Handbook of Evolutionary Computation

Handbook of Evolutionary Computation

Editors in Chief

Thomas Bäck, David B Fogel
and Zbigniew Michalewicz

Taylor & Francis
Taylor & Francis Group
New York London

Taylor & Francis is an imprint of the
Taylor & Francis Group, an informa business

Published in 1997 by
Taylor & Francis Group
270 Madison Avenue
New York, NY 10016

Published in Great Britain by
Taylor & Francis Group
2 Park Square
Milton Park, Abingdon
Oxon OX14 4RN

Printed in the United States of America on acid-free paper
10 9 8 7 6 5 4 3 2

International Standard Book Number-10: 0-7503-0895-8 (Hardcover)
International Standard Book Number-13: 978-0-7503-0895-3 (Hardcover)

Library of Congress Cataloging-in-Publication Data

Catalog record is available from the Library of Congress

informa
Taylor & Francis Group is the Academic Division of Informa plc.

Visit the Taylor & Francis Web site at
http://www.taylorandfrancis.com

Contents

Foreword

The *Handbook of Evolutionary Computation* represents a major milestone for the field of evolutionary computation (EC). As is the case with any new field, there are a number of distinct stages of growth and maturation. The field began in the late 1950s and early 1960s as the availability of digital computing permitted scientists and engineers to build and experiment with various models of evolutionary processes. This early work produced a number of important EC paradigms, including evolutionary programming (EP), evolution strategies (ESs), and genetic algorithms (GAs), which served as the basis for much of the work done in the 1970s, a period of intense exploration and refinement of these ideas. The result was a variety of robust algorithms with significant potential for addressing difficult scientific and engineering problems. By the late 1980s and early 1990s the level of activity had grown to the point that each of the subgroups associated with the primary EC paradigms (GAs, ESs, and EP) was involved in planning and holding its own regularly scheduled conferences.

However, within the field there was a growing sense of the need for more interaction and cohesion among the various subgroups. If the field as a whole were to mature, it needed a name, it needed to have an articulated cohesive structure, and it needed a reservoir for archival literature. The 1990s reflect this maturation with the choice of *evolutionary computation* as the name of the field, the establishment of two journals for the field, and the commitment to produce this handbook as the first clear and cohesive description of the field.

With the publication of this handbook there is now a sense of unity and maturity to the field. The handbook represents a momentous accomplishment for which we owe the editors and the many contributors a great deal of thanks. More importantly, it is designed to be an evolving description of the field and will continue to serve as a foundational reference for the future.

Kenneth De Jong, *George Mason University*
Lawrence Fogel, *Natural Selection Inc.*
Hans-Paul Schwefel, *University of Dortmund*

Preface

The *Handbook of Evolutionary Computation* was designed to fulfill the need for a broad-based reference book reflecting the important role that evolutionary computation plays in a variety of disciplines—ranging from the natural sciences and engineering to evolutionary biology and computer sciences. The basic idea of evolutionary computation, which came on the scene in the 1950s, has been to make use of the powerful process of natural evolution as a problem-solving paradigm; this might be done either by simulating it 'by hand' (or by using calculating devices) in a laboratory or by simulating it on a computer. The *history of evolutionary computation* is discussed in one of the handbook's introductory sections; presented there and discussed in more detail in Chapter B1 are the three independently developed mainstream representatives of evolutionary computation techniques (*genetic algorithms*, *evolution strategies*, and *evolutionary programming*) plus the most prominent *derivative methods* (genetic programming and classifier systems). A2.3 B1.2, B1.3 B1.4, B1.5

In the 1960s, visionary researchers developed these mainstream methods of evolutionary computation: J H Holland (1962) at Ann Arbor, Michigan, and H J Bremermann (1962) at Berkeley, California, for genetic algorithms; L J Fogel (1962) at San Diego, California, for evolutionary programming; and I Rechenberg (1965) and H-P Schwefel (1965) at Berlin, Germany, for evolution strategies. Two of these pioneers serve on the handbook's advisory board. The first generation of books in the field of evolutionary computation still provides an impressive literature that demonstrates the capabilities of evolutionary algorithms—especially if one takes into account the limited hardware capacity available at that time (see Fogel *et al* 1966, Rechenberg 1973, Holland 1975 and Schwefel 1977).

Similar in some ways to other early efforts toward imitating nature's powerful problem-solving devices, such as artificial neural networks and fuzzy systems, before receiving recognition evolutionary algorithms also had to experience a period of rejection based on ignorance. The great success for these methods came in the 1980s when extremely complex optimization problems from various disciplines were solved, thus facilitating the undeniable breakthrough of evolutionary computation as a problem-solving methodology. This breakthrough is reflected by the growing number of publications in this field and a corresponding increase in conferences and journals. The most widely recognized conferences are the International Conference on Genetic Algorithms (ICGA) held biannually since 1985, the Parallel Problem Solving from Nature (PPSN) meeting held biannually since 1990, the Annual Conference on Evolutionary Programming held since 1992, and the IEEE International Conference on Evolutionary Computation (ICEC) held annually since 1994. A number of smaller local meetings have also been held. With the journals *Evolutionary Computation* (MIT Press, Cambridge, MA), *BioSystems* (Elsevier, Amsterdam), and the *IEEE Transactions on Evolutionary Computation* (IEEE Press, Piscataway, NJ), the field now has its own periodicals in which research results are published. Despite these several conferences and journals, a large amount of application-specific work is widely scattered in the publications of many diverse disciplines and presented at their related conferences, thus reflecting the general applicability and success of evolutionary computation methods.

The progress made in the theory of evolutionary computation methods since 1990 impressively confirms both the strengths of these algorithms as well as the limitations. Research in this field has reached maturity, concerning theoretical and application aspects, so it becomes important to provide a complete reference for practitioners, theorists, and teachers in a variety of disciplines.

The *Handbook of Evolutionary Computation* was designed to provide such a reference work. It includes complete, clear, and accessible information, thoroughly describing state-of-the-art evolutionary computation research and applications in a comprehensive style. Researchers will want to find precise statements about the algorithms and their theoretical foundations, including proofs of the important theorems. In Part B (Fundamental Concepts of Evolutionary Computation) and Part C (Evolutionary Computation Models), the researcher's primary needs are satisfied. The teacher will find material to cover all topics on the implementation, theory, or application of evolutionary algorithms. The practitioner with a particular problem to solve should start by checking Part G (Evolutionary Computation in Practice:

Case Studies) for an application case study that is similar to his or her own problem—or should turn to the more general Part F (Applications of Evolutionary Computation) to find the class of problems to which the particular problem belongs. Practitioners should also consult Part E (Evolutionary Computation Implementations) for useful implementation and software hints.

This handbook, in addition to covering all paradigms of evolutionary computation in detail (Part B) and giving an overview of the rationale of evolutionary computation and its biological background (Part A), also offers an in-depth presentation of evolutionary computation models (Part C) according to the types of representations used for classes of typical problems (e.g. binary, real-valued, permutations, finite-state machines, or parse trees). Choosing a classification based on representation, the search operators of mutation and recombination, and others, are straightforwardly grouped according to the semantics of the data they manipulate (Chapter C3). Other topics of strong importance for the design of an evolutionary algorithm—such as the selection operator, the fitness-evaluation and constraint-handling issues, and population structures (including all aspects of the parallelization of evolutionary algorithms)—are also covered in Part C.

The applications of evolutionary computation are covered in Parts F and G. In Part F, the focus is on a general presentation of application groups while, in Part G, the focus is on particular examples of case studies from computer and information sciences, engineering, physics, chemistry, economics, and other disciplines. In Part E, the concentration is on practical implementation issues of evolutionary algorithms. In Part H, short essays from key researchers are presented, indicating promising directions for further research. In Part D, the field of hybrid approaches is discussed, in which evolutionary computation methods are coupled with other optimization methods, fuzzy systems, or neural networks.

Readers are encouraged to use an arbitrary entry point, either by section title or by index, and to proceed in a highly 'nonlinear' way, following cross-references to other handbook locations. Most sections of the handbook may be read individually. Finally, the handbook is designed to be updated regularly with the publication of loose-leaf supplements twice a year. These will not only expand the handbook but also occasionally replace existing pages, reflecting the changes in our knowledge of and our uses for evolutionary computation.

In addition to the printed version, the handbook also exists in an electronic form on CD-ROM and on the World Wide Web (WWW)—the CD-ROM is essentially a WWW browser for the handbook. The complete handbook is accessible as a hypertext document with extensive cross-references and full search possibilities (search on full text, keywords, titles, and authors). The full contents of the handbook are presented in Adobe Acrobat pdf format, providing on-screen rendering of printed pages. The electronic version also provides hyperlinks to other WWW locations, such as author home pages, ftp sites, and so on. By means of extensive WWW usage, the electronic version of the handbook provides fast accessibility of information, extensive search and retrieval possibilities, and the on-line exploitation of cross-references to other sections of the handbook. The WWW database will be steadily updated in advance of changes to and expansions of the printed edition, and a new version of the CD-ROM will be issued every six months. The electronic version of the handbook is therefore a state-of-the-art exploitation of the Internet as a worldwide information-access network.

This project is part of an even larger project. The *Handbook of Evolutionary Computation* has two sister handbooks on neural networks and on fuzzy systems, such that the complete field of computational intelligence is covered by the three. Computational intelligence, or 'soft computing' in Lotfi Zadeh's terminology, focuses on the subsymbolic aspects of information processing to realize systems that exhibit computational adaptivity and fault tolerance, as well as speed and error rates close to human performance (see Bezdek 1994). Evolutionary algorithms, artificial neural networks, and fuzzy systems are currently the dominant paradigms in the field, and they all provide the potential for the development of intelligent systems that come closer to machine intelligence than any other technique or groups of techniques have to date.

We are pleased that the *Handbook of Evolutionary Computation* is a significant piece of the bigger puzzle, and we hope the handbook user (we believe that 'user' is more appropriate than 'reader') will benefit from the information collected, compiled, and presented.

The idea for the *Handbook of Evolutionary Computation* was first conceived in the summer of 1993 at the Fifth International Conference on Genetic Algorithms (ICGA) at the University of Illinois at Urbana-Champaign. It was not conceived of by us but by Sean Pidgeon (now at Oxford University Press, then at Institute of Physics Publishing). His idea for this project became our greatest challenge. We took the time to identify the best possible scientific editorial and advisory board members for the handbook, scholars

who shared our enthusiasm about the project, who were willing to invest their time in it. Indeed, it took us almost four years to complete the project, to see the first edition in print. We have been involved in the many phases of the publishing process, especially the decisions about the handbook's contents and the appropriate authors; several of our necessary editorial meetings were conveniently held at the evolutionary computation conferences.

Organizational support provided by Oxford University Press and Institute of Physics Publishing made it possible to bring our project to completion. In particular, we would like to express our gratitude to our project editor, Matt Giarratano, and his assistant Merilee Johnson. Besides working on editorial and organizational issues, they also had the burden of motivating everyone to complete the handbook on time. Sean Pidgeon was available to answer our many questions, and through the later years we developed a personal relationship that for us all went far beyond business matters. Our gratitude goes as well to all those involved with the handbook from inception to publication.

Thomas Bäck, *Leiden University*
David B Fogel, *Natural Selection Inc.*
Zbigniew Michalewicz, *University of North Carolina*

December 1996

References

Bezdek J C 1994 What is computational intelligence? *Computational Intelligence: Imitating Life* ed J M Zurada, R J Marks II and C J Robinson (Piscataway, NJ: IEEE) pp 1–12

Bremermann H J 1962 Optimization through evolution and recombination *Self-organizing Systems* ed M C Yovits, G T Jacobi and G D Goldstine (Washington, DC: Spartan Books) pp 93–106

Fogel L J 1962 Autonomous automata *Industr. Res.* **4** 14-19

Fogel L J, Owens A J and Walsh M J 1966 *Artificial Intelligence through Simulated Evolution* (New York: Wiley)

Holland J H 1962 Outline for a logical theory of adaptive systems *J. ACM* **3** 297–314

——1975 *Adaptation in Natural and Artificial Systems* (Ann Arbor, MI: University of Michigan Press)

Rechenberg I 1965 Cybernetic solution path of an experimental problem *Royal Aircraft Establishment Library Translation No 1122* (Farnborough, UK)

——1973 *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution* (Stuttgart: Frommann-Holzboog)

Schwefel H-P 1965 *Kybernetische Evolution als Strategie der Experimentellen Forschung in der Strömungstechnik* Diplomarbeit (Berlin: Technische Universität, Hermann Föttinger Institut für Strömungstechnik)

——1977 *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie* Interdisciplinary Systems Research, vol 26 (Basel: Birkhäuser)

How to Use This Handbook

The *Handbook of Evolutionary Computation* is the second in a series of three updatable reference works known collectively as the Computational Intelligence Library. (The other two volumes are the *Handbook of Neural Computation* and the *Handbook of Fuzzy Computation.*) This handbook has been designed to provide valuable information to a diverse readership. Through regular supplements, the handbook will remain fully up to date and will develop and evolve along with the research field that it represents.

WHERE TO LOOK FOR INFORMATION

An informal categorization of readers and their possible information requirements is given below, together with pointers to appropriate sections of the handbook.

The Research Scientist

This reader has a very good general knowledge of evolutionary computation (EC). She may want to

- develop new EC models or improve existing ones (Part C: Evolutionary Computation Models)
- develop new applications of EC (Part F: Applications of Evolutionary Computation; Part G: Evolutionary Computation in Practice: Case Studies)
- improve the underlying theory and/or heuristic principles of EC (Part B: Fundamental Concepts of Evolutionary Computation; Part H: Evolutionary Computation Research)

The Applications Specialist

This reader is working in a technical environment (such as engineering). He perhaps

- has a problem that may be amenable to an EC solution (Part F: Applications of Evolutionary Computation; Part C: Evolutionary Computation Models)
- wants to compare the cost-effectiveness of the EC solution with that of other possible solutions (Part F: Applications of Evolutionary Computation)
- is interested in real systems experience as conveyed by case studies (Part G: Evolutionary Computation in Practice: Case Studies)

The Practitioner

This reader is working in a professional discipline that is not closely related to computer science, such as medicine or finance. She may have heard of the potential of EC for solving problems in her professional field, but might have little or no knowledge of the principles of evolutionary algorithms or of how to apply them in practice. She may want to

- find a quick way into the subject (Part A: Introduction; Part B: Fundamental Concepts of Evolutionary Computation)
- look at real case studies to see what EC has already achieved in her field of interest (Part G: Evolutionary Computation in Practice: Case Studies; Part F: Applications of Evolutionary Computation)
- find a relatively easy and quick route to implementation of an EC solution (Part G: Evolutionary Computation in Practice: Case Studies; Part F: Applications of Evolutionary Computation; Part C: Evolutionary Computation Models)

The Student (or Teacher)

This reader may be

- looking for an easy way into the subject (Part A: Introduction)
- interested in getting a firm grasp of the fundamentals (Part B: Fundamental Concepts of Evolutionary Computation)
- interested in practical examples for projects (Part G: Evolutionary Computation in Practice: Case Studies)

CROSS-REFERENCES

Most of the articles in the handbook contain cross-references to related articles. A section number in the margin indicates that further information on the concept under discussion may be found in that section of the handbook. The notation in the following example indicates that further information on genetic algorithms and evolution strategies may be found in Sections B1.2 and B1.3, respectively.

B1.2 B1.3 Several evolutionary approaches have been proposed for applications of this type. The *genetic algorithm* and the *evolution strategy* were considered in this case.

In the electronic edition of the handbook, these marginal section numbers become hypertext links to the section in question. (Full details of the functionality of the electronic edition are provided in the application itself.)

NUMBERING OF EQUATIONS, FIGURES, PAGES, AND TABLES

To facilitate incorporation of the regular supplements to the handbook, which will include new material and updates to existing articles, a unique system of numbering of equations, figures, pages and tables has been employed. Each section in the handbook starts at page 1 with the section code preceding the page number. For example, Section F1.8 starts on page F1.8:**1** and continues through F1.8:**9**, and then Section F1.9 follows on page F1.9:**1**. Equations, figures, and tables are numbered sequentially throughout each section with the section code preceding the number of the equation, figure, and table. For example, the third equation in Section B2.4 is referred to as equation (B2.4.3) or simply (B2.4.3). The third figure or table in the same section would be referred to as figure B2.4.3 or table B2.4.3.

FURTHER INFORMATION

For the latest information on the *Handbook of Evolutionary Computation*, please visit our websites at http://www.oup-usa.org/acadref/hoec.html, or http://www.iop.org/Books/Catalogue/0750303921, or you may contact the editors in chief or the publisher at the contact addresses given below.

Dr Thomas Bäck
Center for Applied Systems Analysis
Informatik Centrum Dortmund
Joseph-von-Fraunhofer-Strasse 20
Dortmund D-44227, Germany
baeck@ls11.informatik.uni-dortmund.de

Professor Zbigniew Michalewicz
Department of Computer Science
University of North Carolina
9201 University City Boulevard
Charlotte, NC 28223-0001, USA
zbyszek@uncc.edu

Dr David B Fogel
Natural Selection Inc.
Suite 200
3333 North Torrey Pines Court
La Jolla, CA 92037, USA
dfogel@natural-selection.com

Mr Sean Pidgeon
Senior Editor, Scholarly and Professional Reference
Oxford University Press
198 Madison Avenue
New York, NY 10016, USA
sdp@oup-usa.org

Supplements to This Handbook

The *Handbook of Evolutionary Computation* will be updated on a regular basis by means of supplements containing new contributions and revisions to existing articles. To receive these supplements it is essential that you complete the registration card at the front of the loose-leaf binder and return it to the address indicated on the card. (Purchasers of the electronic edition will receive separate registration information.) After you have registered, you will receive new supplements as they are published. The first two supplements are free; thereafter, you will be sent subscription renewal notices. If you wish to keep your copy of the handbook fully up to date, it is essential that you renew your subscription promptly.

If you have not already completed and returned the registration card, please do so now.

IMPORTANT

Please remember that no part of this handbook may be reproduced without the prior permission of Institute of Physics Publishing and Oxford University Press

PART A

INTRODUCTION

A1

Why Evolutionary Computation?

