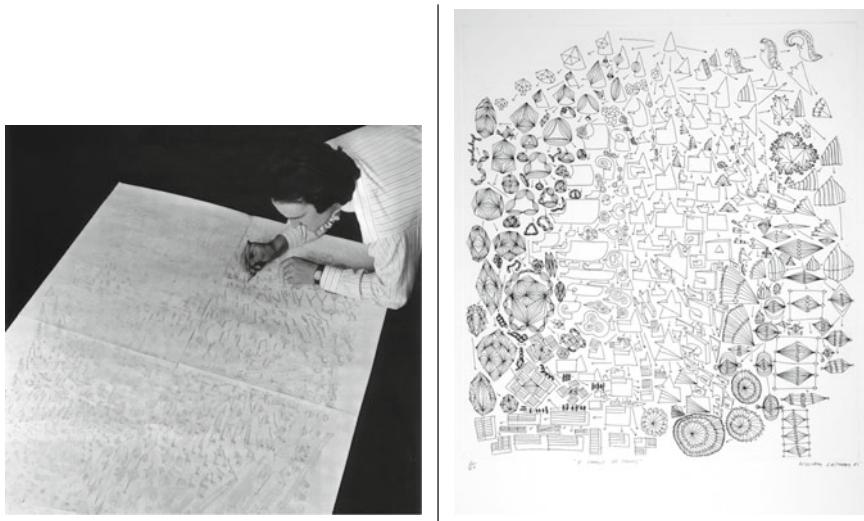


Fig. 26.2 William Latham evolving hand-drawn forms in 1989 (left) and a drawn example of the process (right)

Sims' seminal animations *Panspermia* (1990) and *Primordial Dance* (1991) demonstrated the potential for evolution using human-evaluated aesthetic measures to offer genuinely new possibilities for human–computer creativity [31]. Experiments with evolutionary music began at a similar time with a system devised in 1991 by Andrew Horner and David Goldberg used genetic algorithms for thematic bridging [16]. Al Biles' famous 'GenJam' (1994) software used evolutionary methods to create an automated jazz accompanist that improvised with its human creator [1].

In the years that followed, many more artists and researchers became interested in using evolutionary methods. Sims went on to apply evolutionary methods to evolve 'virtual creatures' where both the physical form and behaviour were evolved simultaneously, akin to evolving body and mind, mimicking (in a highly abstracted and simplistic sense) the evolution of life on Earth (probably one of the first applications of EML to creativity). The system also incorporated a Newtonian physics simulation and physical environment, giving the creatures physically realistic movements and behaviours. To Sims' surprise, his system was able to evolve a series of viable forms and behaviours, many highly reminiscent of real animals, including snake-like shapes that swam towards a light, legged figures that evolved to walk or move using a variety of different locomotive strategies such as jumping or walking, even creatures that learned to battle for possession of an object, some devising strategies such as pushing their opponent away or covering the object with their large 'arms' to prevent an opponent making contact. Even today, 30 years on, the work remains one of the leading examples of how evolution in a computer can be used to generate new and complex form and behaviour [32].

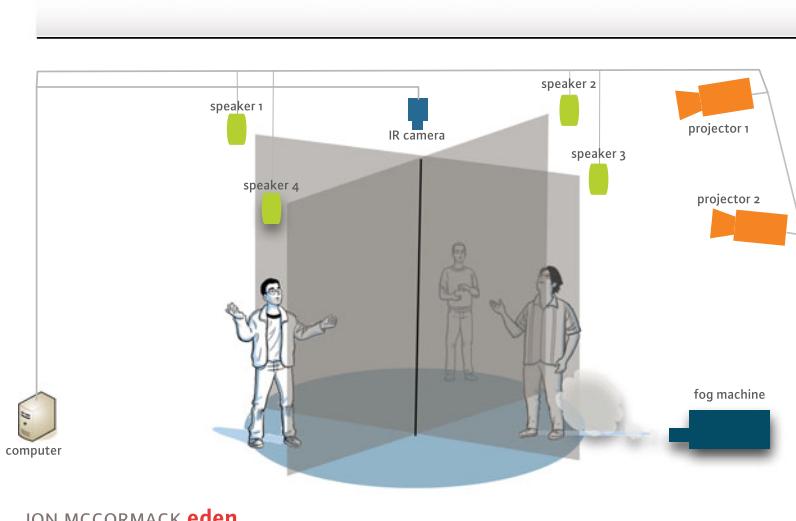


Fig. 26.3 An isometric diagram of the physical layout of *Eden*, showing the use of transparent screens, video projectors, camera sensors and speakers

The combination of EC and ANNs in the visual domain dates to 1994, where interestingly deep models were used [3]. The competitive concept found in Generative Adversarial Networks (GANs) is predated by competitive co-evolution, an early example is Todd and Werner's *Composer/Critic* system for music composition [35]. Machado and colleagues also worked with competitive co-evolution between EC and ANN models [19].

These early and pioneering forays into the creative applications of evolution and EML heralded much interest and excitement, both for artists, researchers, even philosophers and cultural theorists. Many imagined novel and exciting possibilities for art and design, motivated by the technological optimism evangelically expressed in features such as Moore's Law (the exponential growth in the number of transistors in integrated circuits). However, as researchers and artists began to look more closely at these systems, over the coming years their limitations became apparent. 'Interactive' evolution circumvented the problem of devising a fitness representation for subjective criteria such as aesthetics, but the repetitive human evaluation of fitness was more akin to a process of selective breeding than real biological evolution. Users of these systems suffered from selection 'fatigue', assuming a passive and simplistic role in the design process, limiting the results that could be achieved. In specialist domains, it was shown that interactive evolution was more valuable to non-experts than experts, for expert users it was generally counter-productive [34]. The bold predictions of technological advancement did not eventuate.²

While evolution continues to be a useful method, it has also found scepticism in the art world. Such scepticism is summed up by media theorist Jussi Parikka in a discussion on the use of electronic art made with genetic algorithms [28]:

if one looks at several of the art pieces made with genetic algorithms, one gets quickly a feeling of not 'nature at work' but a Designer that after a while starts to repeat himself. There seems to be a teleology anyhow incorporated into the supposed forces of nature expressed in genetic algorithms.

Parikka's point is that art made using evolutionary techniques appears to reflect more of the human designer's intention than that of open-ended evolution. With techniques such as the IGA often described as 'pigeon breeding' it is unsurprising that the results reflect mainly the aesthetic preferences of the human fitness evaluator. Moreover, what any evolutionary system can produce is limited by both the mechanisms that generate phenotype from genotype, and a person's ability to find or discover any specific phenotype. Clearly, organic chemistry is a powerful generative system—capable of building a vast array of all the different forms of life that have existed on Earth. What can be simulated in a computer remains far less capable.

Nonetheless, biological evolution remains a powerful and deep explanatory mechanism, an idea that suggests many interesting possibilities for art and creativity. These possibilities include the self-organisational properties of matter, the emergent properties of bottom-up systems, or the cybernetic concept of *open-ended behaviour* as

² With the advent of new technological fields, such as Artificial Intelligence or Artificial Life, it is common for leading proponents to severely underestimate the difficulty in reaching goals of human-like intelligence or the capabilities of real biological life.

expressed in Descartes Dictum, discussed earlier [2]. One (obvious) aspect often overlooked by technologically focused research is that computers are *not* biological. Computing methodologies are primarily *process-focused*, because computers must be programmed using algorithms (recipes for processes). This necessarily requires abstracting the physical and forcing *representations* in software, due to software's representational limitations. The field of Artificial Life, for example, is built on the premise that life can be defined by process and hence is not specific to any particular physical materialisation. So, in this view, life could be carbon-based like the life on Earth, or it might be silicon-based and running on a computer. These issues have already been well discussed and relate to broader philosophical issues of embodiment, representation, semiotics and physical ontology. For now, I'll just flag it as a problem for the reader to keep in mind as we further explore the creative possibilities of EML.

To that end, in the next section, I present a case study of a specific EML art project, with a view to both laying out the technical, conceptual and creative issues that EML works might address.

26.3 Eden: A Case Study

As an example of how evolutionary machine learning can be used in the arts, I'll give a detailed description—both technical and conceptual—of my artwork, *Eden*, which uses EML methods. The work is interactive, in that human presence and behaviour influence the work's appearance, sound and behaviour. Conceptually, it is an evolutionary ecosystem of virtual 'creatures' that develop a symbiotic relationship with the work's human audience as the system learns and evolves. I began working on this project in the early 2000s and it went through a number of iterations before I arrived at the version described here. Various improvements and updates were made up until 2017 when the work was acquired by a museum. *Eden* has been exhibited numerous times at museums, galleries and festivals in Australia, Europe, the UK, Russia, South and North America and Asia and has received a number of awards, including the John Lansdown Prize for interactive art.³ I list these achievements, not for the sake of self-promotion, but as evidence that the work itself has been recognised for its technical, creative and artistic significance.

The goal of *Eden* was to create an open-ended artistic system that is adaptive to its environment (in this case—the art gallery environment, see Fig. 26.4). In order to address this goal, two important questions were explored during the design and development of the work: first, is it possible to create a virtual world that evolves towards some subjective criteria of the audience experiencing it, without the audience needing to explicitly perform fitness selection and, second, how can the relationship

³ Lansdown was a pioneering British computer graphics and art polymath. The prize, named in his honour, is awarded by the Eurographics Association in recognition of significant contributions to computer graphics and interactivity.