Contents

A1.1 Introduction

David B Fogel

Abstract

A rationale for simulating evolution is offered in this section. Efforts in evolutionary computation commonly derive from one of four different motivations: improving optimization, robust adaptation, machine intelligence, and facilitating a greater understanding of biology. A brief overview for each of these avenues is offered here.

A1.1.1 Introductory remarks

As a recognized field, *evolutionary computation* is quite young. The term itself was invented as recently as 1991, and it represents an effort to bring together researchers who have been following different approaches to simulating various aspects of evolution. These techniques of *genetic algorithms*, *evolution strategies*, B1.2, B1.3 and *evolutionary programming* have one fundamental commonality: they each involve the reproduction, B1.4 random variation, competition, and selection of contending individuals in a population. These form the essential essence of evolution, and once these four processes are in place, whether in nature or in a computer, evolution is the inevitable outcome (Atmar 1994). The impetus to simulate evolution on a computer comes from at least four directions.

A1.1.2 Optimization

Evolution is an optimization process (Mayr 1988, p 104). Darwin (1859, ch 6) was struck with the 'organs of extreme perfection' that have been evolved, one such example being the image-forming eye (Atmar, 1976). Optimization does not imply perfection, yet evolution can discover highly precise functional solutions to particular problems posed by an organism's environment, and even though the mechanisms that are evolved are often overly elaborate from an engineering perspective, function is the sole quality that is exposed to natural selection, and functionality is what is optimized by iterative selection and mutation.

It is quite natural, therefore, to seek to describe evolution in terms of an algorithm that can be used to solve difficult engineering optimization problems. The classic techniques of gradient descent, deterministic hill climbing, and purely random search (with no heredity) have been generally unsatisfactory when applied to nonlinear optimization problems, especially those with stochastic, temporal, or chaotic components. But these are the problems that nature has seemingly solved so very well. Evolution provides inspiration for computing the solutions to problems that have previously appeared intractable. This was a key foundation for the efforts in evolution strategies (Rechenberg 1965, 1994, Schwefel 1965, 1995).

A1.1.3 Robust adaptation

The real world is never static, and the problems of temporal optimization are some of the most challenging. They require changing behavioral strategies in light of the most recent feedback concerning the success or failure of the current strategy. Holland (1975), under the framework of genetic algorithms (formerly called *reproductive plans*), described a procedure that can evolve strategies, either in the form of coded strings or as explicit behavioral rule bases called *classifier systems*, by exploiting the potential to recombine B1.5.2 successful pieces of competing strategies, bootstrapping the knowledge gained by independent individuals. The result is a robust procedure that has the potential to adjust performance based on feedback from the environment.

A1.1.4 Machine intelligence

Intelligence may be defined as the capability of a system to adapt its behavior to meet desired goals in a range of environments (Fogel 1995, p xiii). Intelligent behavior then requires prediction, for adaptation to future circumstances requires predicting those circumstances and taking appropriate action. Evolution has created creatures of increasing intelligence over time. Rather than seek to generate machine intelligence by replicating humans, either in the rules they may follow or in their neural connections, an alternative approach to generating machine intelligence is to simulate evolution on a class of predictive algorithms. This was the foundation for the evolutionary programming research of Fogel (1962, Fogel *et al* 1966).

A1.1.5 Biology

Rather than attempt to use evolution as a tool to solve a particular engineering problem, there is a desire to capture the essence of evolution in a computer simulation and use the simulation to gain new insight
A2.1 into the physics of *natural evolutionary processes* (Ray 1991). Success raises the possibility of studying alternative biological systems that are merely plausible images of what life might be like in some way. It also raises the question of what properties such imagined systems might have in common with life as evolved on Earth (Langton 1987). Although every model is incomplete, and assessing what life might be like in other instantiations lies in the realm of pure speculation, computer simulations under the rubric of *artificial life* have generated some patterns that appear to correspond with naturally occurring phenomena.

A1.1.6 Discussion

The ultimate answer to the question 'why simulate evolution?' lies in the lack of good alternatives. We cannot easily germinate another planet, wait several millions of years, and assess how life might develop elsewhere. We cannot easily use classic optimization methods to find global minima in functions when they are surrounded by local minima. We find that expert systems and other attempts to mimic human intelligence are often brittle: they are not robust to changes in the domain of application and are incapable of correctly predicting future circumstances so as to take appropriate action. In contrast, by successfully exploiting the use of randomness, or in others words *the useful use of uncertainty*, 'all possible pathways are open' for evolutionary computation (Hofstadter 1995, p 115). Our challenge is, at least in some important respects, to not allow our own biases to constrain the potential for evolutionary computation to discover new solutions to new problems in fascinating and unpredictable ways. However, as always, the ultimate advancement of the field will come from the careful abstraction and interpretation of the natural processes that inspire it.

References

Atmar J W 1976 *Speculation on the Evolution of Intelligence and its Possible Realization in Machine Form* Doctoral Dissertation, New Mexico State University

Atmar W 1994 Notes on the simulation of evolution *IEEE Trans. Neural Networks* **NN-5** 130–47

Darwin C R 1859 *On the Origin of Species by Means of Natural Selection or the Preservation of Favoured Races in the Struggle for Life* (London: Murray)

Fogel D B 1995 *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence* (Piscataway, NJ: IEEE)

Fogel L J 1962 Autonomous automata *Industr. Res.* **4** 14–9

Fogel L J, Owens A J and Walsh M J 1966 *Artificial Intelligence through Simulated Evolution* (New York: Wiley)

Hofstadter D 1995 *Fluid Concepts and Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought* (New York: Basic Books)

Holland J H 1975 *Adaptation in Natural and Artificial Systems* (Ann Arbor, MI: University of Michigan Press)

Langton C G 1987 Artificial life *Artificial Life* ed C G Langton (Reading, MA: Addison-Wesley) pp 1–47

Mayr E 1988 *Toward a New Philosophy of Biology: Observations of an Evolutionist* (Cambridge, MA: Belknap)

Ray T 1991 An approach to the synthesis of life *Artificial Life II* ed C G Langton, C Taylor, J D Farmer and S Rasmussen (Reading, MA: Addison-Wesley) pp 371–408

Rechenberg I 1965 *Cybernetic Solution Path of an Experimental Problem* Royal Aircraft Establishment Library Translation 1122, Farnborough, UK

——1994 *Evolutionsstrategies '94* (Stuttgart: Frommann-Holzboog)

Schwefel H-P 1965 *Kybernetische Evolution als Strategie der Experimentellen Forschung in der Strömungstechnik* Diploma Thesis, Technical University of Berlin

——1995 *Evolution and Optimum Seeking* (New York: Wiley)

A1.2 Possible applications of evolutionary computation

David Beasley

Abstract

This section describes some of the applications to which evolutionary computation has been applied. Applications are divided into the areas of planning, design, simulation and identification, control, and classification.

A1.2.1 Introduction

Applications of evolutionary computation (EC) fall into a wide continuum of areas. For convenience, in this section they have been split into five broad categories:

- planning
- design
- simulation and identification
- control
- classification.

These categories are by no means meant to be absolute or definitive. They all overlap to some extent, and many applications could rightly appear in more than one of the categories.

These categories correspond to the sections found in Part F roughly as follows: *planning* is covered by F1.5 (scheduling) and F1.7 (packing); *simulation* and *identification* are covered by F1.4 (identification), F1.8 (simulation models) and F1.10 (simulated evolution); *control* is covered by F1.3 (control); *classification* is covered by F1.6 (pattern recognition). *Design* is not covered by a specific section in Part F. F1.5, F1.7, F1.4 F1.8, F1.10 F1.3, F1.6

Some of the applications mentioned here are described more fully in other parts of this book as indicated by marginal cross-references. The final part of this section lists a number of bibliographies where more extensive information on EC applications can be found.

A1.2.2 Applications in planning

A1.2.2.1 Routing

Perhaps one of the best known combinatorial optimization problems is the *traveling salesman problem* or TSP (Goldberg and Lingle 1985, Grefenstette 1987, Fogel 1988, Oliver *et al* 1987, Mühlenbein 1989, Whitley *et al* 1989, Fogel 1993a, Homaifar *et al* 1993). A salesman must visit a number of cities, and then return home. In which order should the cities be visited to minimize the distance traveled? Optimizing the tradeoff between speed and accuracy of solution has been one aim (Verhoeven *et al* 1992). G1.3, G9.5

A generalization of the TSP occurs when there is more than one salesman (Fogel 1990). The *vehicle routing problem* is similar. There is a fleet of vehicles, all based at the same depot. A set of customers must each receive one delivery. Which route should each vehicle take for minimum cost? There are constraints, for example, on vehicle capacity and delivery times (Blanton and Wainwright 1993, Thangia *et al* 1993).

Closely related to this is the *transportation problem*, in which a single commodity must be distributed G9.8

to a number of customers from a number of depots. Each customer may receive deliveries from one or more depots. What is the minimum-cost solution? (Michalewicz 1992, 1993).

Planning the path which a *robot* should take is another route planning problem. The path must be feasible and safe (i.e. it must be achievable within the operational constraints of the robot) and there must be no collisions. Examples include determining the joint motions required to move the gripper of a robot arm between locations (Parker *et al* 1989, Davidor 1991, McDonnell *et al* 1992), and autonomous vehicle routing (Jakob *et al* 1992, Page *et al* 1992). In unknown areas or nonstatic environments, on-line
G3.6 *planning/navigating* is required, in which the robot revises its plans as it travels.

A1.2.2.2 Scheduling

Scheduling involves devising a plan to carry out a number of activities over a period of time, where the activities require resources which are limited, there are various constraints and there are one or more objectives to be optimized.

G1.2.3 *Job shop scheduling* is a widely studied NP-complete problem (Davis 1985, Biegel and Davern 1990, Syswerda 1991, Yamada and Nakano 1992). The scenario is a manufacturing plant, with machines of different types. There are a number of jobs to be completed, each comprising a set of tasks. Each task requires a particular type of machine for a particular length of time, and the tasks for each job must be completed in a given order. What schedule allows all tasks to be completed with minimum cost? Husbands
G9.3 (1993) has used the additional biological metaphor of an *ecosystem*. His method optimizes the sequence of tasks in each job at the same time as it builds the schedule. In real job shops the requirements may change while the jobs are being carried out, requiring that the schedule be replanned (Fang *et al* 1993). In the limit, the manufacturing process runs continuously, so all scheduling must be carried out on-line, as in a chemical flowshop (Cartwright and Tuson 1994).

Another scheduling problem is to devise a timetable for a set of examinations (Corne *et al* 1994),
G9.4 university lectures (Ling 1992), a *staff rota* (Easton and Mansour 1993) or suchlike.

In computing, scheduling problems include efficiently allocating tasks to processors in a multiprocessor system (Van Driessche and Piessens 1992, Kidwell 1993, Fogel and Fogel 1996), and devising memory cache replacement policies (Altman *et al* 1993).

A1.2.2.3 Packing

Evolutionary algorithms (EAs) have been applied to many packing problems, the simplest of which is
G9.7 the one-dimensional *zero–one knapsack problem*. Given a knapsack of a certain capacity, and a set of items, each with a particular size and value, find the set of items with maximum value which can be accommodated in the knapsack. Various real-world problems are of this type: for example, the allocation of communication channels to customers who are charged at different rates.

There are various examples of two-dimensional packing problems. When manufacturing items are cut from sheet materials (e.g. metal or cloth), it is desirable to find the most compact arrangement of pieces, so as to minimize the amount of scrap (Smith 1985, Fujita *et al* 1993). A similar problem arises in the design of layouts for integrated circuits—how should the subcircuits be arranged to minimize the total chip area required (Fourman 1985, Cohoon and Paris 1987, Chan *et al* 1991)?

In three dimensions, there are obvious applications in which the best way of packing objects into a restricted space is required. Juliff (1993) has considered the problem of packing goods into a truck for
F1.7 delivery. (See also Section F1.7 of this handbook.)

A1.2.3 Applications in design

G3.1 The design of *filters* has received considerable attention. EAs have been used to design electronic or digital systems which implement a desired frequency response. Both finite impulse response (FIR) and infinite impulse response (IIR) filter structures have been employed (Etter *et al* 1982, Suckley 1991, Fogel 1991, Fonseca *et al* 1993, Ifeachor and Harris 1993, Namibar and Mars 1993, Roberts and Wade 1993, Schaffer and Eshelman 1993, White and Flockton 1993, Wicks and Lawson 1993, Wilson and Macleod 1993). EAs have also been used to optimize the design of signal processing systems (San Martin and Knight 1993) and in integrated circuit design (Louis and Rawlins 1991, Rahmani and Ono 1993). The *unequal-area facility layout problem* (Smith and Tate 1993) is similar to integrated circuit design. It involves finding

a two-dimensional arrangement of 'departments' such that the distance which information has to travel between departments is minimized.

EC techniques have been widely applied to *artificial neural networks*, both in the design of network topologies and in the search for optimum sets of weights (Miller *et al* 1989, Fogel *et al* 1990, Harp and Samad 1991, Baba 1992, Hancock 1992, Feldman 1993, Gruau 1993, Polani and Uthmann 1993, Romaniuk 1993, Spittle and Horrocks 1993, Zhang and Mühlenbein 1993, Porto *et al* 1995). They have also been applied to Kohonen feature map design (Polani and Uthmann 1992). Other types of network design problems have also been approached (see Section G1.3 of this handbook), for example, in telecommunications (Cox *et al* 1991, Davis and Cox 1993). G1.3

There have been many *engineering* applications of EC: structure design, both two-dimensional, such as a plane truss (Lohmann 1992, Watabe and Okino 1993), and three-dimensional, such as aircraft design (Bramlette and Bouchard 1991), actuator placement on space structures (Furuya and Haftka 1993), *linear accelerator* design, gearbox design, and chemical reactor design (Powell and Skolnick 1993). In relation to high-energy physics, the design of *Monte Carlo generators* has been tackled. G3 G4.2 G4.1

In order to perform parallel computations requiring global coordination, EC has been used to design *cellular automata* with appropriate communication mechanisms. G1.6

There have also been applications in *testing and fault diagnosis*. For example, an EA can be used to search for challenging fault scenarios for an *autonomous vehicle controller*. G3.4

A1.2.4 Applications in simulation and identification

Simulation involves taking a design or model for a system, and determining how the system will behave. In some cases this is done because we are unsure about the behavior (e.g. when designing a new aircraft). In other cases, the behavior is known, but we wish to test the accuracy of the model (e.g. see Section F1.8 of this handbook). F1.8

EC has been applied to difficult problems in *chemistry* and *biology*. Roosen and Meyer (1992) used an evolution strategy to determine the equilibrium of chemically reactive systems, by determining the minimum free enthalpy of the compounds involved. The determination of the three-dimensional structure of a protein, given its amino acid sequence, has been tackled (Lucasius *et al* 1991). Lucasius and Kateman (1992) approached this as a sequenced subset selection problem, using two-dimensional nuclear magnetic resonance spectrum data as a starting point. Others have searched for energetically favorable protein conformations (Schulze-Kremer 1992, Unger and Moult 1993), and used EC to assist with drug design (Gehlhaar *et al* 1995). EC has been used to simulate how the nervous system learns in order to test an existing theory (see Section G8.4 of this handbook). Similarly, EC has been used in order to help develop models of biological evolution (see Section F1.10 of this handbook). G5, G6 G8.4 F1.10

In the field of *economics*, EC has been used to model *economic interaction* of competing firms in a market. G7.1

Identification is the inverse of simulation. It involves determining the design of a system given its behavior. F1.4

Many systems can be represented by a model which produces a single-valued output in response to one or more input signals. Given a number of observations of input and output values, *system identification* is the task of deducing the details of the model. Flockton and White (1993) concern themselves with determining the poles and zeros of the system. G1.4

One reason for wanting to identify systems is so that we can predict the output in response to a given set of inputs. In Section G4.3 of this handbook EC is employed to fit equations to noisy, chaotic medical data, in order to predict future values. Janikow and Cai (1992) similarly used EC to estimate statistical functions for survival analysis in clinical trials. In a similar area, Manela *et al* (1993) used EC to fit spline functions to noisy pharmaceutical fermentation process data. G4.3

In Section G5.1 of this handbook, EC is used to identify the sources of airborne pollution, given data from a number of monitoring points in an urban area—the *source apportionment problem*. G5.1

In *electromagnetics*, Tanaka *et al* (1993) have applied EC to determining the two-dimensional current distribution in a conductor, given its external magnetic field.

Away from conventional *system* identification, in Section G8.3 of this handbook, an EC approach was used to help with identifying criminal suspects. This system helps witnesses to create a likeness of the suspect, without the need to give an explicit description. G8.3

A1.2.5 Applications in control

There are two distinct approaches to the use of EC in control: *off-line* and *on-line*. The off-line approach uses an EA to design a controller, which is then used to control the system. The on-line approach uses an EA as an active part of the control process. Therefore, with the off-line approach there is nothing evolutionary about the control process itself, only about the design of the controller.

Some researchers (Fogel *et al* 1966, DeJong 1980) have sought to use the adaptive qualities of EAs in order to build on-line controllers for dynamic systems. The advantage of an evolutionary controller is that it can adapt to cope with systems whose characteristics change over time, whether the change is gradual or sudden. Most researchers, however, have taken the off-line approach to the control of relatively unchanging systems.

Fonseca and Fleming (1993) used an EA to design a controller for a gas turbine engine, to optimize its step response. A control system to optimize combustion in multiple-burner furnaces and boiler plants
G3.2 is discussed in Section G3.2. EC has also been applied to the control of guidance and navigation systems (Krishnakumar and Goldberg 1990, 1992).

Hunt (1992b) has tackled the problem of synthesizing LQG (linear–quadratic–Gaussian) and H_∞ (H-infinity) optimal controllers. He has also considered the frequency domain optimization of controllers with fixed structures (Hunt 1992a).

Two control problems which have been well studied are balancing a pole on a movable cart (Fogel 1995), and backing up a trailer truck to a loading bay from an arbitrary starting point (Abu Zitar and Hassoun 1993). In robotics, EAs have been developed which can evolve control systems for visually
G3.7 guided behaviors (see Section G3.7). They can also learn how to control mobile robots (Kim and Shim 1995), for example, controlling the legs of a six-legged 'insect' to make it crawl or walk (Spencer 1993). Almássy and Verschure (1992) modeled the interaction between natural selection and the adaptation of individuals during their lifetimes to develop an agent with a distributed adaptive control framework which learns to avoid obstacles and locate food sources.

A1.2.6 Applications in classification

B1.5.2 A significant amount of EC research has concerned the theory and practice of *classifier systems* (CFS) (Booker 1985, Holland 1985, 1987, Holland *et al* 1987, Robertson 1987, Wilson 1987, Fogarty 1994). Classifier systems are at the heart of many other types of system. For example, many control systems rely on being able to *classify* the characteristics of their environment before an appropriate control decision can be made. This is true in many *robotics* applications of EC, for example, learning to control robot arm motion (Patel and Dorigo 1994) and learning to solve mazes (Pipe and Carse 1994).

An important aspect of a classifier system, especially in a control application, is how the state space is partitioned. Many applications take for granted a particular partitioning of the state space, while in others, the appropriate partitioning of the state space is itself part of the problem (Melhuish and Fogarty
G2.3 1994). In Section G2.3, EC was used to determine optimal symbolic descriptions for concepts.

Game playing is another application for which classification plays a key role. Although EC is often applied to rather simple games (e.g. the prisoner's dilemma (Axelrod 1987, Fogel 1993b)), this is sometimes motivated by more serious applications, such as military ones (e.g. the two-tanks game (Fairley and Yates
G3.3 1994) and air *combat maneuvering*.

EC has been hybridized with feature partitioning and applied to a range of tasks (Güvenir and Şirin 1993), including classification of iris flowers, prediction of survival for heart attack victims from echocardiogram data, diagnosis of heart disease, and classification of glass samples. In *linguistics*, EC has been applied to the classification of Swedish words.

In *economics*, Oliver (1993) has found rules to reflect the way in which consumers choose one brand rather than another, when there are multiple criteria on which to judge a product. A fuzzy hybrid system
G7.2 has been used for *financial decision making*, with applications to credit evaluation, risk assessment, and insurance underwriting.

In *biology*, EC has been applied to the difficult task of protein secondary-structure determination, for
G6.1 example, classifying the locations of particular *protein segments* (Handley 1993). It has also been applied to the classification of soil samples (Punch *et al* 1993).

In *image processing*, there have been further military applications, classifying features in images as targets (Bala and Wechsler 1993, Tackett 1993), and also non-military applications, such as *optical character recognition*. G8.1

Of increasing importance is the efficient storage and retrieval of *information*. Section G2.2 is concerned with generating equifrequency distributions of material, to improve the efficiency of information storage and its subsequent retrieval. EC has also been employed to assist with the representation and storage of chemical structures, and the retrieval from databases of molecules containing certain substructures (Jones *et al* 1993). The retrieval of *documents* which match certain characteristics is becoming increasingly important as more and more information is held on-line. Tools to retrieve documents which contain specified words have been available for many years, but they have the limitation that constructing an appropriate search query can be difficult. Researchers are now using EAs to help with *query construction* (Yang and Korfhage 1993). G2.2 G2.1

A1.2.7 Summary

EC has been applied in a vast number of application areas. In some cases it has advantages over existing computerized techniques. More interestingly, perhaps, it is being applied to an increasing number of areas in which computers have not been used before. We can expect to see the number of applications grow considerably in the future. Comprehensive bibliographies in many different application areas are listed in the further reading section of this article.

References

Abu Zitar R A and Hassoun M H 1993 Regulator control via genetic search and assisted reinforcement *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 254–62

Almássy N and Verschure P 1992 Optimizing self-organising control architectures with genetic algorithms: the interaction between natural selection and ontogenesis *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 451–60

Altman E R, Agarwal V K and Gao G R 1993 A novel methodology using genetic algorithms for the design of caches and cache replacement policy *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 392–9

Axelrod R 1987 The evolution of strategies in the iterated prisoner's dilemma *Genetic Algorithms and Simulated Annealing* ed L Davis (Boston, MA: Pitman) ch 3, pp 32–41

Baba N 1992 Utilization of stochastic automata and genetic algorithms for neural network learning *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 431–40

Bagchi S, Uckun S, Miyabe Y and Kawamura K 1991 Exploring problem-specific recombination operators for job shop scheduling *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R Belew and L Booker (San Mateo, CA: Morgan Kaufmann) pp 10–7

Bala J W and Wechsler H 1993 Learning to detect targets using scale-space and genetic search *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 516–22

Biegel J E and Davern J J 1990 Genetic algorithms and job shop scheduling *Comput. Indust. Eng.* **19** 81–91

Blanton J L and Wainwright R L 1993 Multiple vehicle routing with time and capacity constraints *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 452–9

Booker L 1985 Improving the performance of genetic algorithms in classifier systems *Proc. 1st Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1985)* ed J J Grefenstette (Hillsdale, NJ: Lawrence Erlbaum Associates) pp 80–92

Bramlette M F and Bouchard E E 1991 Genetic algorithms in parametric design of aircraft *Handbook of Genetic Algorithms* ed L Davis (New York: Van Nostrand Reinhold) ch 10, pp 109–23

Cartwright H M and Tuson A L 1994 Genetic algorithms and flowshop scheduling: towards the development of a real-time process control system *Evolutionary Computing (AISB Workshop, Leeds, 1994, Selected Papers) (Lecture Notes in Computer Science 865)* ed T C Fogarty (Berlin: Springer) pp 277–90

Chan H, Mazumder P and Shahookar K 1991 Macro-cell and module placement by genetic adaptive search with bitmap-represented chromosome *Integration VLSI J.* **12** 49–77

Cohoon J P and Paris W D 1987 Genetic placement *IEEE Trans. Computer-Aided Design* **CAD-6** 956–64

Corne D, Ross P and Fang H-L 1994 Fast practical evolutionary timetabling *Evolutionary Computing (AISB Workshop, Leeds, 1994, Selected Papers) (Lecture Notes in Computer Science 865)* ed T C Fogarty (Berlin: Springer) pp 250–63

Cox L A, Davis L and Qiu Y 1991 Dynamic anticipatory routing in circuit-switched telecommunications networks *Handbook of Genetic Algorithms* ed L Davis (New York: Van Nostrand Reinhold) ch 11, pp 124–43

Davidor Y 1991 A genetic algorithm applied to robot trajectory generation *Handbook of Genetic Algorithms* ed L Davis (New York: Van Nostrand Reinhold) ch 12, pp 144–65

Davis L 1985 Job shop scheduling with genetic algorithms *Proc. 1st Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1985)* ed J J Grefenstette (Hillsdale, NJ: Lawrence Erlbaum Associates) pp 136–40

Davis L and Cox A 1993 A genetic algorithm for survivable network design *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 408–15

DeJong K 1980 Adaptive system design: a genetic approach *IEEE Trans. Systems, Man Cybern.* **SMC-10** 566–74

Easton F F and Mansour N 1993 A distributed genetic algorithm for employee staffing and scheduling problems *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 360–67

Etter D M, Hicks M J and Cho K H 1982 Recursive adaptive filter design using an adaptive genetic algorithm *IEEE Int. Conf. on Acoutics, Speech and Signal Processing* (Piscataway, NJ: IEEE) pp 635–8

Fairley A and Yates D F 1994 Inductive operators and rule repair in a hybrid genetic learning system: some initial results *Evolutionary Computing (AISB Workshop, Leeds, 1994, Selected Papers) (Lecture Notes in Computer Science 865)* ed T C Fogarty (Berlin: Springer) pp 166–79

Fang H-L, Ross P and Corne D 1993 A promising genetic algorithm approach to job-shop scheduling, rescheduling and open-shop scheduling problems *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 375–82

Feldman D S 1993 Fuzzy network synthesis with genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 312–7

Flockton S J and White M 1993 Pole-zero system identification using genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 531–5

Fogarty T C 1994 Co-evolving co-operative populations of rules in learning control systems *Evolutionary Computing (AISB Workshop, Leeds, 1994, Selected Papers) (Lecture Notes in Computer Science 865)* ed T C Fogarty (Berlin: Springer) pp 195–209

Fogel D B 1988 An evolutionary approach to the traveling salesman problem *Biol. Cybernet.* **6** 139–44

——1990 A parallel processing approach to a multiple traveling salesman problem using evolutionary programming *Proc. 4th Ann. Symp. on Parallel Processing* (Piscataway, NJ: IEEE) pp 318–26

——1991 *System Identification through Simulated Evolution* (Needham, MA: Ginn)

——1993a Applying evolutionary programming to selected traveling salesman problems *Cybernet. Syst.* **24** 27–36

——1993b Evolving behaviors in the iterated prisoner's dilemma *Evolut. Comput.* **1** 77–97

——1995 *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence* (Piscataway, NJ: IEEE)

Fogel D B and Fogel L J 1996 Using evolutionary programming to schedule tasks on a suite of heterogeneous computers *Comput. Operat. Res.* **23** 527–34

Fogel D B, Fogel L J and Porto V W 1990 Evolving neural networks *Biol. Cybern.* **63** 487–93

Fogel L J, Owens A J and Walsh M J 1966 *Artificial intelligence Through Simulated Evolution* (New York: Wiley)

Fonseca C M and Fleming P J 1993 Genetic algorithms for multiobjective optimization: formulation, discussion and generalization *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 416–23

Fonseca C M, Mendes E M, Fleming P J and Billings S A 1993 Non-linear model term selection with genetic algorithms *Natural Algorithms in Signal Processing (Workshop, Chelmsford, UK, November 1993)* vol 2 (London: IEE) pp 27/1–27/8

Fourman M P 1985 Compaction of symbolic layout using genetic algorithms *Proc. 1st Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1985)* ed J J Grefenstette (Hillsdale, NJ: Lawrence Erlbaum Associates) pp 141–53

Fujita K, Akagi S and Hirokawa N 1993 Hybrid approach for optimal nesting using genetic algorithm and a local minimization algorithm *Advances in Design Automation* vol 1, DE-65–1 (ASME) pp 477–84

Furuya H and Haftka R T 1993 Genetic algorithms for placing actuators on space structures *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 536–42

Gehlhaar D K, Verkhivker G M, Rejto P A, Sherman C J, Fogel D B, Fogel L J and Freer S T 1995 Molecular recognition of the inhibitor AG-1343 by HIV-1 protease: conformationally flexible docking by evolutionary programming *Chem. Biol.* **2** 317–24

Goldberg D E and Lingle R 1985 Alleles, loci and the travelling salesman problem *Proc. 1st Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1985)* ed J J Grefenstette (Hillsdale, NJ: Lawrence Erlbaum Associates) pp 154–9

Grefenstette J J 1987 Incorporating problem specific knowledge into genetic algorithms *Genetic Algorithms and Simulated Annealing* ed L Davis (Boston, MA: Pitman) ch 4, pp 42–60

Gruau F 1993 Genetic synthesis of modular neural networks *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 318–25

Güvenir H A and Şirin I 1993 A genetic algorithm for classification by feature recognition *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 543–8

Hancock P 1992 Recombination operators for the design of neural nets by genetic algorithm *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 441–50

Handley S 1993 Automated learning of a detector for α-helices in protein sequences via genetic programming *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 271–8

Harp S A and Samad T 1991 Genetic synthesis of neural network architecture *Handbook of Genetic Algorithms* ed L Davis (New York: Van Nostrand Reinhold) ch 15, pp 202–21

Holland J H 1985 Properties of the bucket-brigade algorithm *Proc. 1st Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1985)* ed J J Grefenstette (Hillsdale, NJ: Lawrence Erlbaum Associates) pp 1–7

——1987 Genetic algorithms and classifier systems: foundations and future directions *Proc. 2nd Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1985)* ed J J Grefenstette (Hillsdale, NJ: Lawrence Erlbaum Associates) pp 82–9

Holland J H, Holyoak K J, Nisbett R E and Thagard P R 1987 Classifier systems, Q-morphisms and induction *Genetic Algorithms and Simulated Annealing* ed L Davis (Boston, MA: Pitman) ch 9, pp 116–28

Homaifar, A., Guan S and Liepins G 1993 A new approach to the travelling salesman problem by genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 460–6

Hunt K J 1992a Optimal control system synthesis with genetic algorithms *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 381–9

——1992b Polynomial LQG and H_∞ controller synthesis: a genetic algorithm solution *Proc. IEEE Conf. on Decision and Control (Tuscon, AZ)* (Picataway, NJ: IEEE)

Husbands P 1993 An ecosystems model for integrated production planning *Int. J. Comput. Integrated Manufacturing* **6** 74–86

Ifeachor E C and Harris S P 1993 A new approach to frequency sampling filter design using genetic algorithms *Natural Algorithms in Signal Processing (Workshop, Chelmsford, UK, November 1993)* vol 1 (London: IEE) pp 5/1–5/8

Jakob W, Gorges-Schleuter M and Blume C 1992 Application of genetic algorithms to task planning and learning *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 291–300

Janikow C Z and Cai H 1992 A genetic algorithm application in nonparametric functional estimation *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 249–58

Jones G, Brown R D, Clark D E, Willett P and Glen R C 1993 Searching databases of two-dimensional and three dimensional chemical structures using genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 597–602

Juliff K 1993 A multi-chromosome genetic algorithm for pallet loading *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 467–73

Kidwell M D 1993 Using genetic algorithms to schedule distributed tasks on a bus-based system *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 368–74

Kim J-H and Shim H-S 1995 Evolutionary programming-based optimal robust locomotion control of autonomous mobile robots *Proc. 4th Ann. Conf. on Evolutionary Programming* ed J R McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press) pp 631–44

Krishnakumar K and Goldberg D E 1990 Genetic algorithms in control system optimization. *Proc. AIAA Guidance, Navigation, and Control Conf. (Portland, OR)* pp 1568–77

——1992 Control system optimization using genetic algorithms *J. Guidance Control Dynam.* **15** 735–40

Ling S-E 1992 Integrating genetic algorithms with a prolog assignment program as a hybrid solution for a polytechnic timetable problem *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 321–9

Lohmann R 1992 Structure evolution and incomplete induction *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 175–85

Louis S J and Rawlins G J E 1991 Designer genetic algorithms: genetic algorithms in structure design *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, July 1991)* ed R Belew and L Booker (San Mateo, CA: Morgan Kaufmann) pp 53–60

Lucasius C B, Blommers M J, Buydens L M and Kateman G 1991 A genetic algorithm for conformational analysis of DNA *Handbook of Genetic Algorithms* ed L Davis (New York: Van Nostrand Reinhold) ch 18, pp 251–81

Lucasius C B and Kateman G 1992 Towards solving subset selection problems with the aid of the genetic algorithm *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 239–47

Manela, M., Thornhill N and Campbell J A 1993 Fitting spline functions to noisy data using a genetic algorithm *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 549–56

McDonnell J R, Andersen B L, Page W C and Pin F G 1992 Mobile manipulator configuration optimization using evolutionary programming *Proc. 1st Ann. Conf. on Evolutionary Programming* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 52–62

Melhuish C and Fogarty T C 1994 Applying a restricted mating policy to determine state space niches using immediate and delayed reinforcement *Evolutionary Computing (AISB Workshop, Leeds, 1994, Selected Papers) (Lecture Notes in Computer Science 865)* ed T C Fogarty (Berlin: Springer) pp 224–37

Michalewicz Z 1992 *Genetic Algorithms + Data Structures = Evolution Programs* (Berlin: Springer)

——1993 A hierarchy of evolution programs: an experimental study *Evolut. Comput.* **1** 51–76

Miller G F, Todd P M and Hegde S U 1989 Designing neural networks using genetic algorithms. *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 379–84

Mühlenbein H 1989 Parallel genetic algorithms, population genetics and combinatorial optimization *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 416–21

Namibar R and Mars P 1993 Adaptive IIR filtering using natural algorithms *Natural Algorithms in Signal Processing (Workshop, Chelmsford, UK, November 1993)* vol 2 (London: IEE) pp 20/1–20/10

Oliver J R 1993 Discovering individual decision rules: an application of genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 216–22

Oliver I M, Smith D J and Holland J R C 1987 A study of permutation crossover operators on the travelling salesman problem *Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 224–30

Page W C, McDonnell J R and Anderson B 1992 An evolutionary programming approach to multi-dimensional path planning *Proc. 1st Ann. Conf. on Evolutionary Programming* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 63–70

Parker J K, Goldberg D E and Khoogar A R 1989 Inverse kinematics of redundant robots using genetic algorithms *Proc. Int. Conf. on Robotics and Automation (Scottsdale, AZ, 1989)* vol 1 (Los Alamitos: IEEE Computer Society Press) pp 271–6

Patel M J and Dorigo M 1994 Adaptive learning of a robot arm *Evolutionary Computing (AISB Workshop, Leeds, 1994, Selected Papers) (Lecture Notes in Computer Science 865)* ed T C Fogarty (Berlin: Springer) pp 180–94

Pipe A G and Carse B 1994 A comparison between two architectures for searching and learning in maze problems *Evolutionary Computing (AISB Workshop, Leeds, 1994, Selected Papers) (Lecture Notes in Computer Science 865)* ed T C Fogarty (Berlin: Springer) pp 238–49

Polani D and Uthmann T 1992 Adaptation of Kohonen feature map topologies by genetic algorithms *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 421–9

——1993 Training Kohonen feature maps in different topologies: an analysis using genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 326–33

Porto V W, Fogel D B and Fogel L J 1995 Alternative neural network training methods *IEEE Expert* **10** 16–22

Powell D and Skolnick M M 1993 Using genetic algorithms in engineering design optimization with non-linear constraints *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 424–31

Punch W F, Goodman E D, Pei, M, Chia-Shun L, Hovland P and Enbody R 1993 Further research on feature selection and classification using genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 557–64

Rahmani A T and Ono N 1993 A genetic algorithm for channel routing problem *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 494–8

Roberts A and Wade G 1993 A structured GA for FIR filter design *Natural Algorithms in Signal Processing (Workshop, Chelmsford, UK, November 1993)* vol 1 (London: IEE) pp 16/1–16/8

Robertson G 1987 Parallel implementation of genetic algorithms in a classifier system *Genetic Algorithms and Simulated Annealing* ed L Davis (Boston, MA: Pitman) ch 10, pp 129–40

Romaniuk S G 1993 Evolutionary growth perceptrons *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 334–41