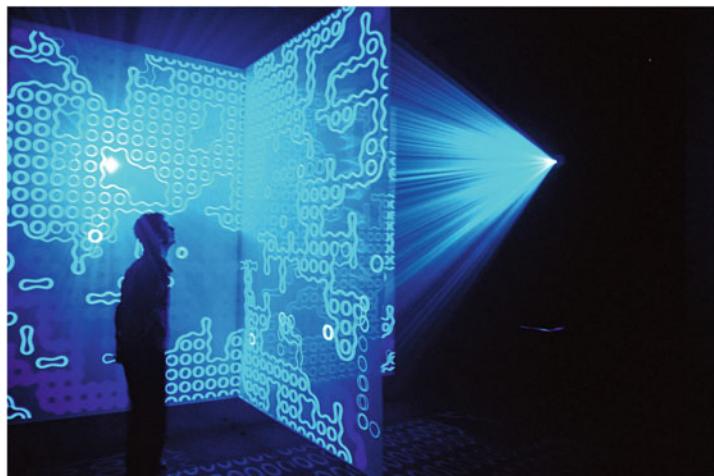


Fig. 26.4 Installation view of *Eden* at the Australian Centre for the Moving Image in 2004

between real space and virtual simulation be realised in such a way that integrates those two in a phenomenological sense?

As a software system *Eden* is an evolutionary simulation that operates over a cellular lattice, inhabited by agents ('creatures') who have, among other capabilities, the ability to make and hear sound.⁴ Creatures use an internal, evolving classifier system to control their behaviour in the world. The virtual environment that the creatures inhabit develops in response to the presence and movement of people experiencing the system as an artwork. The work is conceptualised and designed as a symbiotic *artificial ecosystem* where virtual species interact with the human audience. It is designed as an experiential environment, whereby viewer's participation and activity within the physical space have important consequences over the development of the virtual environment.

26.3.1 Related Work and Inspiration

Conceptually, the work was inspired by a period of time I spent in one of Australia's most remote national parks (Litchfield National Park in the Northern Territory of Australia). Observing such a diverse and exotic variety of flora and fauna in a remote wilderness area over a period of several weeks, I drew inspiration from the collective intelligence of insects and how they fitted into the landscape. So one might think of this work as a contemporary form of landscape painting, but with the significant

⁴ For convenience I'll use terms like 'creature', 'see', 'hear', 'know' and 'learn' in this section, but the reader is reminded that these terms don't imply an anthropomorphic interpretation.

difference that the medium I am ‘painting’ with is *process* rather than physical paint on a canvas. The work is not meant to be a visual representation of a real landscape or ecosystem, rather it aims to capture the phenomenological sense of process in natural systems and environments.

Eden draws its technical inspiration from John Holland’s *Echo* [15], particularly in the use of classifier systems for the internal decision-making system of agents. Others have used evolutionary systems as a basis for musical composition, but in the main for composition *simulation* [35, 40], rather than as a new form of creative tool for the artist and audience.

The *Living Melodies* system [12] used a genetic programming framework to evolve an ecosystem of musical creatures that communicate using sound. *Living Melodies* assumes that all agents have an innate ‘listening pleasure’ that encourages them to make noise to increase their survival chances. *Eden*, contains no such inducement, beyond the fact that some sonic communication strategies that creatures discover should offer a survival or mating advantage. This results in the observation that only some instances of evolution in *Eden* result in the use of sonic communication, whereas, in *Living Melodies*, every instance does. *Living Melodies* restricts its focus to music composition, whereas *Eden* is both a sound and visual experience.

26.3.2 Physical Installation

The artwork is exhibited as an installation, around 6 m × 6 m, that can be experienced by multiple people simultaneously. It consists of two semi-transparent projection screens arranged in an ‘X’ shape, upon which the virtual environment is projected (via two video projectors). The ambient lighting is minimal—making the screens and the light they reflect and transmit the predominant source of visual interest in the space. A fog machine helps emphasise the three-dimensional nature of the projections, extending the simulated virtual world deeply into the installation space.

Multiple speakers are arranged around the screens to spatialise the sound generated from the simulation according to the source location. So, for example, when a specific creature makes a noise, it will appear to the human audience to emanate from the creature’s current location in physical space.

In addition to this audio-visual infrastructure, an infrared digital video camera is placed above the screens, looking down at the space surrounding the projection screens. This area is illuminated by an infrared lighting system, which is invisible to the human eye and so does not affect the perceptual properties of the work. The camera system identifies and tracks the position and movement of people in the immediate vicinity of the artwork. It is not necessary that the audience has any direct knowledge of this sensing. The purpose of identifying the location and tracking of the work’s human audience is an environmental stimulus for the virtual world, which ultimately contributes to selection pressures that encourage a symbiotic relationship between people experiencing the work and the creatures populating the virtual world.

26.3.3 Technical Implementation

I will now discuss the main technical details of the system, with particular emphasis on the EML mechanisms that facilitate behaviour and learning of virtual creatures. Further details, particularly the *payoff* and *bidding* processes for rule selection, may be found in [21, 22].

The environment projected onto the screens is the *Eden world*, made of a two-dimensional toroidal cellular lattice that develops using a global, discrete time-step model—a popular model based on the theory of cellular automata [9, 38]. Each cell in the lattice may be populated by one of the following entities:

- *Rock*: inert matter that is impervious to other entities and opaque to sound and light. Rock is placed in cells at initialisation time using a variation of the *diffusion limited aggregation* (DLA) model [42]. Rocks provide refuge and contribute to more interesting spatial environmental behaviour of the agents.
- *Biomass*: a food source for creatures. Biomass grows in yearly⁵ cycles based on a simple feedback model, similar to that of *Daisyworld* [39]. Radiant energy (in ‘infinite’ supply) drives the growth of biomass. The amount of radiant energy falling on a particular cell is dependent on a number of factors, including the local absorption rate of the biomass and global seasonal variation. Probabilistic parameters can be specified at initialisation time to control these rates and variations. The efficiency at which the biomass converts radiant energy into more biomass is also dependent on the presence of people in the real space of the artwork. This dependency is detailed in Sect. 26.3.5.
- *Creatures*: mobile agents with an internal, evolvable classifier system, based on XCS [41]. Agents obtain energy by eating biomass or by killing and eating other agents. More than one agent may occupy a single cell.

Each creature consists of a set of *sensors*, an evolving learning classifier system, and a set of *actuators*. This configuration is illustrated in Fig. 26.5. Sensors provide measurement of the environment and internal introspection of an individual agent’s status. The XCS relates input messages from the sensors to desired actions. A creature learns through environmental reward, which causes changes in the actions that are performed in response to sensory input. The actuators are used to carry out actions in the world. The success or failure of an intended action will be dependent on the physical constraints in operation at the time and place the intent is instigated.

At initialisation of the world, a number of creatures are seeded into the population. Each agent maintains internal state, which includes the following:

- *Current age*, an integer measured in time steps since birth. Agents live up to 100 years and cannot mate in their first year of life.
- *Health index*: an integer value indicating the overall health of the agent. A value of 100 indicates perfect health; if the health index falls to 0, the agent dies. An agent can lose health via a sustained negative *energy level* differential (explained

⁵ An *Eden* year lasts 600 *Eden* days, but passes by in about 10 minutes of real time.

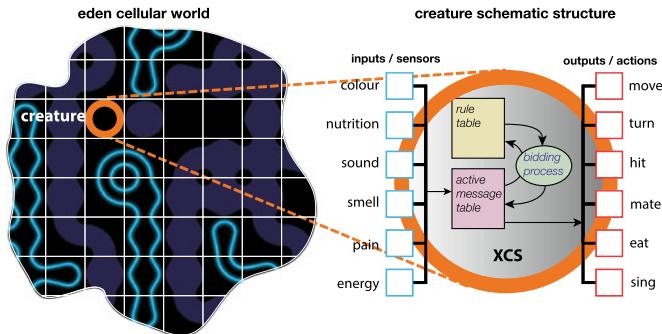


Fig. 26.5 A section of the *Eden* cellular lattice in visual form (left). To emphasise the lattice structure, grid lines have been layered over the image. The image shows rocks (solid), biomass (outline) and an agent (thick circle). The diagram (right) shows the agent’s internal schematic structure, consisting of a number of sensors, a performance system that evolves and a set of actuators

next) by bumping into solid objects, such as rocks, or being hit by other agents. In addition, the loss in health from being hit by another agent depends on both its mass and health index.

- **Energy level:** a measure of the amount of energy the agent currently has. Agents gain energy by eating biomass or other agents. Energy is expended attempting to perform actions (regardless of their success); a small quantity of energy is expended even if no action is performed at a given time step. If an agent’s energy level falls to zero, the agent dies and its body is converted to new biomass in the cell in which it died.
- **Mass:** an agent’s mass is linearly proportional to its energy level, plus an initial ‘birth mass’ that is normally distributed over the population.

26.3.3.1 Sensors

Sensors provide a way for a creature to introspect on itself and its environment [29]. A creature can use a range of sensor types, but the sensors themselves do not undergo any evolution and are fixed in function, sensitivity and morphology. It is up to an individual creature’s learning system to make use of a sensor, so data from a particular sensor will only be used in the long term if it provides useful information that assists the agent’s survival or mating prospects. Sensor use does not incur any cost to the agent. Sensor information available to an agent consists of:

- A simple vision system that detects the ‘colour’ of objects on facing and neighbouring cells (the range is limited to a single cell). Rocks, biomass and agents all have different ‘colours’, which enables an agent to distinguish between them.