Roosen P and Meyer F 1992 Determination of chemical equilibria by means of an evolution strategy *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 411–20

San Martin R and Knight J P 1993 Genetic algorithms for optimization of integrated circuit synthesis *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 432–8

Schaffer J D and Eshelman L J 1993 Designing multiplierless digital filters using genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 439–44

Schulze-Kremer S 1992 Genetic algorithms for protein tertiary structure prediction *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 391–400

Smith A E and Tate D M 1993 Genetic optimization using a penalty function *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 499–505

Smith D 1985 Bin packing with adaptive search *Proc. 1st Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1985)* ed J J Grefenstette (Hillsdale, NJ: Lawrence Erlbaum Associates)

Spencer G F 1993 Automatic generation of programs for crawling and walking *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) p 654

Spittle M C and Horrocks D H 1993 Genetic algorithms and reduced complexity artificial neural networks *Natural Algorithms in Signal Processing (Workshop, Chelmsford, UK, November 1993)* vol 1 (London: IEE) pp 8/1–8/9

Suckley D 1991 Genetic algorithm in the design of FIR filters *IEE Proc.* G **138** 234–8

Syswerda G 1991 Schedule optimization using genetic algorithms *Handbook of Genetic Algorithms* ed L Davis(New York: Van Nostrand Reinhold) ch 21, pp 332–49

Tackett W A 1993 Genetic programming for feature discovery and image discrimination *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 303–9

Tanaka, Y., Ishiguro A and Uchikawa Y 1993 A genetic algorithms application to inverse problems in electromagnetics *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) p 656

Thangia S R, Vinayagamoorthy R and Gubbi A V 1993 Vehicle routing with time deadlines using genetic and local algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 506–13

Unger R and Moult J 1993 A genetic algorithm for 3D protein folding simulations *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 581–8

Van Driessche R and Piessens R 1992 Load balancing with genetic algorithms *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 341–50

Verhoeven M G A, Aarts E H L, van de Sluis E and Vaessens R J M 1992 Parallel local search and the travelling salesman problem *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 543–52

Watabe H and Okino N 1993 A study on genetic shape design *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 445–50

White M and Flockton S 1993 A comparative study of natural algorithms for adaptive IIR filtering *Natural Algorithms in Signal Processing (Workshop, Chelmsford, UK, November 1993)* vol 2 (London: IEE) pp 22/1–22/8

Whitley D, Starkweather T and Fuquay D 1989 Scheduling problems and travelling salesmen: the genetic edge recombination operator *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 133–40

Wicks T and Lawson S 1993 Genetic algorithm design of wave digital filters with a restricted coefficient set *Natural Algorithms in Signal Processing (Workshop, Chelmsford, UK, November 1993)* vol 1 (London: IEE) pp 17/1–17/7

Wilson P B and Macleod M D 1993 Low implementation cost IIR digital filter design using genetic algorithms *Natural Algorithms in Signal Processing (Workshop, Chelmsford, UK, November 1993)* vol 1 (London: IEE) pp 4/1–4/8

Wilson S W 1987 Hierarchical credit allocation in a classifier system *Genetic Algorithms and Simulated Annealing* ed L Davis (Boston, MA: Pitman) ch 8, pp 104–15

Yamada T and Nakano R 1992 A genetic algorithm applicable to large-scale job-shop problems *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 281–90

Yang J-J and Korfhage R R 1993 Query optimization in information retrieval using genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 603–11

Zhang B-T and Mühlenbein H 1993 Genetic programming of minimal neural nets using Occam's razor *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 342–9

Further reading

This article has provided only a glimpse into the range of applications for evolutionary computing. A series of comprehensive bibliographies has been produced by J T Alander of the Department of Information Technology and Production Economics, University of Vaasa, as listed below.

1. **Art and Music:** *Indexed Bibliography of Genetic Algorithms in Art and Music* Report 94-1-ART (ftp.uwasa.fi/cs/report94-1/gaARTbib.ps.Z)
2. **Chemistry and Physics:** *Indexed Bibliography of Genetic Algorithms in Chemistry and Physics* Report 94-1-CHEMPHYS (ftp.uwasa.fi/cs/report94-1/gaCHEMPHYSbib.ps.Z)
3. **Control:** *Indexed Bibliography of Genetic Algorithms in Control.* Report 94-1-CONTROL (ftp.uwasa.fi/cs/report94-1/gaCONTROLbib.ps.Z)
4. **Computer Aided Design:** *Indexed Bibliography of Genetic Algorithms in Computer Aided Design* Report 94-1-CAD (ftp.uwasa.fi/cs/report94-1/gaCADbib.ps.Z)
5. **Computer Science:** *Indexed Bibliography of Genetic Algorithms in Computer Science* Report 94-1-CS (ftp.uwasa.fi/cs/report94-1/gaCSbib.ps.Z)
6. **Economics:** *Indexed Bibliography of Genetic Algorithms in Economics* Report 94-1-ECO (ftp.uwasa.fi/cs/report94-1/gaECObib.ps.Z)
7. **Electronics and VLSI Design and Testing:** *Indexed Bibliography of Genetic Algorithms in Electronics and VLSI Design and Testing* Report 94-1-VLSI (ftp.uwasa.fi/cs/report94-1/gaVLSIbib.ps.Z)
8. **Engineering:** *Indexed Bibliography of Genetic Algorithms in Engineering* Report 94-1-ENG (ftp.uwasa.fi/cs/report94-1/gaENGbib.ps.Z)
9. **Fuzzy Systems:** *Indexed Bibliography of Genetic Algorithms and Fuzzy Systems* Report 94-1-FUZZY (ftp.uwasa.fi/cs/report94-1/gaFUZZYbib.ps.Z)
10. **Logistics:** *Indexed Bibliography of Genetic Algorithms in Logistics* Report 94-1-LOGISTICS (ftp.uwasa.fi/cs/report94-1/gaLOGISTICSbib.ps.Z)
11. **Manufacturing:** *Indexed Bibliography of Genetic Algorithms in Manufacturing* Report 94-1-MANU (ftp.uwasa.fi/cs/report94-1/gaMANUbib.ps.Z)
12. **Neural Networks:** *Indexed Bibliography of Genetic Algorithms and Neural Networks* Report 94-1-NN (ftp.uwasa.fi/cs/report94-1/gaNNbib.ps.Z)
13. **Optimization:** *Indexed Bibliography of Genetic Algorithms and Optimization* Report 94-1-OPTIMI (ftp.uwasa.fi/cs/report94-1/gaOPTIMIbib.ps.Z)
14. **Operations Research:** *Indexed Bibliography of Genetic Algorithms in Operations Research* Report 94-1-OR (ftp.uwasa.fi/cs/report94-1/gaORbib.ps.Z)
15. **Power Engineering:** *Indexed Bibliography of Genetic Algorithms in Power Engineering* Report 94-1-POWER (ftp.uwasa.fi/cs/report94-1/gaPOWERbib.ps.Z)
16. **Robotics:** *Indexed Bibliography of Genetic Algorithms in Robotics* Report 94-1-ROBO (ftp.uwasa.fi/cs/report94-1/gaROBObib.ps.Z)
17. **Signal and Image Processing:** *Indexed Bibliography of Genetic Algorithms in Signal and Image Processing* Report 94-1-SIGNAL (ftp.uwasa.fi/cs/report94-1/gaSIGNALbib.ps.Z)

A1.3 Advantages (and disadvantages) of evolutionary computation over other approaches

Hans-Paul Schwefel

Abstract

The attractiveness of evolutionary algorithms is obvious from the many successful applications already and the huge number of publications in the field of evolutionary computation. Trying to offer hard facts about comparative advantages in general, however, turns out to be difficult—if not impossible. One reason for this is the so-called no-free-lunch (NFL) theorem.

A1.3.1 No-free-lunch theorem

Since, according to the no-free-lunch (NFL) theorem (Wolpert and Macready 1996), there cannot exist any algorithm for solving *all* (e.g. optimization) problems that is generally (on average) superior to any competitor, the question of whether evolutionary algorithms (EAs) are inferior/superior to any alternative approach is senseless. What could be claimed solely is that EAs behave better than other methods with respect to solving a specific class of problems—with the consequence that they behave worse for other problem classes.

The NFL theorem can be corroborated in the case of EAs versus many classical optimization methods insofar as the latter are more efficient in solving linear, quadratic, strongly convex, unimodal, separable, and many other special problems. On the other hand, EAs do not give up so early when discontinuous, nondifferentiable, multimodal, noisy, and otherwise unconventional response surfaces are involved. Their effectiveness (or robustness) thus extends to a broader field of applications, of course with a corresponding loss in efficiency when applied to the classes of simple problems classical procedures have been specifically devised for.

Looking into the historical record of procedures devised to solve optimization problems, especially around the 1960s (see the book by Schwefel (1995)), when a couple of direct optimum-seeking algorithms were published, for example, in the *Computer Journal*, a certain pattern of development emerges. Author A publishes a procedure and demonstrates its suitability by means of tests using some test functions. Next, author B comes along with a counterexample showing weak performance of A's algorithm in the case of a certain test problem. Of course, he also presents a new or modified technique that outperforms the older one(s) with respect to the additional test problem. This game could in principle be played *ad infinitum.*

A better means of clarifying the scene ought to result from theory. This should clearly define the domain of applicability of each algorithm by presenting convergence proofs and efficiency results. Unfortunately, however, it is possible to prove abilities of algorithms only by simplifying them as well as the situations to which they are confronted. The huge remainder of questions must be answered by means of (always limited) test series, and even that cannot tell much about an actual real-world problem-solving situation with yet unanalyzed features, that is, the normal case in applications.

Again unfortunately, there does not exist an agreed-upon test problem catalogue to evaluate old as well as new algorithms in a concise way. It is doubtful whether such a test bed will ever be agreed upon, but efforts in that direction would be worthwhile.

A1.3.2 Conclusions

Finally, what are the truths and consequences? First, there will always remain a dichotomy between efficiency and general applicability, between reliability and effort of problem-solving, especially optimum-seeking, algorithms. Any specific knowledge about the situation at hand may be used to specify an adequate specific solution algorithm, the optimal situation being that one knows the solution in advance. On the other hand, there cannot exist one method that solves all problems effectively as well as efficiently. These goals are contradictory.

If there is already a traditional method that solves a given problem, EAs should not be used. They cannot do it better or with less computational effort. In particular, they do not offer an escape from the curse of dimensionality—the often quadratic, cubic, or otherwise polynomial increase in instructions used as the number of decision variables is increased, arising, for example, from matrix manipulation.

To develop a new solution method suitable for a problem at hand may be a nice challenge to a theoretician, who will afterwards get some merit for his effort, but from the application point of view the time for developing the new technique has to be added to the computer time invested. In that respect, a nonspecialized, robust procedure (and EAs belong to this class) may be, and often proves to be, worthwhile.

A warning should be given about a common practice—the linearization or other decomplexification of the situation in order to make a traditional method applicable. Even a guaranteed globally optimal solution for the simplified task may be a long way off and thus largely inferior to an approximate solution to the real problem.

The best one can say about EAs, therefore, is that they present a methodological framework that is easy to understand and handle, and is either usable as a black-box method or open to the incorporation of new or old recipes for further sophistication, specialization or hybridization. They are applicable even in dynamic situations where the goal or constraints are moving over time or changing, either exogenously or self-induced, where parameter adjustments and fitness measurements are disturbed, and where the landscape is rough, discontinuous, multimodal, even fractal or cannot otherwise be handled by traditional methods, especially those that need global prediction from local surface analysis.

C4.5, F1.9 There exist EA versions for *multiple criteria decision making* (MCDM) and many different parallel computing architectures. Almost forgotten today is their applicability in experimental (non-computing) situations.

E1 Sometimes striking is the fact that even obviously wrong *parameter settings* do not prevent fairly good results: this certainly can be described as robustness. Not yet well understood, but nevertheless very successful are those EAs which self-adapt some of their internal parameters, a feature that can be described as collective learning of the environmental conditions. Nevertheless, even self-adaptation does not circumvent the NFL theorem.

In this sense, and only in this sense, EAs always present an intermediate compromise; the enthusiasm of its inventors is not yet taken into account here, nor the insights available from the analysis of the algorithms for natural evolutionary processes which they try to mimic.

References

Schwefel H-P 1995 *Evolution and Optimum Seeking* (New York: Wiley)

Wolpert D H and Macready W G 1996 *No Free Lunch Theorems for Search* Technical Report SFI-TR-95-02-010 Santa Fe Institute

A2

Evolutionary Computation: The Background

Contents

A2.1 Principles of evolutionary processes

David B Fogel

Abstract

The principles of evolution are considered. Evolution is seen to be the inevitable outcome of the interaction of four essential processes: reproduction, competition, mutation, and selection. Consideration is given to the duality of natural organisms in terms of their genotypes and phenotypes, as well as to characterizing evolution in terms of adaptive landscapes.

A2.1.1 Overview

The most widely accepted collection of evolutionary theories is the neo-Darwinian paradigm. These arguments assert that the vast majority of the history of life can be fully accounted for by physical processes operating on and within populations and species (Hoffman 1989, p 39). These processes are reproduction, mutation, competition, and selection. Reproduction is an obvious property of extant species. Further, species have such great reproductive potential that their population size would increase at an exponential rate if all individuals of the species were to reproduce successfully (Malthus 1826, Mayr 1982, p. 479). Reproduction is accomplished through the transfer of an individual's genetic program (either asexually or sexually) to progeny. Mutation, in a positively entropic system, is guaranteed, in that replication errors during information transfer will necessarily occur. Competition is a consequence of expanding populations in a finite resource space. Selection is the inevitable result of competitive replication as species fill the available space. Evolution becomes the inescapable result of interacting basic physical statistical processes (Huxley 1963, Wooldridge 1968, Atmar 1979).

Individuals and species can be viewed as a duality of their genetic program, the *genotype*, and their expressed behavioral traits, the *phenotype*. The genotype provides a mechanism for the storage of experiential evidence, of historically acquired information. Unfortunately, the results of genetic variations are generally unpredictable due to the universal effects of pleiotropy and polygeny (figure A2.1.1) (Mayr 1959, 1963, 1982, 1988, Wright 1931, 1960, Simpson 1949, p 224, Dobzhansky 1970, Stanley 1975, Dawkins 1986). Pleiotropy is the effect that a single gene may simultaneously affect several phenotypic traits. Polygeny is the effect that a single phenotypic characteristic may be determined by the simultaneous interaction of many genes. There are no one-gene, one-trait relationships in naturally evolved systems. The phenotype varies as a complex, nonlinear function of the interaction between underlying genetic structures and current environmental conditions. Very different genetic structures may code for equivalent behaviors, just as diverse computer programs can generate similar functions. A2.2.2

Selection directly acts only on the expressed behaviors of individuals and species (Mayr 1988, pp 477–8). Wright (1932) offered the concept of adaptive topography to describe the fitness of individuals and species (minimally, isolated reproductive populations termed demes). A population of genotypes maps to respective phenotypes (*sensu* Lewontin 1974), which are in turn mapped onto the adaptive topography (figure A2.1.2). Each peak corresponds to an optimized collection of phenotypes, and thus to one of more sets of optimized genotypes. Evolution probabilistically proceeds up the slopes of the topography toward peaks as selection culls inappropriate phenotypic variants.

Others (Atmar 1979, Raven and Johnson 1986, pp 400–1) have suggested that it is more appropriate to view the adaptive landscape from an inverted position. The peaks become troughs, 'minimized prediction

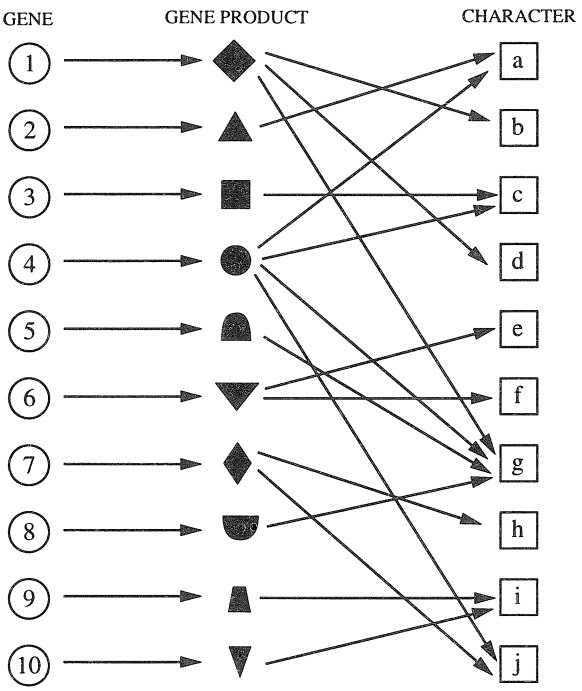


Figure A2.1.1. Pleiotropy is the effect that a single gene may simultaneously affect several phenotypic traits. Polygeny is the effect that a single phenotypic characteristic may be determined by the simultaneous interaction of many genes. These one-to-many and many-to-one mappings are pervasive in natural systems. As a result, even small changes to a single gene may induce a raft of behavioral changes in the individual (after Mayr 1963).

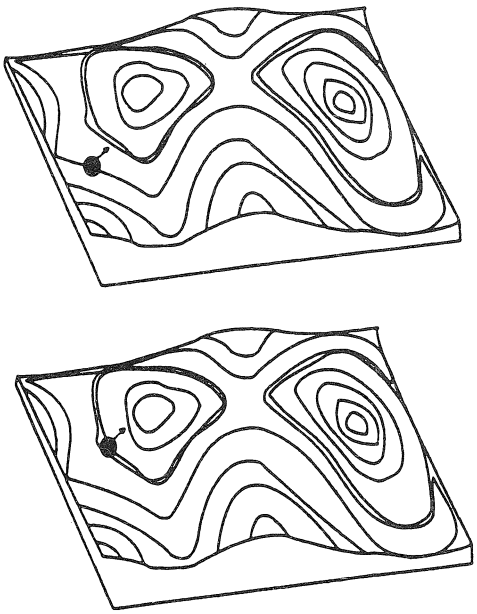


Figure A2.1.2. Wright's adaptive topography, inverted. An adaptive topography, or adaptive landscape, is defined to represent the fitness of all possible phenotypes (generated by the interaction between the genotypes and the environment). Wright (1932) proposed that as selection culls the last appropriate existing behaviors relative to others in the population, the population advances to areas of higher fitness on the landscape. Atmar (1979) and others have suggested viewing the topography from an inverted perspective. Populations advance to areas of lower behavioral error.

error entropy wells' (Atmar 1979). Searching for peaks depicts evolution as a slowly advancing, tedious, uncertain process. Moreover, there appears to be a certain fragility to an evolving phyletic line; an optimized population might be expected to quickly fall of the peak under slight perturbations. The inverted topography leaves an altogether different impression. Populations advance rapidly down the walls of the error troughs until their cohesive set of interrelated behaviors is optimized, at which point stagnation occurs. If the topography is generally static, rapid descents will be followed by long periods of stasis. If, however, the topography is in continual flux, stagnation may never set in.

Viewed in this manner, evolution is an obvious optimizing problem-solving process (not to be confused with a process that leads to perfection). Selection drives phenotypes as close to the optimum as possible, given initial conditions and environment constraints. However the environment is continually changing. Species lag behind, constantly evolving toward a new optimum. No organism should be viewed as being perfectly adapted to its environment. The suboptimality of behavior is to be expected in any dynamic environment that mandates tradeoffs between behavioral requirements. However selection never ceases to operate, regardless of the population's position on the topography.

Mayr (1988, p 532) has summarized some of the more salient characteristics of the neo-Darwinian paradigm. These include:

(i) The individual is the primary target of selection.
(ii) Genetic variation is largely a chance phenomenon. Stochastic processes play a significant role in evolution.
(iii) Genotypic variation is largely a product of recombination and 'only ultimately of mutation'.
(iv) 'Gradual' evolution may incorporate phenotypic discontinuities.
(v) Not all phenotypic changes are necessarily consequences of *ad hoc* natural selection.
(vi) Evolution is a change in adaptation and diversity, not merely a change in gene frequencies.
(vii) Selection is probabilistic, not deterministic.

These characteristics form a framework for evolutionary computation.

References

Atmar W 1979 The inevitability of evolutionary invention, unpublished manuscript

Dawkins R 1986 *The Blind Watchmaker* (Oxford: Clarendon)

Dobzhansky T 1970 *Genetics of the Evolutionary Processes* (New York: Columbia University Press)

Hoffman A 1989 *Arguments on Evolution: a Paleontologist's Perspective* (New York: Oxford University Press)

Huxley J 1963 The evolutionary process *Evolution as a Process* ed J Huxley, A C Hardy and E B Ford (New York: Collier) pp 9–33

Lewontin R C 1974 *The Genetic Basis of Evolutionary Change* (New York: Columbia University Press)

Malthus T R 1826 *An Essay on the Principle of Population, as it Affects the Future Improvement of Society* 6th edn (London: Murray)

Mayr E 1959 Where are we? *Cold Spring Harbor Symp. Quant. Biol.* **24** 409–40

——1963 *Animal Species and Evolution* (Cambridge, MA: Belknap)

——1982 *The Growth of Biological Thought: Diversity, Evolution and Inheritance* (Cambridge, MA: Belknap)

——1988 *Toward a New Philosophy of Biology: Observations of an Evolutionist* (Cambridge, MA: Belknap)

Raven P H and Johnson G B 1986 *Biology* (St Louis, MO: Times Mirror)

Simpson G G 1949 *The Meaning of Evolution: a Study of the History of Life and its Significance for Man* (New Haven, CT: Yale University Press)

Stanley S M 1975 A theory of evolution above the species level *Proc. Natl Acad. Sci. USA* **72** 646–50

Wooldridge D E 1968 *The Mechanical Man: the Physical Basis of Intelligent Life* (New York: McGraw-Hill)

Wright S 1931 Evolution in Mendelian populations *Genetics* **16** 97–159

——1932 The roles of mutation, inbreeding, crossbreeding, and selection in evolution *Proc. 6th Int. Congr. on Genetics (Ithaca, NY)* vol 1, pp 356–66

——1960 The evolution of life, panel discussion *Evolution After Darwin: Issues in Evolution* vol 3, ed S Tax and C Callender (Chicago, IL: University of Chicago Press)

A2.2 Principles of genetics

Raymond C Paton

Abstract

The purpose of this section is to provide the reader with a general overview of the biological background to evolutionary computing. This is not a central issue to understanding how evolutionary algorithms work or how they can be applied. However, many biological terms have been reapplied in evolutionary computing and many researchers seek to introduce new ideas from biological sources. It is hoped to provide valuable background for such readers.

A2.2.1 Introduction

The material covers a number of key areas which are necessary to understanding the nature of the evolutionary process. We begin by looking at some basic ideas of heredity and how variation occurs in interbreeding populations. From here we look at the gene in more detail and then consider how it can undergo change. The next section looks at aspects of population thinking needed to appreciate selection. This is crucial to an appreciation of Darwinian mechanisms of evolution. The article concludes with selected references to further information. In order to keep this contribution within its size limits, the material is primarily about the biology of higher plants and animals.

A2.2.2 Some fundamental concepts in genetics

Many plants and animals are produced through sexual means by which the nucleus of a male sperm cell fuses with a female egg cell (ovum). Sperm and ovum nuclei each contain a single complement of nuclear material arranged as ribbon-like structures called chromosomes. When a sperm fuses with an egg the resulting cell, called a zygote, has a double complement of chromosomes together with the cytoplasm of the ovum. We say that a single complement of chromosomes constitutes a haploid set (abbreviated as n)

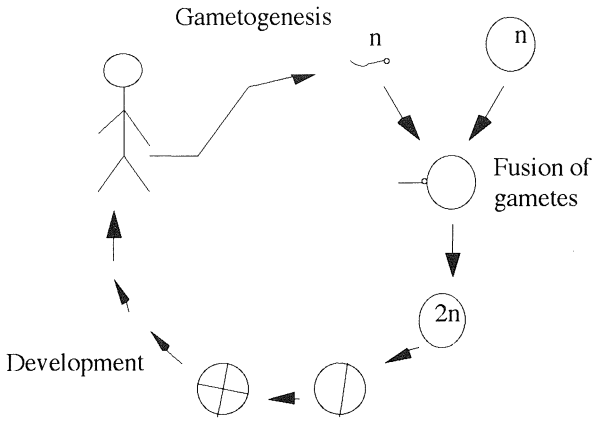


Figure A2.2.1. A common life cycle model.

and a double complement is called the diploid set ($2n$). Gametes (sex cells) are haploid whereas most other cells are diploid. The formation of gametes (gametogenesis) requires the number of chromosomes in the gamete-forming cells to be halved (see figure A2.2.1).

Gametogenesis is achieved through a special type of cell division called meiosis (also called reduction division). The intricate mechanics of meiosis ensures that gametes contain only one copy of each chromosome.

A genotype is the genetic constitution that an organism inherits from its parents. In a diploid organism, half the genotype is inherited from one parent and half from the other. Diploid cells contain two copies of each chromosome. This rule is not universally true when it comes to the distribution of sex chromosomes. Human diploid cells contain 46 chromosomes of which there are 22 pairs and an additional two sex chromosomes. Sex is determined by one pair (called the sex chromosomes); female is X and male is Y. A female human has the sex chromosome genotype of XX and a male is XY. The inheritance of sex is summarized in figure A2.2.2. The members of a pair of nonsex chromosomes are said to be homologous (this is also true for XX genotypes whereas XY are not homologous).

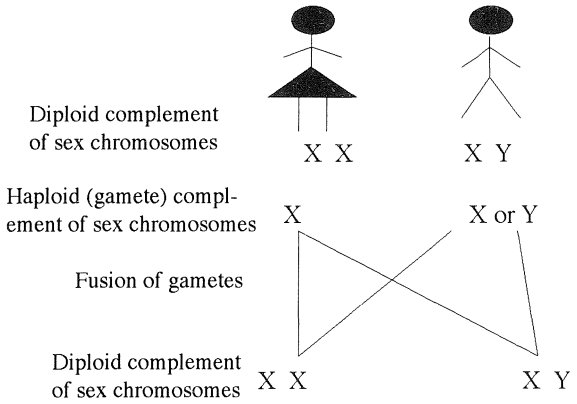


Figure A2.2.2. Inheritance of sex chromosomes.

Although humans have been selectively breeding domestic animals and plants for a long time, the modern study of genetics began in the mid-19th century with the work of Gregor Mendel. Mendel investigated the inheritance of particular traits in peas. For example, he took plants that had wrinkled seeds and plants that had round seeds and bred them with plants of the same phenotype (i.e. observable appearance), so wrinkled were bred with wrinkled and round were bred with round. He continued this over a number of generations until round always produced round offspring and wrinkled, wrinkled. These are called pure breeding plants. He then cross-fertilized the plants by breeding rounds with wrinkles. The subsequent generation (called the F1 hybrids) was all round. Then Mendel crossed the F1 hybrids with each other and found that the next generation, the F2 hybrids, had round and wrinkled plants in the ratio of 3 (round) : 1 (wrinkled).

Mendel did this kind of experiment with a number of pea characteristics such as:

color of cotyledons	yellow or green
color of flowers	red or white
color of seeds	gray/brown or white
length of stem	tall or dwarf.

In each case he found that the the F1 hybrids were always of one form and the two forms reappeared in the F2. Mendel called the form which appeared in the F1 generation dominant and the form which reappeared in the F2 recessive (for the full text of Mendel's experiments see an older genetics book, such as that by Sinnott *et al* (1958)).

A modern interpretation of inheritance depends upon a proper understanding of the nature of a gene and how the gene is expressed in the phenotype. The nature of a gene is quite complex as we shall see later (see also Alberts *et al* 1989, Lewin 1990, Futuyma 1986). For now we shall take it to be the functional unit of inheritance. An allele (allelomorph) is one of several forms of a gene occupying a given locus (location) on a chromosome. Originally related to pairs of contrasting characteristics (see examples

above), the idea of observable unit characters was introduced to genetics around the turn of this century by such workers as Bateson, de Vries, and Correns (see Darden 1991). The concept of a gene has tended to replace allele in general usage although the two terms are not the same.

How can the results of Mendel's experiments be interpreted? We know that each parent plant provides half the chromosome complement found in its offspring and that chromosomes in the diploid cells are in pairs of homologues. In the pea experiments pure breeding parents had homologous chromosomes which were identical for a particular gene; we say they are homozygous for a particular gene. The pure breeding plants were produced through self-fertilization and by selecting those offspring of the desired phenotype. As round was dominant to wrinkled we say that the round form of the gene is R ('big r') and the wrinkled r ('little r'). Figure A2.2.3 summarizes the cross of a pure breeding round (RR) with a pure breeding wrinkled (rr).

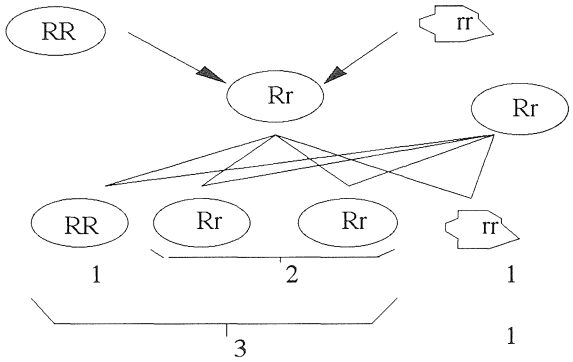


Figure A2.2.3. A simple Mendelian experiment.

We see the appearance of the heterozygote (in this case Rr) in the F1 generation. This is phenotypically the same as the dominant phenotype but genotypically contains both a dominant and a recessive form of the particular gene under study. Thus when the heterozygotes are randomly crossed with each other the phenotype ratio is three dominant : one recessive. This is called the monohybrid ratio (i.e. for one allele). We see in Mendel's experiments the independent segregation of alleles during breeding and their subsequent independent assortment in offspring.

In the case of two genes we find more phenotypes and genotypes appearing. Consider what happens when pure breeding homozygotes for round yellow seeds (RRYY) are bred with pure breeding homozygotes for wrinkled green seeds (rryy). On being crossed we end up with heterozygotes with a genotype of RrYy and phenotype of round yellow seeds. We have seen that the genes segregate independently during meiosis so we have the combinations shown in figure A2.2.4.

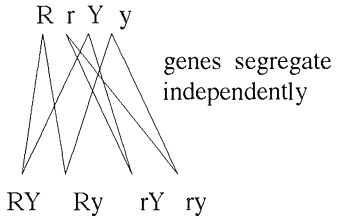


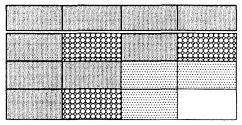
Figure A2.2.4. Genes segregating independently.

Thus the gametes of the heterozygote can be of four kinds though we assume that each form can occur with equal frequency. We may examine the possible combinations of gametes for the next generation by producing a contingency table for possible gamete combinations. These are shown in figure A2.2.5.

We summarize this set of genotype combinations in the phenotype table (figure A2.2.5(*b*)). The resulting ratio of phenotypes is called the dihybrid ratio (9:3:3:1). We shall consider one final example in this very brief summary. When pure breeding red-flowered snapdragons were crossed with pure breeding white-flowered plants the F1 plants were all pink. When these were selfed the population of offspring was in the ratio of one red : two pink : one white. This is a case of incomplete dominance in the heterozygote.

	RY	Ry	rY	ry
RY	RRYY	RRYy	RrYY	RrYy
Ry	RRYy	RRyy	RrYy	Rryy
rY	RrYY	RrYy	rrYY	rrYy
ry	RrYy	Rryy	rrYy	rryy

(a) Pattern of Genotypes

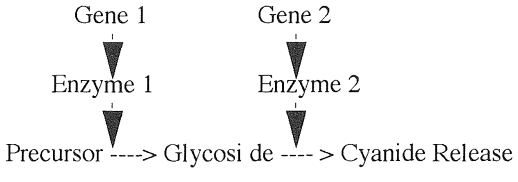


dominant for both genes (9)
dominant for one gene (3)
dominant for one gene (3)
recessive for both genes (1)

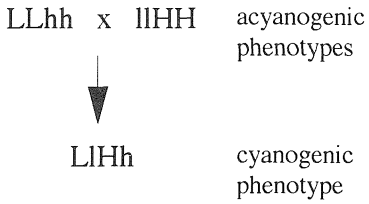
(b) Pattern of Phenotypes

Figure A2.2.5. Genotype and phenotype patterns in F2.

It has been found that the Mendelian ratios do not always apply in breeding experiments. In some cases this is because certain genes interact with each other. Epistasis occurs when the expression of one gene masks the phenotypic effects of another. For example, certain genotypes (cyanogenics) of clover can resist grazing because they produce low doses of cyanide which makes them unpalatable. Two genes are involved in cyanide production, one which produces an enzyme which converts a precursor molecule into a glycoside and another gene which produces an enzyme which converts the glycoside into hydrogen cyanide (figure A2.2.6(*a*)). If two pure breeding acyanogenic strains are crossed the heterozygote is cyanogenic (figure A2.2.6(*b*)).



(a) Cyanide Production



(b) Cyanogenic hybrid

Figure A2.2.6. Cyanogenic clover: cyanide production and cyanogenic hybrid.

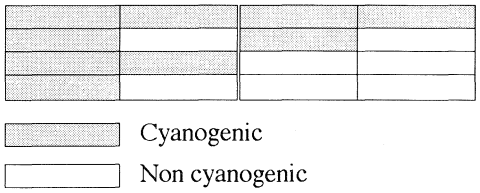
When the cyanogenic strain is selfed the genotypes are as summarized in figure A2.2.7(*a*). There are only two phenotypes produced, cyanogenic and acyanogenic, as summarized in figure A2.2.7(*b*).

So far we have followed Mendel's laws regarding the independent segregation of genes. This independent segregation does not occur when genes are located on the same chromosome. During meiosis homologous chromosomes (i.e. matched pairs one from each parental gamete) move together and are seen to be joined at the centromere (the clear oval region in figure A2.2.8).

In this simplified diagram we show a set of genes (rectangles) in which those on the top are of the opposite form to those on the bottom. As the chromosomes are juxtaposed they each are doubled up so

	LH	Lh	lH	lh
LH	LLHH	LLHh	LlHH	LlHh
Lh	LLHh	LLhh	LlHh	Llhh
lH	LlHH	LlHh	llHH	llHh
lh	LlHh	Llhh	llHh	llhh

(a) Genotype patterns



(b) Phenotype patterns

Figure A2.2.7. Epistasis in clover: genotype and phenotype patterns.

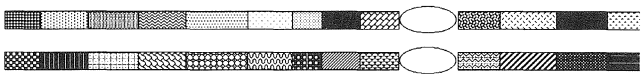


Figure A2.2.8. Pairing of homologous chromosomes.

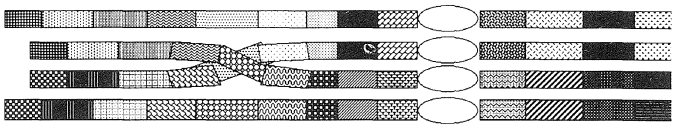


Figure A2.2.9. Crossing over in a tetrad.

that four strands (usually called chromatids) are aligned. The close proximity of the inner two chromatids and the presence of enzymes in the cellular environment can result in breakages and recombinations of these strands as summarized in figure A2.2.9.

The result is that of the four juxtaposed strands two are the same as the parental chromosomes and two, called the recombinants, are different. This crossover process mixes up the genes with respect to original parental chromosomes. The chromosomes which make up a haploid gamete will be a random mixture of parental and recombinant forms. This increases the variability between parents and offspring and reduces the chance of harmful recessives becoming homozygous.

A2.2.3 The gene in more detail

Genes are located on chromosomes. Chromosomes segregate independently during meiosis whereas genes can be linked on the same chromosome. The conceptual reasons why there has been confusion are the differences in understanding about gene and chromosome such as which is the unit of heredity (see Darden 1991). The discovery of the physicochemical nature of hereditary material culminated in the Watson–Crick model in 1953 (see figure A2.2.10). The coding parts of the deoxyribonucleic acid (DNA) are called bases; there are four types (adenine, thymine, cytosine, and guanine). They are strung along a sugar-and-phosphate string, which is arranged as a helix. Two intertwined strings then form the double helix. The functional unit of this code is a triplet of bases which can code for a single amino acid. The genes are located along the DNA strand.