- A sensor to detect the local cell nutritional value. Cells that contain biomass or dead agents have a high nutritional value, rocks and empty cells do not.
- A sound sensor that detects sound pressure levels over a range of frequency bands. Sound can be detected over a much larger spatial range than vision and also with greater fidelity.
- An introspection of *pain*. Pain corresponds to a negative health index differential and would usually indicate attack by another agent or that the agent is bumping into rocks.
- An introspection of the current energy level.

26.3.3.2 Actuators

Actuators are used to signal a creature's intent to carry out an action in the world. The physical laws of the world will determine whether the intended action can be carried out or not. For example, the creature may intend to 'walk forward one cell', but if that cell contains a rock, the action will not be possible. Furthermore, all actions cost energy, the amount dependent on the type of action and its context (e.g. attempting to walk into a rock will cost more energy than walking into an empty cell).

As with the sensors, the number and function of actuators are fixed and do not change as the system evolves. Actions will only be used in the long term if they provide benefit. Analysis of actions used by creatures who are successful in surviving shows that not all make use of the full set of actuators.

At each timestep, creatures may perform any of the following actions:

- *Move* forward in the current direction.
- *Turn* left or right.
- *Hit* whatever else is in the cell occupied by the agent. Hitting another agent reduces that agent's health level using a non-linear combination of the mass, health and energy level of the agent performing the hit. Hitting other objects or being hit will cause pain and a loss of health.
- *Mate* with whatever is currently occupying the current cell. Obviously, this is only useful if another agent is in the same cell. Mating is only possible if the age of both agents is greater than 1 year.
- *Eat* whatever is occupying the current cell. Creatures can only eat biomass or dead creatures (which turn into biomass shortly after death).
- *Sing*: make a sound that can be heard by other agents.

Performing an action costs energy, so creatures quickly learn not to perform actions without benefit. For example, attempting to eat when your nutritional sensor is not activated has a cost but no benefit. Attempting to move into a rock has a cost greater than moving into an empty cell.

A creature may also choose not to perform any action at a given time step (a 'do nothing' action), but even this costs energy, although less than any other action.

26.3.4 XCS Classifier System

Each creature learns via a variant of Wilson's eXtended Classifier System (XCS) [41]. Classifier systems are rule-based systems that translate input sensory data into actions based on a set of competing rules. Rules have varying degrees of generality, allowing a creature to deal with highly specific or general situations effectively. New rules enter the system when no current rule matches an input stimulus, or via an evolutionary process where existing rules are subject to the standard genetic operators of mutation and crossover. Over time a creature learns how to live in its environment using a combination of the XCS learning system and a simulation of natural selection—only those creatures that have learnt to survive in the virtual world will end up being successful and having offspring.

26.3.4.1 Evolution

An environmental reward system allows rules that have contributed to the agent's successful survival to be used more often and for learning to occur over an agent's lifetime.

Evolutionary algorithms generally follow a Darwinian metaphor in that phenotypes pass their genetic information onto offspring, via the processes of crossover and mutation. An implicit or explicit fitness evaluation determines which members of a population survive and hence, which genes (rules) become dominant in the population over many generations.

Rather than adopting this standard evolutionary metaphor, *Eden* uses a form of Lamarkian evolution [6], where a creature passes on the rules it has learnt during its lifetime to its offspring at birth. Rules are selected proportional to their previous success, so only the most successful rules end up staying in the population over generations.

This design decision was used to allow more rapid adaptation to changing environmental conditions: a necessary feature if the creatures in the artificial ecosystem are to adapt to the behaviour of people experiencing the work in real time. Another way to consider this approach is that parents teach their offspring all the good things they have learnt so far in their lifetime—a kind of social learning—but in *Eden* this happens instantly at the moment of birth.

26.3.5 Interaction

The *Eden* system has a unique relationship between the physical and virtual components of the system. As shown in Fig. 26.3, an infrared video camera and machine vision system monitors the space of the installation environment, recognising the presence and movement of people in the space. A video digitisation sub-system per-

forms basic image processing and analysis, such as background removal and feature detection. This data is converted into a stream of vectors indicating the location and movement of individuals in the exhibition area, and used to drive environmental parameters in the virtual simulation.

Eden has no explicit fitness function. Agents continue to be part of the system based on how well they can survive and mate in the current environment. If certain selection pressures are applied, such as food becoming scarce, only those agents who can adapt and find food will prosper. By driving environmental conditions from the presence and movement of people in the exhibition space, agents must implicitly adapt to an environment that includes aspects of the world outside of the simulation.

To achieve this, two mappings were used:

- *Presence in the real environment maps to biomass growth rates.* The presence of people around the screen area affects the rate of biomass growth in that local area of the simulated world. Areas with no people correspond to a barren environment: little biomass will grow without the presence of people in the gallery environment.
- *Movement in the real environment maps to genotype mutation rates.* The greater the movement of people in the space, the higher the mutation rate for rule evolution.

These mappings were based on some assumptions. First, people will generally spend time experiencing something only if it interests them. In the context of experiencing an artwork, people generally may spend a short time evaluating their interest in an artwork, but, after a short time, if it no longer interests them, they will leave. There may be other reasons for leaving, but in general, the duration of stay will have some relation to how ‘interesting’ the experience is.

Agents require food to survive. If people are in the real environment, then food will grow at a more rapid rate. An agent who is making ‘interesting’ noises, for instance, would have a better chance of keeping a person’s attention than one who is not. Moreover, an agent making a progression of sounds, rather than a just a single, repeating sound, is likely to hold a person’s attention even longer. Agents who encourage and hold a person’s attention in the space implicitly give the environment around that agent a more plentiful food supply.

The mapping of people’s movement in the space to mutation rates is based on the assumption that people will move over an area looking for something that interests them and, when they find it, will stay relatively still and observe it. Hence, the movement of people within the real space serves to inject ‘noise’ into the genome of agents who are close to the source of movement. Higher mutation rates result in more variation of rules.⁶ If an agent or group of agents are holding the viewer’s attention, then less rule discovery is needed in the current environment, whereas, if people are continually moving, looking for something ‘interesting’, this will aid in the generation of new rules.

⁶ Most child rules that mutate will not be ‘better’ than the parent rule, but, in general, the use of mutation does provide the possibility for the system to discover rules that would not be possible by crossover alone.

26.3.6 *Observations*

As mentioned at the beginning of this section, the work has been exhibited extensively, which has allowed the collection of much anecdotal evidence about the system and how people interact with it. Early on I realised that certain factors have a marked effect on people's behaviour and require specific compensations in the software. For example, when the gallery is closed, there will be no people in the space, which diminishes the food supply for the creatures. Without compensation for gallery opening hours, the entire population dies out each night! The general flow and number of visitors varies according to location and time of day, so the system tries to compensate for this through careful tuning of the system parameters.

Analysis of the rules agents use shows that sound is often used to assist in mating, as would be expected, and with the influence of people, sound is used in other ways as well. Once the environmental pressures from audience behaviour are incorporated into the system, the generation of sound shows a marked increase and analysis of the rules discovered shows that making sound is not only used for mating purposes.

The resulting sounds produced by the system *do* appear to interest and engage the audience, and combined with the visual experience, *Eden* does incite, from visitors, an interest and curiosity in what it is doing. In many cases, people are not aware of the learning system, camera sensing or even the fact that what they are experiencing is a complex virtual ecosystem. This does not necessarily hinder the experience, as there is no correct or incorrect way to behave except to appreciate the experience. Anecdotal accounts from people who have experienced the work describe it as 'having a sense that it is somehow alive', or 'like being in a strange forest at night'. In a number of exhibitions, people returned to the work over a period of several days, to see how the qualitative behaviour of the virtual environment had changed. In one exhibition, a local businessman visited the work during his lunch hour every day for 3 weeks, describing the experience as 'fascinating...one that made me more sensitive to my own environment'. While these are, of course, subjective evaluations, it does appear that *Eden* is able to adapt and evolve to create an ongoing interest for its audience.

Having now gone into this detailed example of EML in an artistic project, the next section looks at the current fascination with generative AI and how evolutionary systems might expand the possibilities for generative AI in the arts.

26.4 The Rapid Rise of Generative AI Art

In recent years, generative AI has increasingly become a central concern for art that is created with technology. AI methods have been used for artistic applications at least since the 1960s [7]. For example, one of the most prominent and dedicated artists working with AI was Harold Cohen (1928–2016), who devoted the majority of his professional career to trying to uncover the creative possibilities of AI, primarily using software he referred to as AARON. The AI methods used by Cohen were

very simple and primitive by today's standards, but his skills and experience as a conventional painter played an important role in the software's development and the works he developed. While Cohen's work was certainly well known amongst people working in the computer arts, it went largely unrecognised by the broader art world.

Things changed dramatically in October 2018. AI-generated art made headlines around the world when a 'work of art created by an algorithm' was sold at auction by Christie's for US\$432,500—more than 40 times the value estimated before auction [8]. The work, titled *Portrait of Edmond Belamy* was one of 'a group of portraits of the fictional Belamy family'⁷ created by the Paris-based collective *Obvious*.