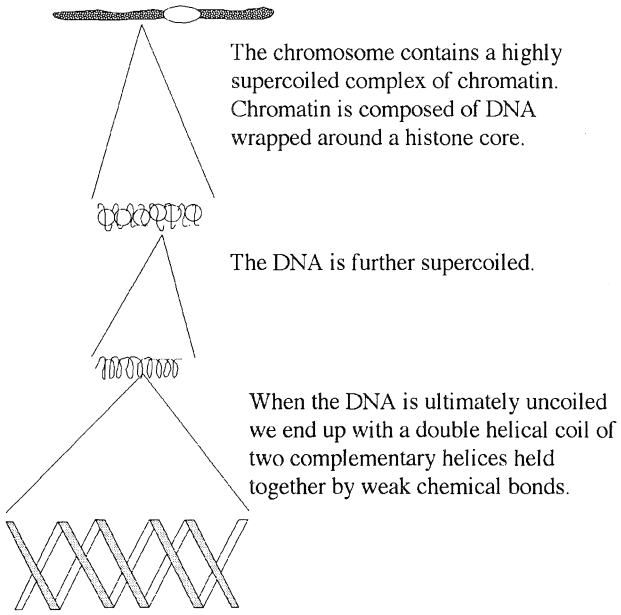


Figure A2.2.10. Idealization of the organization of chromosomes in a eukaryotic cell. (A eukaryotic cell has an organized nucleus and cytoplasmic organelles.)

Transcription is the synthesis of ribonucleic acid (RNA) using the DNA template. It is a preliminary step in the ultimate synthesis of protein. A gene can be transcribed through the action of enzymes and a chain of transcript is formed as a polymer called messenger RNA (mRNA). This mRNA can then be translated into protein. The translation process converts the mRNA code into a protein sequence via another form of RNA called transfer RNA (tRNA). In this way, genes are transcribed so that mRNA may be produced, from which protein molecules (typically the 'workhorses' and structural molecules of a cell) can be formed. This flow of information is generally unidirectional. (For more details on this topic the reader should consult a molecular biology text and look at the central dogma of molecular biology, see e.g. Lewin 1990, Alberts *et al* 1989.)

Figure A2.2.11 provides a simplified view of the anatomy of a structural gene, that is, one which codes for a protein or RNA.

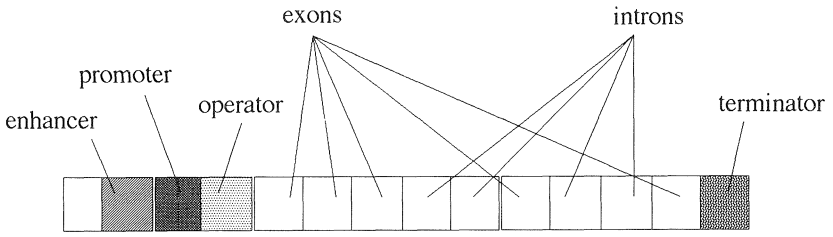


Figure A2.2.11. A simplified diagram of a structural gene.

That part of the gene which ultimately codes for protein or RNA is preceded upstream by three stretches of code. The enhancer facilitates the operation of the promoter region, which is where RNA polymerase is bound to the gene in order to initiate transcription. The operator is the site where transcription can be halted by the presence of a repressor protein. Exons are expressed in the final gene product (e.g. the protein molecule) whereas introns are transcribed but are removed from the transcript leaving the fragments of exon material to be spliced. One stretch of DNA may consist of several overlapping genes. For example, the introns in one gene may be the exons in another (Lewin 1990). The terminator is the postexon region of the gene which causes transcription to be terminated. Thus a biological gene contains not only code to be read but also coded instructions on how it should be read and what should be read. Genes are highly organized. An operon system is located on one chromosome and consists of a regulator gene and a number of contiguous structural genes which share the same promoter and terminator and code for enzymes which are involved in specific metabolic pathways (the classical example is the Lac operon, see figure A2.2.12).

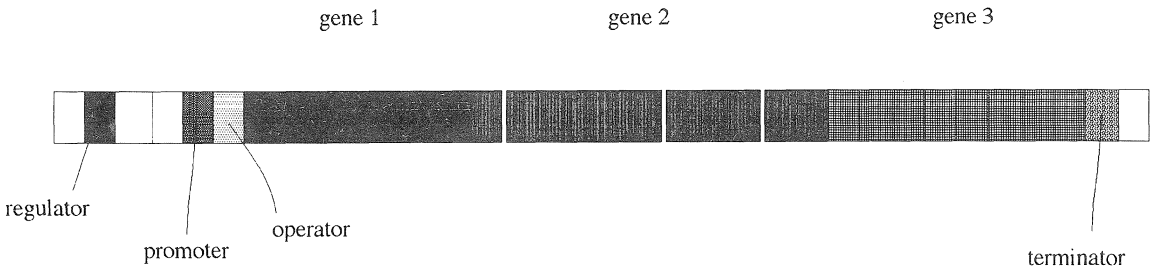


Figure A2.2.12. A visualization of an operon.

Operons can be grouped together into higher-order (hierarchical) regulatory genetic systems (Neidhart *et al* 1990). For example, a number of operons from different chromosomes may be regulated by a single gene known as a regulon. These higher-order systems provide a great challenge for change in a genome. Modification of the higher-order gene can have profound effects on the expression of structural genes that are under its influence.

A2.2.4 Options for change

We have already seen how sexual reproduction can mix up the genes which are incorporated in a gamete through the random reassortment of paternal and maternal chromosomes and through crossing over and recombination. Effectively though, the gamete acquires a subset of the same genes as the diploid gamete-producing cells; they are just mixed up. Clearly, any zygote that is produced will have a mixture of genes and (possibly) some chromosomes which have both paternal and maternal genes.

There are other mechanisms of change which alter the genes themselves or change the number of genes present in a genome. We shall describe a mutation as any change in the sequence of genomic DNA. Gene mutations are of two types: point mutation, in which a single base is changed, and frameshift mutation, in which one or more bases (but not a multiple of three) are inserted or deleted. This changes the frame in which triplets are transcribed into RNA and ultimately translated into protein. In addition some genes are able to become transposed elsewhere in a genome. They 'jump' about and are called transposons. Chromosome changes can be caused by deletion (loss of a section), duplication (the section is repeated), inversion (the section is in the reverse order), and translocation (the section has been relocated elsewhere). There are also changes at the genome level. Ploidy is the term used to describe multiples of a chromosome complement such as haploid (n), diploid ($2n$), and tetraploid ($4n$). A good example of the influence of ploidy on evolution is among such crops as wheat and cotton. Somy describes changes to the frequency of particular chromosomes: for example, trisomy is three copies of a chromosome.

A2.2.5 Population thinking

So far we have focused on how genes are inherited and how they or their combinations can change. In order to understand *evolutionary processes* we must shift our attention to looking at populations (we shall not emphasize too much whether of genes, chromosomes, genomes, or organisms). Population thinking is central to our understanding of models of evolution. A2.1

The Hardy–Weinberg theorem applies to frequencies of genes and genotypes in a population of individuals, and states that the relative frequency of each gene remains in equilibrium from one generation to the next. For a single allele, if the frequency of one form is p then that of the other (say q) is $1 - p$. The three genotypes that exist with this allele have the population proportions of

$$p^2 + 2pq + q^2 = 1.$$

This equation does not apply when a mixture of four factors changes the relative frequencies of genes in a population: mutation, selection, gene flow, and random genetic drift (drift). Drift can be described as the effect of the sampling of a population on its parents. Each generation can be thought of as a sample of its parents' population. In that the current population is a sample of its parents, we acknowledge that a statistical sampling error should be associated with gene frequencies. The effect will be small in large populations because the relative proportion of random changes will be a very small component of the large numbers. However, drift in a small population will have a marked effect.

One factor which can counteract the effect of drift is differential migration of individuals between populations which leads to gene flow. Several models of gene flow exist. For example, migration which occurs at random among a group of small populations is called the *island model* whereas in the *stepping stone* model each population receives migrants only from neighboring populations. Mutation, selection, and gene flow are deterministic factors so that if fitness, mutation rate, and rate of gene flow are the same for a number of populations that begin with same gene frequencies, they will attain the same equilibrium composition. Drift is a stochastic process because the sampling effect on the parent population is random. C6.3

Sewall Wright introduced the idea of an *adaptive landscape* to explain how a population's allele frequencies might evolve over time. The peaks on the landscape represent genetic compositions of a population for which the mean fitness is high and troughs are possible compositions where the mean fitness is low. As gene frequencies change and mean fitness increases the population moves uphill. Indeed, selection will operate to increase mean fitness so, on a multipeaked landscape, selection may operate to move populations to local maxima. On a fixed landscape drift and selection can act together so that populations may move uphill (through selection) or downhill (through drift). This means that the global maximum for the landscape could be reached. These ideas are formally encapsulated in Wright's (1968–1978) *shifting balance theory* of evolution. Further information on the relation of population genetics to B2.7

evolutionary theory can be studied further in the books by Wright (1968–1978), Crow and Kimura (1970) and Maynard Smith (1989).

The change of gene frequencies coupled with changes in the genes themselves can lead to the emergence of new species although the process is far from simple and not fully understood (Futuyma, 1986, Maynard Smith 1993). The nature of the species concept or (for some) concepts which is central to Darwinism is complicated and will not be discussed here (see e.g. Futuyma 1986). Several mechanisms apply to promote speciation (Maynard Smith 1993): geographical or spatial isolation, barriers preventing formation of hybrids, nonviable hybrids, hybrid infertility, and hybrid breakdown—in which post-F1 generations are weak or infertile.

Selectionist theories emphasize invariant properties of the system: the system is an internal generator of variations (Changeux and Dehaene 1989) and diversity among units of the population exists prior to any testing (Manderick 1994). We have seen how section operates to optimize fitness. Darden and Cain (1987) summarize a number of common elements in selectionist theories as follows:

- a set of a given entity type (i.e. the units of the population)
- a particular property (P) according to which members of this set vary
- an environment in which the entity type is found
- a factor in the environment to which members react differentially due to their possession or nonpossession of the property (P)
- differential benefits (both shorter and longer term) according to the possession or nonpossession of the property (P).

This scheme summarizes the selectionist approach. In addition, Maynard Smith (1989) discusses a number of selection systems (particular relevant to animals) including sexual, habitat, family, kin, group, and synergistic (cooperation). A very helpful overview of this area of ecology, behavior, and evolution is that by Sigmund (1993). Three selectionist systems in the biosciences are the neo-Darwinian theory of evolution in a population, clonal selection theory applied to the immune system, and the theory of neuronal group selection (for an excellent summary with plenty of references see that by Manderick (1994)).

There are many important aspects of evolutionary biology which have had to be omitted because of lack of space. The relevance of neutral molecular evolution theory (Kimura 1983) and nonselectionist approaches (see e.g. Goodwin and Saunders 1989, Lima de Faria 1988, Kauffman 1993) has not been discussed. In addition some important ideas have not been considered, such as evolutionary game theory (Maynard Smith 1989, Sigmund 1993), the role of sex (see e.g. Hamilton *et al* 1990), the evolution of cooperation (Axelrod 1984), the red queen (Van Valen 1973, Maynard Smith 1989), structured genomes, for example, incorporation of regulatory hierarchies (Kauffman 1993, Beaumont 1993, Clarke *et al* 1993), experiments with endosymbiotic systems (Margulis and Foster 1991, Hilario and Gogarten 1993), coevolving parasite populations (see e.g. Collins 1994; for a biological critique and further applications see Sumida and Hamilton 1994), inheritance of acquired characteristics (Landman 1991), and genomic imprinting and other epigenetic inheritance systems (for a review see Paton 1994). There are also considerable philosophical issues which must be addressed in this area which impinge on how biological sources are applied to evolutionary computing (see Sober 1984). Not least among these is the nature of adaptation.

References

Alberts B, Bray D, Lewis J, Raff M, Roberts K and Watson J D 1989 *Molecular Biology of the Cell* (New York: Garland)

Axelrod R 1984 *The Evolution of Co-operation* (Harmondsworth: Penguin)

Beaumont M A 1993 Evolution of optimal behaviour in networks of Boolean automata *J. Theor. Biol.* **165** 455–76

Changeux J-P and Dehaene S 1989 Neuronal models of cognitive functions *Cognition* **33** 63-109

Clarke B, Mittenthal J E and Senn M 1993 A model for the evolution of networks of genes *J. Theor. Biol.* **165** 269–89

Collins R 1994 Artificial evolution and the paradox of sex *Computing with Biological Metaphors* ed R C Paton (London: Chapman and Hall)

Crow J F and Kimura M 1970 *An Introduction to Population Genetics Theory* (New York: Harper and Row)

Darden L 1991 *Theory Change in Science* (New York: Oxford University Press)

Darden L and Cain J A 1987 Selection type theories *Phil. Sci.* **56** 106–29

Futuyma D J 1986 *Evolutionary Biology* (MA: Sinauer)

Goodwin B C and Saunders P T (eds) 1989 *Theoretical Biology: Epigenetic and Evolutionary Order from Complex Systems* (Edinburgh: Edinburgh University Press)

Hamilton W D, Axelrod A and Tanese R 1990 Sexual reproduction as an adaptation to resist parasites *Proc. Natl Acad. Sci. USA* **87** 3566–73

Hilario E and Gogarten J P 1993 Horizontal transfer of ATPase genes—the tree of life becomes a net of life *BioSystems* **31** 111–9

Kauffman S A 1993 *The Origins of Order* (New York: Oxford University Press)

Kimura, M 1983 *The Neutral Theory of Molecular Evolution* (Cambridge: Cambridge University Press)

Landman O E 1991 The inheritance of acquired characteristics *Ann. Rev. Genet.* **25** 1–20

Lewin B 1990 *Genes IV* (Oxford: Oxford University Press)

Lima de Faria A 1988 *Evolution without Selection* (Amsterdam: Elsevier)

Margulis L and Foster R (eds) 1991 *Symbiosis as a Source of Evolutionary Innovation: Speciation and Morphogenesis* (Cambridge, MA: MIT Press)

Manderick B 1994 The importance of selectionist systems for cognition *Computing with Biological Metaphors* ed R C Paton (London: Chapman and Hall)

Maynard Smith J 1989 *Evolutionary Genetics* (Oxford: Oxford University Press)

——1993 *The Theory of Evolution* Canto edn (Cambridge: Cambridge University Press)

Neidhart F C, Ingraham J L and Schaechter M 1990 *Physiology of the Bacterial Cell* (Sunderland, MA: Sinauer)

Paton R C 1994 Enhancing evolutionary computation using analogues of biological mechanisms *Evolutionary Computing (Lecture Notes in Computer Science 865)* ed T C Fogarty (Berlin: Springer) pp 51–64

Sigmund K 1993 *Games of Life* (Oxford: Oxford University Press)

Sinnott E W, Dunn L C and Dobzhansky T 1958 *Principles of Genetics* (New York: McGraw-Hill)

Sober E 1984 *The Nature of Selection: Evolutionary Theory in Philosophical Focus* (Chicago, IL: University of Chicago Press)

Sumida B and Hamilton W D 1994 Both Wrightian and 'parasite' peak shifts enhance genetic algorithm performance in the travelling salesman problem *Computing with Biological Metaphors* ed R C Paton (London: Chapman and Hall)

Van Valen L 1973 A new evolutionary law *Evolutionary Theory* **1** 1–30

Wright S 1968–1978 *Evolution and the Genetics of Populations* vols 1–4 (Chicago, IL: Chicago University Press)

A2.3 A history of evolutionary computation

Kenneth De Jong, David B Fogel and Hans-Paul Schwefel

Abstract

This section presents a brief but comprehensive summary of the history of evolutionary computation, starting with the ground-breaking work of the 1950s, tracing the rise and development of the three primary forms of evolutionary algorithms (evolution strategies, evolutionary programming and genetic algorithms), and concluding with the present time, in which a more unified view of the field is emerging.

A2.3.1 Introduction

No one will ever produce a completely accurate account of a set of past events since, as someone once pointed out, writing history is as difficult as forecasting. Thus we dare to begin our historical summary of evolutionary computation rather arbitrarily at a stage as recent as the mid-1950s.

At that time there was already evidence of the use of digital computer models to better understand the natural process of evolution. One of the first descriptions of the use of an evolutionary process for computer problem solving appeared in the articles by Friedberg (1958) and Friedberg *et al* (1959). This represented some of the early work in machine learning and described the use of an evolutionary algorithm for *automatic programming*, i.e. the task of finding a program that calculates a given input–output function. Other founders in the field remember a paper of Fraser (1957) that influenced their early work, and there may be many more such forerunners depending on whom one asks.

In the same time frame Bremermann presented some of the first attempts to apply simulated evolution to numerical optimization problems involving both linear and convex optimization as well as the solution of nonlinear simultaneous equations (Bremermann 1962). Bremermann also developed some of the early evolutionary algorithm (EA) theory, showing that the optimal mutation probability for linearly separable problems should have the value of $1/\ell$ in the case of ℓ bits encoding an individual (Bremermann *et al* 1965).

Also during this period Box developed his *evolutionary operation* (EVOP) ideas which involved an evolutionary technique for the design and analysis of (industrial) experiments (Box 1957, Box and Draper 1969). Box's ideas were never realized as a computer algorithm, although Spendley *et al* (1962) used them as the basis for their so-called *simplex design* method. It is interesting to note that the REVOP proposal (Satterthwaite 1959a, b) introducing randomness into the EVOP operations was rejected at that time.

As is the case with many ground-breaking efforts, these early studies were met with considerable skepticism. However, by the mid-1960s the bases for what we today identify as the three main forms of EA were clearly established. The roots of *evolutionary programming* (EP) were laid by Lawrence Fogel B1.4 in San Diego, California (Fogel *et al* 1966) and those of *genetic algorithms* (GAs) were developed at B1.2 the University of Michigan in Ann Arbor by Holland (1967). On the other side of the Atlantic Ocean, *evolution strategies* (ESs) were a joint development of a group of three students, Bienert, Rechenberg, and B1.3 Schwefel, in Berlin (Rechenberg 1965).