The three members of Obvious had backgrounds in Machine Learning, Business and Economics. They had no established or serious history as artists. Their reasoning for producing the works was to create artworks 'in a very accessible way (portraits framed that look like something you can find in a museum)' with the expectation of giving 'a view of what is possible with these algorithms'.[27]

The works' production involved the use of Generative Adversarial Networks (GANs), a technique developed by Ian Goodfellow and colleagues at the University of Montreal in 2014 [14]. GANs became a technology enthusiastically adopted by a number of artists, heralding a new era of 'AI Art'. Such was the hyped enthusiasm for 'GAN Art' that machine learning researcher, Francois Chollet proposed the term *GANism*,⁸ indicating that the method's distinctive aesthetics were significant enough to make 'GAN Art' worthy of being compared alongside major movements in the history of art, speculating that it might 'become a significant modern art trend'.

In her book on AI Art, Joanna Zylinska took a more critical view of the GAN Art 'movement':

Through both their art and their discourse on art, Kogan, Tyka, Akten and Klingemann adopt a worryingly uncritical instrumentalism, where seemingly child-like curiosity is underpinned by the progressivist model of technological expansion towards some kind of improvement - of accuracy, data sets and, ultimately, art as we know it. They are thus poster boys for AI art as underwritten by Google in the way they incarnate the very ideas of progress, innovation and upward trajectory that drive the gung-ho progressivism of the AI 2.0 era.

Zylinska picked up on what she referred to as the 'gen-art boys' discourse', arguing that 'dreamy neural network art of this kind thus ultimately has a pacifying effect, anaesthetising us into the perception of banal sameness that creates an illusion of diversification without being able to account for differences that matter—for why and where they matter, when and to whom. It thus ends up enforcing a mode of existence that philosopher Franco 'Bifo' Berardi has called a 'neurototalitarianism' [43].

With the rise of text-to-image systems such as DALL-E 2, MidJourney and Stable Diffusion, which allow the generation of detailed and complex imagery from short text descriptions, 'GAN Art' rapidly faded into technological and creative oblivion, superseded by newer generative AI techniques trained on a vast corpus of human visual culture. These Text-to-Image (TTI) systems allow anyone to write a brief

⁷ The name is derived from the French interpretation of 'Goodfellow': *Bel ami*.

⁸ <https://twitter.com/fchollet/status/885378870848901120>.



Fig. 26.6 The progress of Generative AI. Left: an image generated using a Generative Adversarial Network (GAN) similar to the ‘Portrait of Edmund Belamy’ which sold for US\$432,500 in 2018; and Right: an image created in less than 20 seconds using *MidJourney* software in 2022 with the text prompt ‘19th century portrait of a French nobleman’

English prompt and have the system respond with a series of images that depict the scene described in the prompt.

As shown in Fig. 26.6, in the space of a few short years, generative AI has progressed from blurry and vague visual depictions defined by numerous technical artefacts to images with a quality that rivals highly competent human artists and illustrators. And as the technology evangelists keep reminding us, ‘AI is currently the worst it will ever be’.

This example is illustrative for several reasons. Firstly, it demonstrates the rapid pace of technological development of generative AI systems—in just a few years the quality and capability of generative AI systems has increased significantly. Secondly, from the perspective of Art, it demonstrates how a reliance exclusively on a technology does not make for a significant creative practice or define an ‘art movement’. The price paid for *Portrait of Edmond Belamy* now seems even more excessive than it did at the time,⁹ given how outdated the technology has so quickly become.

In the earliest days of using computers for art, artists were also programmers, necessarily so because anyone using a computer *had to program it* to get it to do anything. As software developed, both technically and culturally, there was a shift to using software, written by others, with software evolving to become more like an artist’s tool rather than something of their own creation. With movements such as generative art, algorithmic art and software art, the emphasis on creative coding as an artistic practice remains high, but much of the current efforts in ‘AI Art’ require the use of software written by others, over which the artist may have minimal understanding or direct control.

⁹ It’s worth noting that no other AI artwork has since achieved anything near a price at auction.

It has long been discussed how the tools used shape both the limitations and outcomes of a creative practice. Tools themselves influence how one thinks about making with that tool and therefore, what one makes and *how* it gets made. A tool as simple as a pencil has enormous and varied expressive power, even though anything drawn with a pencil looks like a ‘pencil drawing’ as a class of image making. With generative AI, there are important differences because the level of direct control is removed and important creative features, such as intent, embodiment and expressive capability are either non-existent or severely diminished. Moreover, generative AI systems are trained on a vast corpus of human art and can only ever generate statistical mixups of prior human art.

So while TTI systems, for example, can generate ‘new’ images (in the sense that the exact specific image has not existed before), they are all statistical amalgamations of pre-existing images, precluding the possibility of stylistic innovation, or any conceptual innovation. In the terminology of Boden [4], they support only combinatorial creativity (the combination of existing elements), not transformational creativity (where elements are transformed into something new). In simple terms, TTI systems could not ‘invent’ Cubism if they were only trained on data prior to 1908, yet this was possible for human artists.

So while many people are now using generative AI to make ‘art’ using TTI software like Stable Diffusion and MidJourney, I would argue that the majority of the creativity in this process rests with the data on which the AI was trained, i.e. from prior human creativity. MidJourney, for example, allows you to input an image and get an AI to describe it for you as a ‘prompt’ (Fig. 26.7). Notice that the generated text prompt often includes the phrase ‘in the style of’. While some styles are general across a number of individual artists (‘postmodern photomontage’), others are specific to an individual (‘larry carlson’), what I have dubbed *style theft*, where you can request the system generates an image that mimics the style of a specific artist, without that artist’s permission or blessing.



1. an amorphous sunflower with yellow and purple petals, in the style of postmodern photomontage, spiky mounds, made of insects, ps1 graphics, green and crimson, canon af35m, tangled nests –ar 4:3

2. an orange dandelion with purple spikes, in the style of postmodern photomontage, bryce 3d, tangled nests, dinocore, group f/64, photo taken with ektachrome, exotic flora and fauna –ar 4:3

3. these flowers are yellow and black, in the style of larry carlson, spiky mounds, angus mckie, colorized, made of insects, octane render, bugcore –ar 4:3

4. a group of yellow flowers with purple leaves, in the style of larry carlson, tangled nests, angus mckie, cinestill 50d, neogeo, spiky mounds, made of insects –ar 4:3

Fig. 26.7 An example of MidJourney’s image-to-prompt AI: At left an original image (an evolved virtual sunflower, a still frame from the evolutionary artwork *Turbulence* by the author [20]). Right, the four text prompts The –ar 4:3 at the end of each prompt refers to the image aspect ratio



Fig. 26.8 Example images generated by the four text prompts shown in Fig. 26.7

Prompts generated in this can then be fed into MidJourney’s text-to-image system, synthesising images from the machine-generated prompt. Figure 26.8 shows an example image generated by each of the four generated text prompts shown in Fig. 26.7 (right).

As can be observed, while the images capture something of the original, they are stylistically different¹⁰ and conceptually unrelated. The complex and somewhat cryptic nature of the generated prompt text makes it difficult to anticipate how changing the prompt will affect the resultant image, so for now, working with prompt-based systems remains a challenge. More importantly, the derivative nature of generative AI challenges human artists to find creatively interesting, non-derivative ways of working with these systems.

¹⁰ If there were enough images of the original kind in the training set, it might be possible to say ‘in the style of Jon McCormack’. This is left as an exercise for the reader.

26.4.1 Evolution and Generative AI

Evolutionary techniques have been combined with generative AI systems, for example, in the evolution of prompts in text-to-image systems, evolving, for example, according to aesthetic criteria. However the fundamental problem of a machine understanding human subjective criteria remains an open problem [23].

In a summary of the view of aesthetic appreciation from the perspective of neuroimaging, researchers highlighted that the neuroscientific evidence currently suggests that ‘aesthetic appreciation is not a distinct neurobiological process assessing certain objects, but a general system, centred on the mesolimbic reward circuit, for assessing the hedonic value of any sensory object’ [33]. Another important finding was that hedonic values are not solely determined by object properties. They are subject to numerous factors extrinsic to the object itself. Similar claims have come from psychological models [18]. These findings suggest that any algorithmic measure of aesthetics that only considers an object’s visual appearance ignores many other extrinsic factors that humans use to form an aesthetic judgement (including context, prior knowledge and experience, emotional state and affect). Hence, they are unlikely to correlate strongly with specific human judgements.

Nonetheless, what generative image systems (and the internet in general) tell us that there are certain properties of images, for example, that have wide appeal, so it is not that everyone’s taste is so different as to be entirely unique. Some of the reasons for this are well known, including evolutionary adaptations [11, 13, 26]. But in specific creative contexts, such as Western art, where originality and uniqueness are highly valued, generalisations rarely work (if they did it would be easy to formulate and therefore teach ‘creativity’).

Deep learning methods can be used to learn an individual’s style or aesthetic preference [25]; however, such systems require training on large datasets which can be tedious and time-consuming for the artist and still do not do as well as the trained artist’s eye in resolving aesthetic decisions.

26.5 Conclusion

What can evolutionary machine learning contribute to art and creativity? This very much comes down to the individual approach. Technological art is, by definition, a technology-driven approach, but a focus exclusively on technological achievement or efficiency has become increasingly untenable over the last decade when considering the results and their impact on art, culture and environment (at least in the opinion of this author). Technology is increasingly inseparable from broader concerns, including how it affects culture, cultural preservation and diversity; social and environmental impacts. AI has questionable practices regarding the acquisition of training data, the exploitation of cheap labour in developing countries and the dominance of large technology companies focused on profit-driving development.

This does not mean EML cannot be valuable for the arts. The 2023 European STARTS prize for integrating art, science and technology was awarded to Alexandra Daisy Ginsberg from the UK for the work ‘Pollinator Pathmaker’. This work used machine learning methods to design gardens from the perspective of pollinators, allowing people to design and plant gardens that are ‘pollinator friendly’, using art to create agency in this era of ecological crisis. This is just one example of many artworks that are making use of machine learning methods to effect positive change—for both us and our environment—a topic that is becoming increasingly vital globally.

References

1. GenJam: A Genetic Algorithm for Generating Jazz Solos, San Francisco (1994). International Computer Music Association
2. Ashby, W.R.: Can a mechanical chess-player outplay its designer? *Br. J. Philos.* **3**(9), 44–57 (1952)
3. Baluja, S., Pomerleau, D., Jochem, T.: Simulating user’s preferences: towards automated artificial evolution for computer generated images. *Connect. Sci.* **6**, 325–354 (1994)
4. Boden, M.A.: *The Creative Mind: Myths & Mechanisms*. Basic Books, New York, N.Y. (1991)
5. Bommasani, R., Hudson, D.A., Adeli, E., Altman, R., Arora, S., et al.: On the Opportunities and Risks of Foundation Models (2021). arxiv.org/abs/2108.07258
6. Bowler, P.J.: Lamarckism. In: Keller, E.F., Lloyd, E.A. (eds.) *Keywords in Evolutionary Biology*, pp. 188–193. Harvard University Press, Cambridge, MA (1992)
7. Brown, P., Gere, C., Lambert, N., Mason, C.: *White Heat Cold Logic: British Computer Art 1960–1980*. MIT Press, Boston, MA (2009)
8. Christies.: Is artificial intelligence set to become art’s next medium? (2018)
9. Conrad, M., Pattee, H.H.: Evolution experiments with an artificial ecosystem. *J. Theor. Biol.* **28**(3), 393–409 (1970)
10. Dawkins, R.: *The Blind Watchmaker*. Longman Scientific & Technical, Essex, UK (1986)
11. Dissanayake, E.: *Homo Aestheticus: Where Art Comes From and Why*. University of Washington Press, Seattle (1995)
12. Dorin, A., McCormack, J. (eds.): *Living Melodies: Coevolution of Sonic Communication*, Melbourne, Australia (1999). Centre for Electronic Media Art
13. Dutton, D.: *The Art Instinct: Beauty, Pleasure, and Human Evolution*. Oxford University Press (2009)
14. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: *Advances in Neural Information Processing Systems*, pp. 2672–2680 (2014)
15. Holland, J.H.: *Hidden Order: How Adaptation Builds Complexity*. Helix books. Addison-Wesley, Reading, Mass (1995)
16. Horner, A., Goldberg, D.E.: Genetic algorithms and computer-assisted music composition. Technical report, University of Illinois, Chicago (1991)
17. Lambert, N., Latham, W., Fol Leymarie F.: The emergence and growth of evolutionary art: 1980–1993. In: *ACM SIGGRAPH 2013 Art Gallery, SIGGRAPH ’13*, pp. 367–375, New York, NY, USA (2013). Association for Computing Machinery
18. Leder, H., Nadal, M.: Ten years of a model of aesthetic appreciation and aesthetic judgments: the aesthetic episode - developments and challenges in empirical aesthetics. *Br. J. Psychol.* **105**, 443–464 (2014)
19. Machado, P., Romero, J., Manaris, B.: Experiments in computational aesthetics. In: Romero, J., Machado, P. (eds.) *The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music*, pp. 381–415. Springer, Berlin Heidelberg, Berlin, Heidelberg (2008)

20. McCormack, J.: Turbulence: an interactive installation exploring artificial life. In: ACM SIGGRAPH, pp. 182–183, New York (1994)
21. McCormack, J.: Eden: an evolutionary sonic ecosystem. In: Kelemen, J., Sosík, P. (eds.) Advances in Artificial Life, Proceedings of the Sixth European Conference, ECAL, pp. 133–142. Springer (2001)
22. McCormack, J.: On the evolution of sonic ecosystems. In: Adamatzky, A., Komosinski, M. (eds.) Artificial Life Models in Software, pp. 211–230. Springer, London (2005)
23. McCormack, J.: Open problems in evolutionary music and art. In: Rothlauf, F., Branke, J., Cagnoni, S., Corne, D.W., Drechsler, R., Jin, Y., Machado, P., Marchiori, E., Romero, J., Smith, G.D., Squillero, G. (eds.) EvoWorkshops. Lecture Notes in Computer Science, vol. 3449, pp. 428–436. Springer (2005)
24. McCormack, J., Cruz Gambardella, C., Rajcic, N., Krol, S.J., Llano, M.T., Yang, M.: Is writing prompts really making art? In: Johnson, C., Rodríguez-Fernández, N., Rebelo, S.M. (eds.) Artificial Intelligence in Music, Sound, Art and Design, pp. 196–211. Springer Nature Switzerland, Cham (2023)
25. McCormack, J., Lomas, A.: Deep learning of individual aesthetics. *Neural Comput. Appl.* **33**(1), 3–17 (2020)
26. Miller, G.F.: The Mating Mind: How Sexual Choice Shaped the Evolution of Human Nature. William Heinemann, London (2000)
27. Obvious.: Obvious, explained., February (2018)
28. Parikka, J.: Book Review: The Art of Artificial Evolution: A Handbook on Evolutionary Art and Music. *Leonardo* (2008)
29. Pattee, H.H.: The physics of symbols: bridging the epistemic cut. *BioSystems* **60**, 5–21 (2001)
30. Sims, K.: Artificial evolution for computer graphics. In: Computer Graphics, vol. 25, pp. 319–328, New York (1991). ACM SIGGRAPH, ACM SIGGRAPH
31. Sims, K.: Interactive evolution of equations for procedural models. *Vis. Comput.* **9**, 466–476 (1993)
32. Sims, K.: Evolving virtual creatures. In: Computer Graphics (Siggraph '94 Proceedings), July 1994, pp. 15–22 (1994)
33. Skov, M.: Aesthetic appreciation: the view from neuroimaging. *Empir. Stud. Arts* **37**(2), 220–248 (2019)
34. Takagi, H.: Interactive evolutionary computation: fusion of the capabilities of EC optimization and human evaluation. *Proc. IEEE* **89**, 1275–1296 (2001)
35. Todd, P.M., Werner, G.M.: Frankensteinian methods for evolutionary music composition. In: Griffith, N., Todd, P.M. (eds.) Musical Networks: Parallel Distributed Perception and Performance, pp. 313–339. MIT Press/Bradford Books, Cambridge, MA (1999)
36. Todd, S., Latham, W.: Mutator: a subjective human interface for evolution of computer sculptures. Technical report, IBM United Kingdom Scientific Centre, Winchester, UK (1991)
37. Todd, S., Latham, W.: Evolutionary Art and Computers. Academic Press, London (1992)
38. Ulam, S.: On some mathematical problems connected with patterns of growth of figures. *Proc. Symp. Appl. Math.* **14**, 215–224 (1962)
39. Watson, A.J., Lovelock, J.E.: Biological homeostasis of the global environment: the parable of daisyworld. *Tellus* **35B**, 284–289 (1983)
40. Wiggins, G., Papadopoulos, G., Phon-Annuaisuk, S., Tuson, A.: Evolutionary methods for musical composition. Technical report, Department of Artificial Intelligence, University of Edinburgh, Scotland, UK (1999)
41. Wilson, S.W.: State of XCS classifier system research. Technical report, Prediction Dynamics, Concord, MA (1999)
42. Witten, T.A., Sander, L.M.: Diffusion-limited aggregation, a kinetic critical phenomenon. *Phys. Rev. Lett.* **47**, 1400–1403 (1981)
43. Zylinska, J.: AI Art: Machine Visions and Warped Dreams. Open Humanities Press (2020)

Index

A

ABS ranking, 696
Abstract imagery, 298
ACCEL, 723
Accessibility, 21, 311, 740
Activation function, 31, 33, 39, 44, 46–48, 50, 85, 211, 221, 265, 368, 377–379, 594, 616, 625
AdaBoost, 36, 69
Adaptive feedback control, 682
Adversarial benchmarks, 466, 480
Adversarial examples, 14, 46, 457, 458, 462–466, 468–470, 472, 475, 480, 481, 488, 499, 500
Adversarial robustness, 457–459, 462, 466–470, 472, 475, 480–482
Adversarial training, 304, 466, 467, 469
Adversarial vulnerability, 14, 458, 468, 481
AmoebaNet, 43, 44
Angle-aware geometric semantic GP, 136
Angry Birds, 726
Animation, 291, 292, 298, 743
Approximate computing, 373
Approximate models, 492, 496, 497
Archipelago model, 614
Architectural innovations, 32
Archives, 72, 92, 143, 193, 194, 210, 223, 296, 307, 411, 679, 723, 727, 730
Area Under the Curve (AUC), 191, 192, 495
Artificial biochemical networks, 603
Astronomical object discovery, 545
Attack design, 463
Autoencoder, 48, 250, 296, 305, 307, 308, 310, 442, 496, 540, 569, 578, 600, 720, 724, 726–728