Over the next 25 years each of these branches developed quite independently of each other, resulting in unique parallel histories which are described in more detail in the following sections. However, in 1990 there was an organized effort to provide a forum for interaction among the various EA research

communities. This took the form of an international workshop entitled *Parallel Problem Solving from Nature* at Dortmund (Schwefel and Männer 1991).

Since that event the interaction and cooperation among EA researchers from around the world has continued to grow. In the subsequent years special efforts were made by the organizers of *ICGA'91* (Belew and Booker 1991), *EP'92* (Fogel and Atmar 1992), and *PPSN'92* (Männer and Manderick 1992) to provide additional opportunities for interaction.

This increased interaction led to a consensus for the name of this new field, *evolutionary computation* (EC), and the establishment in 1993 of a journal by the same name published by MIT Press. The increasing interest in EC was further indicated by the *IEEE World Congress on Computational Intelligence (WCCI)* at Orlando, Florida, in June 1994 (Michalewicz *et al* 1994), in which one of the three simultaneous conferences was dedicated to EC along with conferences on neural networks and fuzzy systems. The dramatic growth of interest provided additional evidence for the need of an organized EC handbook (which you are now reading) to provide a more cohesive view of the field.

That brings us to the present in which the continued growth of the field is reflected by the many EC events and related activities each year, and its growing maturity reflected by the increasing number of books and articles about EC.

In order to keep this overview brief, we have deliberately suppressed many of the details of the historical developments within each of the three main EC streams. For the interested reader these details are presented in the following sections.

A2.3.2 Evolutionary programming

Evolutionary programming (EP) was devised by Lawrence J Fogel in 1960 while serving at the National Science Foundation (NSF). Fogel was on leave from Convair, tasked as special assistant to the associate director (research), Dr Richard Bolt, to study and write a report on investing in basic research. Artificial intelligence at the time was mainly concentrated around heuristics and the simulation of primitive neural networks. It was clear to Fogel that both these approaches were limited because they model humans rather than the essential process that produces creatures of increasing intellect: evolution. Fogel considered intelligence to be based on adapting behavior to meet goals in a range of environments. In turn, prediction was viewed as the key ingredient to intelligent behavior and this suggested a series of experiments on the
C1.5 use of simulated evolution of *finite-state machines* to forecast nonstationary time series with respect to arbitrary criteria. These and other experiments were documented in a series of publications (Fogel 1962, 1964, Fogel *et al* 1965, 1966, and many others).

Intelligent behavior was viewed as requiring the composite ability to (i) predict one's environment, coupled with (ii) a translation of the predictions into a suitable response in light of the given goal. For the sake of generality, the environment was described as a sequence of symbols taken from a finite alphabet. The evolutionary problem was defined as evolving an algorithm (essentially a program) that would operate on the sequence of symbols thus far observed in such a manner so as to produce an output symbol that is likely to maximize the algorithm's performance in light of both the next symbol to appear in the environment and a well-defined payoff function. Finite-state machines provided a useful representation for the required behavior.

The proposal was as follows. A population of finite-state machines is exposed to the environment, that is, the sequence of symbols that have been observed up to the current time. For each parent machine, as each input symbol is offered to the machine, each output symbol is compared with the next input symbol. The worth of this prediction is then measured with respect to the payoff function (e.g. all–none, absolute error, squared error, or any other expression of the meaning of the symbols). After the last prediction is made, a function of the payoff for each symbol (e.g. average payoff per symbol) indicates the fitness of the machine.

Offspring machines are created by randomly mutating each parent machine. Each parent produces offspring (this was originally implemented as only a single offspring simply for convenience). There are five possible modes of random mutation that naturally result from the description of the machine: change an output symbol, change a state transition, add a state, delete a state, or change the initial state. The deletion of a state and change of the initial state are only allowed when the parent machine has more than one state. Mutations are chosen with respect to a probability distribution, which is typically uniform. The number of mutations per offspring is also chosen with respect to a probability distribution or may be fixed *a priori*. These offspring are then evaluated over the existing environment in the same manner as

their parents. Other mutations, such as majority logic mating operating on three or more machines, were proposed by Fogel *et al* (1966) but not implemented.

The machines that provide the greatest payoff are retained to become parents of the next generation. (Typically, half the total machines were saved so that the parent population remained at a constant size.) This process is iterated until an actual prediction of the next symbol (as yet unexperienced) in the environment is required. The best machine generates this prediction, the new symbol is added to the experienced environment, and the process is repeated. Fogel (1964) (and Fogel *et al* (1966)) used 'nonregressive' evolution. To be retained, a machine had to rank in the best half of the population. Saving lesser-adapted machines was discussed as a possibility (Fogel *et al* 1966, p 21) but not incorporated.

This general procedure was successfully applied to problems in prediction, identification, and automatic control (Fogel *et al* 1964, 1966, Fogel 1968) and was extended to simulate coevolving populations by Fogel and Burgin (1969). Additional experiments evolving finite-state machines for sequence prediction, pattern recognition, and gaming can be found in the work of Lutter and Huntsinger (1969), Burgin (1969), Atmar (1976), Dearholt (1976), and Takeuchi (1980).

In the mid-1980s the general EP procedure was extended to alternative representations including ordered lists for the traveling salesman problem (Fogel and Fogel 1986), and real-valued vectors for continuous function optimization (Fogel and Fogel 1986). This led to other applications in route planning (Fogel 1988, Fogel and Fogel 1988), optimal subset selection (Fogel 1989), and training neural networks (Fogel *et al* 1990), as well as comparisons to other methods of *simulated evolution* (Fogel and Atmar 1990). F1.10
Methods for extending evolutionary search to a two-step process including evolution of the mutation variance were offered by Fogel *et al* (1991, 1992). Just as the proper choice of step sizes is a crucial part of every numerical process, including optimization, the internal adaptation of the mutation variance(s) is of utmost importance for the algorithm's efficiency. This process is called *self-adaptation* or autoadaptation C7.1
in the case of no explicit control mechanism, e.g. if the variances are part of the individuals' characteristics and underlie probabilistic variation in a similar way as do the ordinary decision variables.

In the early 1990s efforts were made to organize annual conferences on EP, these leading to the first conference in 1992 (Fogel and Atmar 1992). This conference offered a variety of optimization applications of EP in robotics (McDonnell *et al* 1992, Andersen *et al* 1992), path planning (Larsen and Herman 1992, Page *et al* 1992), *neural network design and training* (Sebald and Fogel 1992, Porto 1992, D1
McDonnell 1992), *automatic control* (Sebald *et al* 1992), and other fields. F1.3

First contacts were made between the EP and ES communities just before this conference, and the similar but independent paths that these two approaches had taken to simulating the process of evolution were clearly apparent. Members of the ES community have participated in all successive EP conferences (Bäck *et al* 1993, Sprave 1994, Bäck and Schütz 1995, Fogel *et al* 1996). There is less similarity between EP and GAs, as the latter emphasize simulating specific mechanisms that apply to natural genetic systems whereas EP emphasizes the behavioral, rather than genetic, relationships between parents and their offspring. Members of the GA and GP communities have, however, also been invited to participate in the annual conferences, making for truly interdisciplinary interaction (see e.g. Altenberg 1994, Land and Belew 1995, Koza and Andre 1996).

Since the early 1990s, efforts in EP have diversified in many directions. Applications in training neural networks have received considerable attention (see e.g. English 1994, Angeline *et al* 1994, McDonnell and Waagen 1994, Porto *et al* 1995), while relatively less attention has been devoted to evolving *fuzzy* D2
systems (Haffner and Sebald 1993, Kim and Jeon 1996). Image processing applications can be found in the articles by Bhattacharjya and Roysam (1994), Brotherton *et al* (1994), Rizki *et al* (1995), and others. Recent efforts to use EP in medicine have been offered by Fogel *et al* (1995) and Gehlhaar *et al* (1995). Efforts studying and comparing methods of self-adaptation can be found in the articles by Saravanan *et al* (1995), Angeline *et al* (1996), and others. Mathematical analyses of EP have been summarized by Fogel (1995).

To offer a summary, the initial efforts of L J Fogel indicate some of the early attempts to (i) use simulated evolution to perform prediction, (ii) include variable-length encodings, (iii) use representations that take the form of a sequence of instructions, (iv) incorporate a population of candidate solutions, and (v) coevolve evolutionary programs. Moreover, Fogel (1963, 1964) and Fogel *et al* (1966) offered the early recognition that natural evolution and the human endeavor of the scientific method are essentially similar processes, a notion recently echoed by Gell-Mann (1994). The initial prescriptions for operating on finite-state machines have been extended to arbitrary representations, mutation operators, and selection methods, and techniques for self-adapting the evolutionary search have been proposed and implemented.

The population size need not be kept constant and there can be a variable number of offspring per parent, much like the *($\mu + \lambda$) methods* offered in ESs. In contrast to these methods, selection is often C2 made probabilistic in EP, giving lesser-scoring solutions some probability of surviving as parents into the next generation. In contrast to GAs, no effort is made in EP to support (some say maximize) *schema* B2 *processing*, nor is the use of random variation constrained to emphasize specific mechanisms of genetic transfer, perhaps providing greater versatility to tackle specific problem domains that are unsuitable for genetic operators such as crossover.

A2.3.3 Genetic algorithms

The first glimpses of the ideas underlying genetic algorithms (GAs) are found in Holland's papers in the early 1960s (see e.g. Holland 1962). In them Holland set out a broad and ambitious agenda for understanding the underlying principles of adaptive systems—systems that are capable of self-modification in response to their interactions with the environments in which they must function. Such a theory of adaptive systems should facilitate both the understanding of complex forms of adaptation as they appear in natural systems and our ability to design robust adaptive artifacts.

In Holland's view the key feature of robust natural adaptive systems was the successful use of competition and innovation to provide the ability to dynamically respond to unanticipated events and changing environments. Simple models of biological evolution were seen to capture these ideas nicely via notions of survival of the fittest and the continuous production of new offspring.

This theme of using evolutionary models both to understand natural adaptive systems and to design robust adaptive artifacts gave Holland's work a somewhat different focus than those of other contemporary groups that were exploring the use of evolutionary models in the design of efficient experimental optimization techniques (Rechenberg 1965) or for the evolution of intelligent agents (Fogel *et al* 1966), as reported in the previous section.

By the mid-1960s Holland's ideas began to take on various computational forms as reflected by the PhD students working with Holland. From the outset these systems had a distinct 'genetic' flavor to them in the sense that the objects to be evolved over time were represented internally as 'genomes' and the mechanisms of reproduction and inheritance were simple abstractions of familiar population genetics operators such as mutation, crossover, and inversion.

Bagley's thesis (Bagley 1967) involved tuning sets of weights used in the evaluation functions of game-playing programs, and represents some of the earliest experimental work in the use of diploid representations, the role of inversion, and selection mechanisms. By contrast Rosenberg's thesis (Rosenberg 1967) has a very distinct flavor of simulating the evolution of a simple biochemical system in which single-celled organisms capable of producing enzymes were represented in diploid fashion and were evolved over time to produce appropriate chemical concentrations. Of interest here is some of the earliest experimentation with adaptive crossover operators.

Cavicchio's thesis (Cavicchio 1970) focused on viewing these ideas as a form of adaptive search, and tested them experimentally on difficult search problems involving subroutine selection and pattern C2.7.4 recognition. In his work we see some of the early studies on *elitist* forms of selection and ideas for adapting the rates of crossover and mutation. Hollstien's thesis (Hollstien 1971) took the first detailed B2.7.4 look at alternate selection and mating schemes. Using a test suite of two-dimensional *fitness landscapes*, he experimented with a variety of breeding strategies drawn from techniques used by animal breeders. Also of interest here is Hollstien's use of binary string encodings of the genome and early observations about the virtues of Gray codings.

In parallel with these experimental studies, Holland continued to work on a general theory of adaptive B2.5 systems (Holland 1967). During this period he developed his now famous *schema analysis* of adaptive systems, relating it to the optimal allocation of trials using k-armed bandit models (Holland 1969). He used these ideas to develop a more theoretical analysis of his *reproductive plans* (simple GAs) (Holland 1971, 1973). Holland then pulled all of these ideas together in his pivotal book *Adaptation in Natural and Artificial Systems* (Holland 1975).

Of interest was the fact that many of the desirable properties of these algorithms being identified by Holland theoretically were frequently not observed experimentally. It was not difficult to identify the reasons for this. Hampered by a lack of computational resources and analysis tools, most of the early experimental studies involved a relatively small number of runs using small population sizes (generally

less than 20). It became increasingly clear that many of the observed deviations from expected behavior could be traced to the well-known phenomenon in population genetics of *genetic drift,* the loss of genetic diversity due to the stochastic aspects of selection, reproduction, and the like in small populations.

By the early 1970s there was considerable interest in understanding better the behavior of implementable GAs. In particular, it was clear that choices of population size, representation issues, the choice of operators and operator rates all had significant effects of the observed behavior of GAs. Frantz's thesis (Frantz 1972) reflected this new focus by studying in detail the roles of crossover and inversion in populations of size 100. Of interest here is some of the earliest experimental work on multipoint crossover operators.

De Jong's thesis (De Jong 1975) broaded this line of study by analyzing both theoretically and experimentally the interacting effects of population size, crossover, and mutation on the behavior of a family of GAs being used to optimize a fixed test suite of functions. Out of this study came a strong sense that even these simple GAs had significant potential for solving difficult optimization problems.

The mid-1970s also represented a branching out of the family tree of GAs as other universities and research laboratories established research activities in this area. This happened slowly at first since initial attempts to spread the word about the progress being made in GAs were met with fairly negative perceptions from the artificial intelligence (AI) community as a result of early overhyped work in areas such as self-organizing systems and perceptrons.

Undaunted, groups from several universities including the University of Michigan, the University of Pittsburgh, and the University of Alberta organized an *Adaptive Systems Workshop* in the summer of 1976 in Ann Arbor, Michigan. About 20 people attended and agreed to meet again the following summer. This pattern repeated itself for several years, but by 1979 the organizers felt the need to broaden the scope and make things a little more formal. Holland, De Jong, and Sampson obtained NSF funding for *An Interdisciplinary Workshop in Adaptive Systems,* which was held at the University of Michigan in the summer of 1981 (Sampson 1981).

By this time there were several established research groups working on GAs. At the University of Michigan, Bethke, Goldberg, and Booker were continuing to develop GAs and explore Holland's *classifier systems* as part of their PhD research (Bethke 1981, Booker 1982, Goldberg 1983). At the University of Pittsburgh, Smith and Wetzel were working with De Jong on various GA enhancements including the *Pitt approach* to rule learning (Smith 1980, Wetzel 1983). At the University of Alberta, Brindle continued to look at optimization applications of GAs under the direction of Sampson (Brindle 1981). B1.5.2

The continued growth of interest in GAs led to a series of discussions and plans to hold the first *International Conference on Genetic Algorithms (ICGA)* in Pittsburgh, Pennsylvania, in 1985. There were about 75 participants presenting and discussing a wide range of new developments in both the theory and application of GAs (Grefenstette 1985). The overwhelming success of this meeting resulted in agreement to continue *ICGA* as a biannual conference. Also agreed upon at *ICGA'85* was the initiation of a moderated electronic discussion group called *GA List*.

The field continued to grow and mature as reflected by the *ICGA* conference activities (Grefenstette 1987, Schaffer 1989) and the appearance of several books on the subject (Davis 1987, Goldberg 1989). Goldberg's book, in particular, served as a significant catalyst by presenting current GA theory and applications in a clear and precise form easily understood by a broad audience of scientists and engineers.

By 1989 the *ICGA* conference and other GA-related activities had grown to a point that some more formal mechanisms were needed. The result was the formation of the International Society for Genetic Algorithms (ISGA), an incorporated body whose purpose is to serve as a vehicle for conference funding and to help coordinate and facilitate GA-related activities. One of its first acts of business was to support a proposal to hold a theory workshop on the *Foundations of Genetic Algorithms (FOGA)* in Bloomington, Indiana (Rawlins 1991).

By this time nonstandard GAs were being developed to evolve complex, nonlinear variable-length structures such as rule sets, LISP code, and neural networks. One of the motivations for *FOGA* was the sense that the growth of GA-based applications had driven the field well beyond the capacity of existing theory to provide effective analyses and predictions.

Also in 1990, Schwefel hosted the first *PPSN* conference in Dortmund, which resulted in the first organized interaction between the ES and GA communities. This led to additional interaction at *ICGA'91* in San Diego which resulted in an informal agreement to hold *ICGA* and *PPSN* in alternating years, and a commitment to jointly initiate a journal for the field.

It was felt that in order for the journal to be successful, it must have broad scope and include other species of EA. Efforts were made to include the EP community as well (which began to organize its own conferences in 1992), and the new journal *Evolutionary Computation* was born with the inaugural issue in the spring of 1993.

The period from 1990 to the present has been characterized by tremendous growth and diversity of the GA community as reflected by the many conference activities (e.g. *ICGA* and *FOGA*), the emergence
C4.2.4 of new books on GAs, and a growing list of journal papers. New paradigms such as *messy GAs* (Goldberg
B1.5.1 *et al* 1991) and *genetic programming* (Koza 1992) were being developed. The interactions with other EC communities resulted in considerable crossbreeding of ideas and many new hybrid EAs. New GA applications continue to be developed, spanning a wide range of problem areas from engineering design problems to operations research problems to automatic programming.

A2.3.4 Evolution strategies

In 1964, three students of the Technical University of Berlin, Bienert, Rechenberg, and Schwefel, did not at all aim at devising a new kind of optimization procedure. During their studies of aerotechnology and space technology they met at an Institute of Fluid Mechanics and wanted to construct a kind of research robot that should perform series of experiments on a flexible slender three-dimensional body in a wind tunnel so as to minimize its drag. The method of minimization was planned to be either a one variable at a time or a discrete gradient technique, gleaned from classical numerics. Both strategies, performed manually, failed, however. They became stuck prematurely when used for a two-dimensional demonstration facility, a joint plate—its optimal shape being a flat plate—with which the students tried to demonstrate that it was possible to find the optimum automatically.