AutoML, 439, 440, 444, 445, 449, 451–453, 566
AutoML-Zero, 49, 50
Autonomous rendezvous and docking, 618
Auto-WEKA, 444, 445, 452

B

Bagging, 207, 209–211
Baikal loss, 45, 46
Ballistic trajectories, 613, 614
Behavior trees, 659, 669
BenchENAS, 273, 274, 461, 480, 482
Benchmarking ENAS, 461, 480, 481
Bias, 4, 21, 35, 49, 68, 125, 159–161, 177, 190, 207, 209, 308, 347, 409, 441, 470, 507–509, 520, 673, 676
Blind Watchmaker, 291, 740, 742
Block-based search space, 260–263
Blood-pressure prediction, 39
Body, 80, 103, 231, 263, 284, 293, 299, 300, 304, 306, 312, 427, 467, 480, 502, 538, 541, 549, 569, 595, 614, 618, 621, 622, 642, 657–659, 666, 670–680, 682, 684, 718, 724, 740, 743, 749
Boosting, 207, 209, 211
Bootstrap, 209, 220, 331–333, 335, 340, 342–344, 348, 361, 362

C

C4.5, 65, 69, 190
Cancer, 184, 190, 228, 443, 591–596

- Cartesian Genetic Programming (CGP), 43, 85, 103, 377, 379, 388, 601, 642, 646
- Cassini mission, 616
- Catastrophic forgetting, 103, 230
- CelebA, 419–422, 424–427
- Cell-based search space, 260, 262, 263
- Centroid, 10, 12, 157, 158, 580, 720
- Challenges in XAI, 487
- Chaotic dynamics, 350, 603
- Chernoff bound, 277
- CIFAR-10, 37, 38, 43, 47, 51, 248, 250, 264, 270, 273, 380, 382, 385, 388, 458, 459, 462, 464, 467, 468, 470–472, 474–476
- Climate change, 105, 548, 563
- Clinical assessments, 602, 604
- Cluster analysis, 151–157, 159–163
- Cluster shape, 158
- Cluster validity index, 163
- Clustering ensemble, 163
- CNN, 37, 47, 248–250, 255, 256, 259, 260, 266, 276, 367–369, 371, 373–384, 386–392, 423, 458, 462, 466, 468, 481, 569, 573, 594, 598
- CoDeepNEAT, 41, 42, 306
- Codex, 337
- Coevolution, 209, 225, 297, 306
- Community detection, 151, 152, 154, 155, 157–165
- Competitive coevolution, 91, 93, 209, 220, 226, 230, 297, 397, 409, 429
- Complete Binary Genetic Programming (CBGP), 640
- Complexification, 31–33, 35, 89
- Composer/Critic, 744
- Comprehensibility, 72
- Computational cost, 42, 43, 72, 79, 81, 85–87, 138, 165, 180, 262, 297, 310, 409, 428, 569, 571
- Computational creativity, 108
- Consensus solution, 163
- Constrained Surprise Search, 727
- Continual evolution, 362, 644
- Continuous time recurrent neural network (CTRNN), 621
- Controller, 80, 85, 98, 102, 227, 248, 296, 297, 299, 306, 372, 595, 600, 617–622, 624, 625, 629–640, 642–648, 657–660, 662–667, 669–671, 674, 675, 677, 679, 680, 682, 683, 727
- Control objective, 629, 630, 634
- Control optimization, 635, 637
- Control-oriented features, 646
- Control stability, 634, 645
- Convolutional layers, 255–257, 261, 369, 370, 375, 378–381, 387, 388, 471
- Cooperative coevolution, 95–98, 206, 210, 211, 215, 217–219, 221, 223–225, 231, 300, 379
- Coronary artery disease, 451
- Counterfactual fairness, 526
- Counterfactuals, 493, 498–501, 507, 526–528
- Covariance Matrix Adaptation Evolution Strategy (CMA-ES), 45, 46, 89, 96, 104, 296, 303, 341, 539, 620, 635, 637, 645, 725, 726
- COVID-19, 412, 421–424, 543, 591, 592, 594, 596–601
- CPPN-fixed seed, 347
- CPPN-mutable seed, 347, 351
- Creativity, 52, 80, 92, 108, 623, 740, 742–744, 756, 758
- Credit assignment, 79, 81, 84, 90, 93–98, 100, 106, 208, 213, 215, 218, 222, 231
- Credit rating, 695
- Cross-entropy, 45, 95, 463–465, 525
- Cross-validation, 71, 442, 448, 449, 451, 459, 497, 518, 520
- Cryptocurrencies, 701–703
- Crystal structure prediction, 535, 536, 538, 539
- Curriculum learning, 722
- D**
- Dark energy analysis, 546
- Dark matter analysis, 546
- Data augmentation, 12, 40, 49, 51, 52, 302, 412, 460, 471, 598
- Data cleaning, 440, 441, 564, 583
- Data flow, 292, 371, 372, 375, 387, 551
- Data imputation, 12, 60, 61, 64, 66, 138, 188, 412, 566
- Data quality control, 357
- Data understanding, 496
- Denoising Autoencoder, 727, 728
- Density functional theory, 538, 539
- Derivatives, 47, 123, 143, 494, 538, 539, 601, 640, 702, 757
- Design Cost, 391, 392
- Developmental processes, 10
- Diabetes treatment recommendation, 39
- Diagnosis, 172, 184, 188, 190, 198, 591–594, 596, 598, 604

- Diff model, 335–338, 340, 341, 345, 346, 348–352, 729, 730
Diffusion architectures, 44
Diffusion limited aggregation, 748
Digital video digitisation, 747
Digital twins, 644, 645, 647
Dimensionality reduction, 12, 59, 60, 62, 66, 70, 73, 442, 491, 495, 496
Direct policy search, 549, 617, 618, 621
Discovering physical laws, 538
Discrete maps, 603
Discriminant function, 172, 173
Discrimination score, 509, 510, 521, 524
Discriminator, 297, 306, 398–400, 405, 407, 408, 411–414, 418, 424
Dissimilarity matrix, 153, 154, 158
Divergence, 339, 358, 407
Diversity, 16, 51, 71, 79, 80, 86, 88–93, 96–98, 132, 134, 137, 161, 206–212, 216, 218–221, 226, 227, 231, 297, 306, 338–340, 343, 346, 348–353, 360, 397–399, 411–414, 419–426, 428, 429, 480, 493, 495, 498, 501, 502, 516, 525, 540, 542, 548, 638, 673, 679, 723, 725, 727–729, 758
Divide-and-conquer strategy, 576
Domain randomization, 682
Drift method, 277
Drug design, 542, 600
Dynamic fitness, 287
- E**
Early stopping, 269
Echo, 747
Efficient evaluation, 272, 274, 275, 278, 444
Embodied cognition, 658, 670
Energy efficiency, 375, 376, 383, 620
Epidemiological modeling, 591, 599, 600, 604
Episode-based learning, 93, 94
Equilibrium shaping, 623
Ethics, 564
Evolution through Large Models, 331, 332, 729
Evolutionary art, 286, 292, 302, 307
Evolutionary Curriculum Design, 722
Evolutionary data reduction, 172, 178
Evolutionary deep structures, 197, 198
Evolutionary design, 20, 292, 368, 376, 379, 391, 541
Evolutionary Feature Synthesis, 564
Evolutionary generative model, 283–287, 290, 291, 294–297, 300, 301, 303, 304, 307, 309–313
Evolutionary KNN, 176
Evolutionary multi-tasking, 195
Evolutionary Polynomial Regression, 564
Evolutionary surrogate models, 196–198
Evolutionary transfer learning, 195, 198
Evolutionary tree, 742
Explainability, 21, 31, 35, 38, 52, 171, 193, 194, 311, 440, 443, 453, 487–490, 500–502, 536
Explaining data, 491, 495
Explaining model internals, 492
Explaining predictions, 492
Exploration, 17, 23, 83, 84, 90–93, 97–99, 101, 107, 136, 155, 162, 277, 278, 296, 302, 307, 308, 337, 362, 487, 502, 544, 599, 612, 619–621, 635, 638, 647, 723, 730
Exploratory Landscape Analysis (ELA), 647
Expression-based image generation, 292
External archives, 72
External index, 102
- F**
F1-score, 524
Failure modes, 361, 499, 502
Fairness, 21, 49, 162, 487, 507–509, 511, 514–519, 521, 524, 525, 527–529
Fairness measures, 507–511, 517–521, 523–526, 528
Fairness optimisation, 508, 511, 515, 524, 525, 527
Fault recovery, 624
Feature construction, 12, 62, 66, 68–70, 171, 178–181, 185, 194, 195, 197, 210, 227, 231, 496, 507, 515, 527, 564–566, 569–572, 577, 580, 581, 583, 594, 647
Feature discretization, 60, 61, 64, 65
Feature engineering, 128, 227, 439–442, 447, 450–452, 496, 536, 563–566, 580, 583, 598, 616
Feature importance, 143, 176, 441, 443, 489, 491, 493, 494, 496
Feature ranking, 143
Feature selection, 12, 62, 66–71, 73, 139, 140, 143, 171, 173, 178–181, 194, 196, 197, 439–442, 447, 451, 488, 491, 496, 507, 508, 518–521, 524, 527, 528, 536, 550–552, 564–571,

582, 583, 592, 594, 598, 604, 617, 619
 FID, 420–424, 426–428
 Filter methods, 62, 227
 Filter weights, 369
 Financial systems, 701
 Finger taps, 602
 Fitness caching, 268, 271
 Flexibility, 108, 128, 144, 155, 225, 253, 313, 445, 453, 495, 514, 550, 563, 576, 583, 603, 642, 644
 Foreign exchange, 701
 Formation flying, 622–624
 Forward model, 720, 728, 729
 FPGA-compatible components, 33
 Frechet Inception Distance, 414
 Free control parameters, 635
 F1-score, 524
 FT50 list, 696
 Futures, 701

G

Game playing, 39, 89, 217, 718, 721, 731
 Gaussian processes, 601
 Gene linkage, 216, 219, 231
 General control laws, 639
 Generalized advantage estimation, 357
 Generative Adversarial Network (GAN), 46, 397, 398, 400, 412, 717, 724, 744, 754
 Generative encoding, 265, 664
 Generative modelling, 284, 285
 Generator, 293, 297, 302, 303, 305, 306, 340, 398–400, 403–408, 411–414, 418, 420, 424–428, 638, 639, 663, 721, 729
 Genetic heterogeneity, 452
 GenJam, 293, 294, 743
 Geographical information systems, 548, 549
 Geological features, 548, 549
 Geometric semantic GP, 135, 136, 573
 Gini index, 69
 Global explanations, 488, 494, 500
 Gradient-free evolutionary RL, 95
 Graph-based representation, 157, 159, 160, 164, 178, 179, 181, 212, 453
 Graphics Processing Unit (GPU), 21, 89, 90, 227, 248, 249, 259, 260, 273, 277, 310, 357, 368, 371, 373, 386, 387, 423, 461, 462
 Gravity assist maneuvers, 614, 615
 Group-level fairness, 509, 510

Group size, 162
 Guidance, Navigation and Control (GNC), 611, 618

H

Hardware accelerators, 368, 369, 371, 375, 376, 380, 390–392
 Hardware-aware NAS, 368, 375, 383, 384, 386, 391
 Hardware co-design, 368, 387, 389
 Hardware constraints, 31, 33, 40, 388, 423
 Hardware parameters, 368, 374–378, 382, 384, 386, 387, 389, 391
 Hardware simulators, 368, 374
 Hardwired fitness function, 290, 298
 Hearthstone, 727
 Hebbian learning, 666, 682
 Hedonic values, 758
 Hexapods, 621
 Hierarchical RL, 97, 100
 Hierarchy, 87, 102, 107, 210, 223, 375
 Hitchhikers, 215
 Human-competitiveness, 563, 565, 579, 580
 Human intelligence, 740
 Human interpretability, 646, 648
 Hypergraph Clustering Artificial Bee Colony, 568
 Hyper-NEAT, 106, 221, 250, 661, 664, 666, 721, 731
 HyperNEAT approach, 34
 Hyper-volume measure, 516, 525, 526

I

Image alignment, 565, 578, 583
 Image captioning, 41, 42
 Image classification, 4, 38, 42, 172, 184, 185, 248, 250, 257, 259, 270, 273, 309, 367, 368, 379, 386, 412, 458, 463, 468, 499, 594, 730
 Image generation, 292, 298, 299, 302, 303, 309, 310, 398
 ImageNet, 43, 44, 248, 264, 270, 273, 382, 383, 385, 390, 468
 Impact of Data Pre-Processing, 474
 Impact of discretization, 476
 Improving treatment, 595
 Imputation, 60, 61, 64–66, 73, 121, 137–140, 188, 189, 566
 Inception score, 309, 414
 Inception-like modules, 43
 Incomplete data, 64, 137, 139, 140, 188, 189, 441

- Incremental evolution, 104, 106, 107
Incremental growth, 86, 87, 89, 103
Indexed memory, 100, 101
Indirect encodings, 34, 252, 343, 345, 467, 721
Individual-level fairness, 509
Information gain, 35, 69
Infrared video camera, 751
In-memory computing, 373, 392
In-orbit assembly, 622
Input feature maps, 369, 376
Input variables, 121–124, 139, 277
Input-To-State Stability (ISS), 645
Insight, 17, 98, 108, 141, 144, 188, 207, 226, 227, 331, 333, 360, 362, 443, 444, 460, 480, 488, 491, 493, 502, 599, 616, 619
In situ measurements, 564
Insurance, 703
Intelligent mutation, 337
Intensive care, 39
Interactive evolution, 726, 744
Interestingness, 72, 443
Internet of Things, 275
Interpretability, 37, 121, 141, 142, 193–195, 210, 229, 272, 276–278, 311, 440–444, 448, 449, 489, 490, 494–497, 501, 502, 525, 529, 554, 563, 564, 569, 570, 572, 583, 601, 604, 624, 629, 644, 669
Interpretable models, 141, 188, 193, 276, 441, 493, 494, 496, 497, 570, 572, 574, 598, 599
Interval functions, 139
Interval GP, 189
Interventions, 12, 59, 105, 253, 290, 295, 443, 444, 591, 592, 596, 599, 600
Inverse design, 535, 536, 538, 540, 541, 555
Island model, 621
- K**
K-nearest Neighbors, 64
Knee solution, 163
Kernel Inception Distance, 309
- L**
Lamarkian evolution, 751
Large language models, 16, 311, 331–334, 729, 740
Latent space, 16, 290, 296, 303, 307, 308, 398, 412, 492, 542, 725, 726
Latent variable exploration, 296, 303, 308
- Layer-based search space, 260–262
Layered learning, 106, 107, 220
Learning Classifier Systems (LCS), 36, 37, 126, 144, 172–175, 188, 197, 633, 642–644
Learning curve, 275
Level of selection, 206, 208–211, 213, 215, 217, 222, 231
Levodopa-induced dyskinesia, 603
Lexicase selection, 132, 209, 449, 515, 516, 528
Lexicographic optimisation, 511, 513, 518
Lifelong learning, 79, 105, 108
Linear Genetic Programming (LGP), 87, 125, 187, 636, 640
Living melodies, 747
Loans, 502, 701
Local explanations, 494, 497, 500
Local Outlier Factor, 72
Locomotion, 104, 106, 226, 356, 620, 660, 661, 663, 665, 666, 673, 679, 684
Long short-term memory networks, 100, 663
Logistic regression, 441, 448, 497, 515, 517, 518, 524, 593
Loss function, 12, 31, 39, 44–47, 51, 52, 85, 127, 177, 309, 398, 399, 407, 411, 412, 414, 416, 419, 424, 425, 439, 442, 443, 447, 448, 463, 508, 538
Low fidelity, 268–270
Low-thrust transfers, 616–618
LSTM design, 43
Lyapunov control, 616, 618
- M**
Machine learning pipelines, 439–441, 445–447, 449–453, 563, 566
Machine reinforcement learning, 81
Manifold Learning, 62, 63, 66, 70, 307
MAP-Elites, 89, 92, 296, 303, 331, 340, 341, 343, 346, 352, 361, 362, 725, 727
Mapping, 63, 70, 79, 82, 96, 97, 106, 133, 135, 151, 153, 175, 189, 205, 213, 217, 292, 339, 368, 372, 374–376, 379–381, 389, 390, 492, 495, 581, 618, 630, 632, 660, 666, 672, 702, 704, 752
MarioGPT, 730
Markets, 701
Markov decision process, 31, 82
Masking effect, 62, 71
Materials discovery, 502, 535, 542
McCulloch and Pitts neuron model, 661

- Medical imaging, 604
 Medoid, 157, 158
 Memristive crossbars, 392
 Mergers and acquisitions, 701
 Mesolimbic reward circuit, 758
 Metalearning, 31, 39–41, 43, 50–52
 Minimax, 46, 398, 400, 405, 408, 428, 717, 720
 Missing data, 12, 61, 64, 66, 121, 188–190, 197
 Missing value imputation, 64, 566
 MNIST, 273, 379, 380, 420–424, 426, 428
 Model-based interpretability, 193
 Model complexity, 128, 130–132, 140, 450, 489
 Model explainability, 440, 443
 Model extraction, 496, 497
 Model identification, 640, 644, 647
 Model interpretability, 570
 Model understandability, 444
 Model usability, 443, 444
 Modularity, 18, 39, 95, 100–102, 144, 161, 205, 206, 229, 311, 664, 669
 Molecular dynamic simulation, 541–543
 Molecule design, 541
 Monte Carlo Tree Search, 717
 Morphology evolution, 619, 620, 625
 Multi-agent system, 205, 206, 213, 215–217, 219–221, 223, 230, 231
 Multiclass classification, 565, 572, 575–577, 581
 Multi-layer perceptron (MLP), 96, 379, 387, 540, 541, 624
 Multiple Gravity Assist (MGA), 614, 616
 Multiply-And-Accumulate, 369
 Multi-start metaheuristic, 71
 Multitask learning, 42, 226, 230
 Multi-task RL, 87
 Music generation, 293, 299, 302, 306, 307, 309
 Mutator, 380, 742
- N**
 Navigation, 91–93, 99, 101, 104, 221, 226, 231, 232, 311, 611, 618, 663, 670, 684
 NEAT, 32, 33, 41, 86, 96, 99, 100, 290, 296, 304, 306, 664, 721, 731
 Negative transfer, 104, 195
 Neural Architecture Search (NAS), 41, 43, 44, 49, 247–250, 264, 266, 276, 278, 367, 368, 383–392, 467, 611, 659, 661
 Neurocontrollers, 611, 612, 618, 621
- Neuroevolution, 10, 12, 19, 34, 86, 89, 90, 99, 100, 104, 108, 250, 290, 296, 297, 304, 457, 570, 611, 617, 624, 625, 659–661, 664, 684, 717, 719–721, 723, 731
 Neurototalitarianism, 754
 Niche, 86, 89, 92, 340, 341, 346, 348, 349, 351–353, 612, 619
 Noisy features, 567
 Non-dominated solutions, 72, 512–515, 525, 527, 528, 635
 Non-stationary environments, 104
 Norm-Constrained Perturbations, 465, 480
 Novelty Search, 91–93, 108, 221, 304, 306, 308, 412, 501, 723, 727, 730, 731
 NSGA-II, 64, 66, 88, 276, 379, 380, 383, 412, 449, 497–499, 515, 516, 524, 525, 527, 528, 577, 580, 595, 601
- O**
 Object transportation, 684
 OneMax, 277
 Online learning, 17
 Online training, 727
 Opaque gradients, 31
 Open-endedness, 23, 331–333, 335, 338–340, 342, 361, 362, 716
 Optimal control, 595, 611, 617, 618, 625, 630, 631, 637, 638, 640, 643
 Orbital parameter determination, 543
 Outlier detection, 59, 60, 62, 71, 72, 491
 Output feature maps, 369, 370
 Output variables, 121, 122, 143
- P**
 PAIRED, 30, 93, 268, 408, 722
 Pangaea system, 47
 Panspermia, 743
 Parameterized control laws, 634, 636
 Pareto dominance, 512, 518, 524, 525, 527, 528
 Pareto front, 386, 388, 391, 449, 512, 514, 516, 517, 525, 635
 Pareto optimisation, 511, 512
 Parkinson's disease, 591, 592, 601
 Partial differential equations, 536, 538, 539
 Partial effect, 143
 Partial observability, 87, 96, 99, 102
 Pathologies, 29, 209, 213, 215, 217, 218, 230, 231, 398–400, 407, 411, 419, 424, 427
 Pattern detection, 535, 536
 Permutation-based feature importance, 143

- Physics simulation, 743
Pipeline evaluation, 448
Pipeline initialization, 447
Pipeline selection, 448
Pipeline variation, 447
Planaria, 670
Plausibility, 18, 498, 527
POET, 93, 339, 723
Policy, 82–93, 95–104, 106–108, 226, 269, 357, 389, 402, 413, 617, 661, 667, 668, 671, 720, 721, 728
Polymorphism, 37
Portfolio optimization, 695
Post-hoc interpretability, 195, 525
Precision scaling, 368, 373, 379
Predictive accuracy, 68, 69, 72, 441, 443, 446, 449, 457, 459, 481, 507, 508, 511, 514, 515, 518–520, 522, 524, 525, 527, 592
Predictive performance, 386, 440–443, 457–459, 461, 462, 508, 518
Price of robustness, 618
Pricing, 701
Primordial dance, 743
Probabilistic cellular automata, 599
Processing Elements, 372, 389
Program simplification, 142, 194
Propulsion systems, 613, 614
Protein function prediction, 546
Protein structure design, 546, 547
Prototype, 157, 158, 164
Proximal policy optimization, 83, 357
Proxy models, 275, 375, 492
- Q**
Qualitative observations, 360, 361
Quality diversity, 91, 92, 108, 339, 725–729, 731
- R**
Radial seed, 347, 349–352
Real estate, 701, 702
Reality gap, 657, 659, 669, 679–683
Recall, 99–101, 339, 345, 346, 357, 358, 368, 514, 520, 528
Reduced-Order Modeling (ROM), 648
Redundant features, 172, 178
Reflex motions, 620
Regressor selection, 442
Regularization, 12, 30, 31, 33, 40, 44–46, 48, 49, 407
Reinforcement Learning, 11, 17, 36, 37, 40, 43, 105, 107, 174, 205, 206, 217, 222, 226–228, 232, 247, 248, 331, 333, 343, 354, 397, 398, 400–402, 408, 536, 546, 553, 555, 617, 618, 632, 637, 641, 644, 646, 659, 667, 716, 717, 720, 727, 728, 730, 731
Reliability, 46, 162, 368, 391, 392, 573, 612
Remote sensing, 65, 564, 567, 569, 570, 572, 575, 577, 579–581, 583
Representational diversity, 208, 211, 212
Representation learning, 39, 63, 156, 226, 231, 398, 446, 722, 724
Reproducibility, 391, 625
Residual module, 33
Resource-constraint NAS, 272, 275, 278
Reward diversity, 208, 209, 211
Reward signal, 82–84, 90–92, 94, 95, 103, 667, 668
Ring, 306, 413, 421, 422, 540, 621
Risk, 39, 128, 131, 441, 444, 452, 502, 549, 701, 702
Robotic subsystems, 684
Robot optimization, 657
Runtime analysis, 277, 409
Run time bound, 277
- S**
Satellite imagery, 565, 566, 568, 573, 575, 577
Security, 230, 276, 368, 391, 392, 462, 551
Selection fatigue, 744
Self-assembly, 619, 621, 622, 661
Semantic GP, 133, 134, 144, 573
Semiotics, 745
Sensor, 749
Septic shock, 39
Signal propagation, 413, 421, 422, 426, 427
Silhouette width, 163
Similarity, 16, 46, 65, 91, 92, 138, 153, 155, 220–222, 510, 522, 526, 577, 583, 742
Simulator, 96, 345, 368, 369, 392, 596, 621, 624, 679–683
Single dataset, 482
Smearing effect, 62, 71
Sodarace, 331, 333, 342–345, 347, 351, 352, 355, 362
Soft Robots, 663, 665, 672–676, 679, 729
Software security, 550
Solar sails, 618
Solvable problems, 4

Spatial architectures, 368, 372
 Square seed, 347, 348, 351, 353
 Stacking, 102, 212, 263, 451
 Statistics, 4, 163, 285, 309, 380, 460, 545, 546, 641, 728
 Step-based learning, 94
 Stellar spectra modeling, 543, 544
 Stellar structure modeling, 543, 544
 Stock trading, 39, 217, 229
 Streaming data, 225, 229–231
 Style theft, 756
 Super Mario Bros, 293, 297, 717, 720, 724, 727, 728, 730
 Supernet, 249, 275, 278, 383, 384, 386, 387, 389
 Surprise search, 92, 727
 Surrogate-based approaches, 43
 Surrogate modeling, 16, 51, 196–198, 209, 227, 391, 496, 538, 540, 555, 601, 616, 619, 646, 647, 726–728
 Swarm robotics, 661, 666
 Symbiosis, 224, 313
 Symbolic regression, 65, 121, 124, 127–129, 140, 144, 453, 540, 593

T

Tangled program graphs, 85, 97, 102, 226
 Targeted attacks, 463
 Task transfer, 103, 104, 217, 220, 221, 226, 230, 231
 TaylorGLO method, 46
 Team, 51, 94, 97, 205–207, 213–215, 217–227, 231, 615, 720
 Team composition, 206, 213, 215–218, 221, 222, 224, 232
 Technical analysis, 701
 Temporal abstraction, 107
 Temporal architecture, 368, 371
 Tensor Processing Unit (TPU), 89, 373
 Text generation, 293, 295, 298, 300, 302, 306, 412
 Threat models, 458, 462, 464–466, 469, 470, 472–474, 480, 481
 Time series, 7, 34, 39, 61, 62, 66, 71, 229, 231, 565, 569, 573, 576, 577, 582, 583
 Topology-based search space, 260, 264
 TPOT, 20, 66, 439, 445–453, 566
 Trading, 216, 229, 695, 701, 703
 Training from scratch, 106, 268
 Trajectory optimization, 645
 Transfer learning, 17, 79, 105, 106, 140, 171, 195, 270, 412, 646, 723
 Transferability, 263, 463, 579, 679, 680, 683

Transformer architectures, 335
 Transformers, 20, 43, 450, 604
 Transparency, 175, 193, 221, 313, 487, 489, 502

Transportation, 477, 548, 550

Tree-based GP, 173

Trustworthiness, 21, 193

U

UK Biobank, 451

Unbalanced data, 171, 190–192, 197

Unconventional Hardware Platforms, 368, 392

Uni-temporal data, 565

Unit of selection, 206, 214, 231

Unmanned Aerial Vehicles (UAV), 564, 565, 568, 575, 578, 637, 639, 640, 670

Unsupervised Environment Design, 723

Untargeted attacks, 463, 464, 472

Usability, 21, 443, 444

V

Validity, 163, 164, 361, 377, 460, 475, 510, 526, 541, 631

Validation Framework, 71

Vapnik-Chervonenkis dimension, 131

Variable size ensembles, 224

Variational auto-encoders, 305, 308, 310, 600, 726

Venture capital, 701

Virtual creatures, 10, 671, 672, 677, 748

Voxel-based soft robot, 672, 674, 675, 679

Voyager 2, 614

W

Weight compression, 368, 373, 382

Weight inheritance, 268, 272

Weight sharing, 382

Wheels, 340

White-box Attacks, 463, 472, 482

Within-cluster variance, 155, 161

World model, 84, 87, 96, 99, 100, 103, 728

Wrapper methods, 180, 227

X

XCSF, 126, 127

XCS-SR, 144

X-ray, 29, 421, 423, 596

Z

Zebrafish, 603