Only then did Rechenberg (1965) hit upon the idea to use dice for random decisions. This was the breakthrough—on 12 June 1964. The first version of an evolutionary strategy (ES), later called the $(1 + 1)$ ES, was born, with discrete, binomially distributed mutations centered at the ancestor's position, and just one parent and one descendant per generation. This ES was first tested on a mechanical calculating machine by Schwefel before it was used for the *experimentum crucis*, the joint plate. Even then, it took a while to overcome a merely locally optimal S shape and to converge towards the expected global optimum, the flat plate. Bienert (1967), the third of the three students, later actually constructed a kind of robot that could perform the actions and decisions automatically.

Using this simple *two-membered* ES, another student, Lichtfuß (1965), optimized the shape of a bent pipe, also experimentally. The result was rather unexpected, but nevertheless obviously better than all shapes proposed so far.

First computer experiments, on a Zuse Z23, as well as analytical investigations using binomially distributed integer mutations, had already been performed by Schwefel (1965). The main result was that such a strategy can become stuck prematurely, i.e. at 'solutions' that are not even locally optimal. Based on this experience the use of normally instead of binomially distributed mutations became standard in most of the later computer experiments with real-valued variables and in theoretical investigations into the method's efficiency, but not however in experimental optimization using ESs. In 1966 the little ES community was destroyed by dismissal from the Institute of Fluid Mechanics ('Cybernetics as such is no longer pursued at the institute!'). Not before 1970 was it found together again at the Institute of Measurement and Control of the Technical University of Berlin, sponsored by grants from the German Research Foundation (DFG). Due to the circumstances, the group missed publishing its ideas and results properly, especially in English.

In the meantime the often-cited two-phase nozzle optimization was performed at the Institute of Nuclear Technology of the Technical University of Berlin, then in an industrial surrounding, the AEG research laboratory (Schwefel 1968, Klockgether and Schwefel 1970), also at Berlin. For a hot-water flashing flow the shape of a three-dimensional convergent–divergent (thus supersonic) nozzle with maximum energy efficiency was sought. Though in this experimental optimization an exogenously controlled binomial-like distribution was used again, it was the first time that gene duplication and deletion were incorporated into an EA, especially in a $(1 + 1)$ ES, because the optimal length of the nozzle was not known in advance. As in case of the bent pipe this experimental strategy led to highly unexpected results, not easy to understand even afterwards, but definitely much better than available before.

First Rechenberg and later Schwefel analyzed and improved their ES. For the $(1+1)$ ES, Rechenberg, in his Dr.-Ing. thesis of 1971, developed, on the basis of two convex n-dimensional model functions, a

convergence rate theory for $n \gg 1$ variables. Based on these results he formulated a $\frac{1}{5}$ success rule for adapting the standard deviation of mutation (Rechenberg 1973). The hope of arriving at an even better strategy by imitating organic evolution more closely led to the incorporation of the population principle and the introduction of recombination, which of course could not be embedded in the $(1 + 1)$ ES. A first *multimembered* ES, the $(\mu + 1)$ ES—the notation was introduced later by Schwefel—was also designed by Rechenberg in his seminal work of 1973. Because of its inability to self-adapt the mutation step sizes (more accurately, standard deviations of the mutations), this strategy was never widely used.

Much more widespread became the $(\mu+\lambda)$ ES and (μ, λ) ES, both formulated by Schwefel in his Dr.-Ing. thesis of 1974–1975. It contains theoretical results such as a convergence rate theory for the $(1+\lambda)$ ES and the $(1, \lambda)$ ES ($\lambda > 1$), analogous to the theory introduced by Rechenberg for the $(1+1)$ ES (Schwefel 1977). The *multimembered* ($\mu > 1$) ESs arose from the otherwise ineffective incorporation of mutatable mutation parameters (variances and covariances of the Gaussian distributions used). Self-adaptation was achieved with the (μ, λ) ES first, not only with respect to the step sizes, but also with respect to correlation coefficients. The enhanced ES version with correlated mutations, described already in an internal report (Schwefel 1974), was published much later (Schwefel 1981) due to the fact that the author left Berlin in 1976. A more detailed empirical analysis of the on-line self-adaptation of the internal or strategy parameters was first published by Schwefel in 1987 (the tests themselves were secretly performed on one of the first small instruction multiple data (SIMD) parallel machines (CRAY1) at the Nuclear Research Centre (KFA) Jülich during the early 1980s with a first parallel version of the multimembered ES with correlated mutations). It was in this work that the notion of *self-adaptation by collective learning* first came up. The importance of recombination (for object as well as strategy parameters) and soft selection (or $\mu > 1$) was clearly demonstrated. Only recently has Beyer (1995a, b) delivered the theoretical background to that particularly important issue.

It may be worth mentioning that in the beginning there were strong objections against increasing λ as well as μ beyond one. The argument against $\lambda > 1$ was that the exploitation of the current knowledge was unnecessarily delayed, and the argument against $\mu > 1$ was that the survival of inferior members of the population would unnecessarily slow down the evolutionary progress. The hint that λ successors could be evaluated in parallel did not convince anybody since parallel computers were neither available nor expected in the near future. The two-membered ES and the very similar creeping random search method of Rastrigin (1965) were investigated thoroughly with respect to their convergence and convergence rates also by Matyas (1965) in Czechoslovakia, Born (1978) on the Eastern side of the Berlin wall (!), and Rappl (1984) in Munich.

Since this early work many new results have been produced by the ES community consisting of the group at Berlin (Rechenberg, since 1972) and that at Dortmund (Schwefel, since 1985). In particular, strategy variants concerning other than only real-valued parameter optimization, i.e. real-world problems, were invented. The first use of an ES for binary optimization using multicellular individuals was presented by Schwefel (1975). The idea of using several subpopulations and *niching mechanisms* for global optimization was propagated by Schwefel in 1977; due to a lack of computing resources, however, it could not be tested thoroughly at that time. Rechenberg (1978) invented a notational scheme for such nested ESs. C6.1

Beside these nonstandard approaches there now exists a wide range of other ESs, e.g. several parallel concepts (Hoffmeister and Schwefel 1990, Lohmann 1991, Rudolph 1991, 1992, Sprave 1994, Rudolph and Sprave 1995), ESs for *multicriterion problems* (Kursawe 1991, 1992), for mixed-integer tasks (Lohmann 1992, Rudolph 1994, Bäck and Schütz 1995), and even for problems with a variable-dimensional parameter space (Schütz and Sprave 1996), and variants concerning nonstandard step size and direction adaptation schemes (see e.g. Matyas 1967, Stewart *et al* 1967, Fürst *et al* 1968, Heydt 1970, Rappl 1984, Ostermeier *et al* 1994). Comparisons between ESs, GAs, and EP may be found in the articles by Bäck *et al* (1991, 1993). It was Bäck (1996) who introduced a common algorithmic scheme for all brands of current EAs. F1.9

Omitting all these other useful nonstandard ESs—a commented collection of literature concerning ES applications was made at the University of Dortmund (Bäck *et al* 1992)—the history of ESs is closed with a mention of three recent books by Rechenberg (1994), Schwefel (1995), and Bäck (1996) as well as three recent contributions that may be seen as written tutorials (Schwefel and Rudolph 1995, Bäck and Schwefel 1995, Schwefel and Bäck 1995), which on the one hand define the actual standard ES algorithms and on the other hand present some recent theoretical results.

References

Altenberg L 1994 Emergent phenomena in genetic programming *Proc. 3rd Annu. Conf. on Evolutionary Programming (San Diego, CA, 1994)* ed A V Sebald and L J Fogel (Singapore: World Scientific) pp 233–41

Andersen B, McDonnell J and Page W 1992 Configuration optimization of mobile manipulators with equality constraints using evolutionary programming *Proc. 1st Ann. Conf. on Evolutionary Programming (La Jolla, CA, 1992)* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 71–9

Angeline P J, Fogel D B and Fogel L J 1996 A comparison of self-adaptation methods for finite state machines in a dynamic environment *Evolutionary Programming V—Proc. 5th Ann. Conf. on Evolutionary Programming (1996)* ed L J Fogel, P J Angeline and T Bäck (Cambridge, MA: MIT Press)

Angeline P J, Saunders G M and Pollack J B 1994 An evolutionary algorithm that constructs recurrent neural networks *IEEE Trans. Neural Networks* **NN-5** 54–65

Atmar J W 1976 *Speculation of the Evolution of Intelligence and Its Possible Realization in Machine Form* ScD Thesis, New Mexico State University

Bäck T 1996 *Evolutionary Algorithms in Theory and Practice* (New York: Oxford University Press)

Bäck T, Hoffmeister F and Schwefel H-P 1991 A survey of evolution strategies *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 2–9

——1992 *Applications of Evolutionary Algorithms* Technical Report of the University of Dortmund Department of Computer Science Systems Analysis Research Group SYS-2/92

Bäck T, Rudolph G and Schwefel H-P 1993 Evolutionary programming and evolution strategies: similarities and differences *Proc. 2nd Ann. Conf. on Evolutionary Programming (San Diego, CA, 1993)* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 11–22

Bäck T and Schütz M 1995 Evolution strategies for mixed-integer optimization of optical multilayer systems *Evolutionary Programming IV—Proc. 4th Ann. Conf on Evolutionary Programming (San Diego, CA, 1995)* ed J R McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press) pp 33–51

Bäck T and Schwefel H-P 1995 Evolution strategies I: variants and their computational implementation *Genetic Algorithms in Engineering and Computer Science, Proc. 1st Short Course EUROGEN-95* ed G Winter, J Périaux, M Galán and P Cuesta (New York: Wiley) pp 111–26

Bagley J D 1967 *The Behavior of Adaptive Systems which Employ Genetic and Correlation Algorithms* PhD Thesis, University of Michigan

Belew R K and Booker L B (eds) 1991 *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, 1991)* (San Mateo, CA: Morgan Kaufmann)

Bethke A D 1981 *Genetic Algorithms as Function Optimizers* PhD Thesis, University of Michigan

Beyer H-G 1995a *How GAs do Not Work—Understanding GAs Without Schemata and Building Blocks* Technical Report of the University of Dortmund Department of Computer Science Systems Analysis Research Group SYS-2/95

——1995b Toward a theory of evolution strategies: on the benefit of sex—the $(\mu/\mu, \lambda)$-theory *Evolutionary Comput.* **3** 81–111

Bhattacharjya A K and Roysam B 1994 Joint solution of low-, intermediate- and high-level vision tasks by evolutionary optimization: application to computer vision at low SNR *IEEE Trans. Neural Networks* **NN-5** 83–95

Bienert P 1967 *Aufbau einer Optimierungsautomatik für drei Parameter* Dipl.-Ing. Thesis, Technical University of Berlin, Institute of Measurement and Control Technology

Booker L 1982 *Intelligent Behavior as an Adaptation to the Task Environment* PhD Thesis, University of Michigan

Born J 1978 *Evolutionsstrategien zur numerischen Lösung von Adaptationsaufgaben* PhD Thesis, Humboldt University at Berlin

Box G E P 1957 Evolutionary operation: a method for increasing industrial productivity *Appl. Stat.* **6** 81–101

Box G E P and Draper N P 1969 *Evolutionary Operation. A Method for Increasing Industrial Productivity* (New York: Wiley)

Bremermann H J 1962 Optimization through evolution and recombination *Self-Organizing Systems* ed M C Yovits *et al* (Washington, DC: Spartan)

Bremermann H J, Rogson M and Salaff S 1965 Search by evolution *Biophysics and Cybernetic Systems—Proc. 2nd Cybernetic Sciences Symp.* ed M Maxfield, A Callahan and L J Fogel (Washington, DC: Spartan) pp 157–67

Brindle A 1981 *Genetic Algorithms for Function Optimization* PhD Thesis, University of Alberta

Brotherton T W, Simpson P K, Fogel D B and Pollard T 1994 Classifier design using evolutionary programming *Proc. 3rd Ann. Conf. on Evolutionary Programming (San Diego, CA, 1994)* ed A V Sebald and L J Fogel (Singapore: World Scientific) pp 68–75

Burgin G H 1969 On playing two-person zero-sum games against nonminimax players *IEEE Trans. Syst. Sci. Cybernet.* **SSC-5** 369–70

Cavicchio D J 1970 *Adaptive Search Using Simulated Evolution* PhD Thesis, University of Michigan

Davis L 1987 *Genetic Algorithms and Simulated Annealing* (London: Pitman)

Dearholt D W 1976 Some experiments on generalization using evolving automata *Proc. 9th Int. Conf. on System Sciences (Honolulu, HI)* pp 131–3

De Jong K A 1975 *Analysis of Behavior of a Class of Genetic Adaptive Systems* PhD Thesis, University of Michigan

English T M 1994 Generalization in populations of recurrent neural networks *Proc. 3rd Ann. Conf. on Evolutionary Programming (San Diego, CA, 1994)* ed A V Sebald and L J Fogel (Singapore: World Scientific) pp 26–33

Fogel D B 1988 An evolutionary approach to the traveling salesman problem *Biol. Cybernet.* **60** 139–44

——1989 Evolutionary programming for voice feature analysis *Proc. 23rd Asilomar Conf. on Signals, Systems and Computers (Pacific Grove, CA)* pp 381–3

——1995 *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence* (New York: IEEE)

Fogel D B and Atmar J W 1990 Comparing genetic operators with Gaussian mutations in simulated evolutionary processing using linear systems *Biol. Cybernet.* **63** 111–4

——(eds) 1992 *Proc. 1st Ann. Conf. on Evolutionary Programming (La Jolla, CA, 1992)* (La Jolla, CA: Evolutionary Programming Society)

Fogel D B and Fogel L J 1988 Route optimization through evolutionary programming *Proc. 22nd Asilomar Conf. on Signals, Systems and Computers (Pacific Grove, CA)* pp 679–80

Fogel D B, Fogel L J and Atmar J W 1991 Meta-evolutionary programming *Proc. 25th Asilomar Conf. on Signals, Systems and Computers (Pacific Grove, CA)* ed R R Chen pp 540–5

Fogel D B, Fogel L J, Atmar J W and Fogel G B 1992 Hierarchic methods of evolutionary programming *Proc. 1st Ann. Conf. on Evolutionary Programming (La Jolla, CA, 1992)* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 175–82

Fogel D B, Fogel L J and Porto V W 1990 Evolving neural networks *Biol. Cybernet.* **63** 487–93

Fogel D B, Wasson E C and Boughton E M 1995 Evolving neural networks for detecting breast cancer *Cancer Lett.* **96** 49–53

Fogel L J 1962 Autonomous automata *Industrial Res.* **4** 14–9

——1963 *Biotechnology: Concepts and Applications* (Englewood Cliffs, NJ: Prentice-Hall)

——1964 *On the Organization of Intellect* PhD Thesis, University of California at Los Angeles

——1968 Extending communication and control through simulated evolution *Bioengineering—an Engineering View Proc. Symp. on Engineering Significance of the Biological Sciences* ed G Bugliarello (San Francisco, CA: San Francisco Press) pp 286–304

Fogel L J, Angeline P J and Bäck T (eds) 1996 *Evolutionary Programming V—Proc. 5th Ann. Conf. on Evolutionary Programming (1996)* (Cambridge, MA: MIT Press)

Fogel L J and Burgin G H 1969 *Competitive Goal-seeking through Evolutionary Programming* Air Force Cambridge Research Laboratories Final Report Contract AF 19(628)-5927

Fogel L J and Fogel D B 1986 *Artificial Intelligence through Evolutionary Programming* US Army Research Institute Final Report Contract PO-9-X56-1102C-1

Fogel L J, Owens A J and Walsh M J 1964 On the evolution of artificial intelligence *Proc. 5th Natl Symp. on Human Factors in Electronics* (San Diego, CA: IEEE)

——1965 Artificial intelligence through a simulation of evolution *Biophysics and Cybernetic Systems* ed A Callahan, M Maxfield and L J Fogel (Washington, DC: Spartan) pp 131–56

——1966 *Artificial Intelligence through Simulated Evolution* (New York: Wiley)

Frantz D R 1972 *Non-linearities in Genetic Adaptive Search* PhD Thesis, University of Michigan

Fraser A S 1957 Simulation of genetic systems by automatic digital computers *Aust. J. Biol. Sci.* **10** 484–99

Friedberg R M 1958 A learning machine: part I *IBM J.* **2** 2–13

Friedberg R M, Dunham B and North J H 1959 A learning machine: part II *IBM J.* **3** 282–7

Fürst H, Müller P H and Nollau V 1968 Eine stochastische Methode zur Ermittlung der Maximalstelle einer Funktion von mehreren Veränderlichen mit experimentell ermittelbaren Funktionswerten und ihre Anwendung bei chemischen Prozessen *Chem.–Tech.* **20** 400–5

Gehlhaar *et al* 1995Gehlhaar D K *et al* 1995 Molecular recognition of the inhibitor AG-1343 by HIV-1 protease: conformationally flexible docking by evolutionary programming *Chem. Biol.* **2** 317–24

Gell-Mann M 1994 *The Quark and the Jaguar* (New York: Freeman)

Goldberg D E 1983 *Computer-Aided Gas Pipeline Operation using Genetic Algorithms and Rule Learning* PhD Thesis, University of Michigan

——1989 *Genetic Algorithms in Search, Optimization and Machine Learning* (Reading, MA: Addison-Wesley)

Goldberg D E, Deb K and Korb B 1991 Don't worry, be messy *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 24–30

Grefenstette J J (ed) 1985 *Proc. 1st Int. Conf. on Genetic Algorithms and Their Applications (Pittsburgh, PA, 1985)* (Hillsdale, NJ: Erlbaum)

——1987 *Proc. 2nd Int. Conf. on Genetic Algorithms and Their Applications (Cambridge, MA, 1987)* (Hillsdale, NJ: Erlbaum)

Haffner S B and Sebald A V 1993 Computer-aided design of fuzzy HVAC controllers using evolutionary programming *Proc. 2nd Ann. Conf. on Evolutionary Programming (San Diego, CA, 1993)* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 98–107

Heydt G T 1970 *Directed Random Search* PhD Thesis, Purdue University

Hoffmeister F and Schwefel H-P 1990 A taxonomy of parallel evolutionary algorithms *Parcella '90, Proc. 5th Int. Workshop on Parallel Processing by Cellular Automata and Arrays* vol 2, ed G Wolf, T Legendi and U Schendel (Berlin: Academic) pp 97–107

Holland J H 1962 Outline for a logical theory of adaptive systems *J. ACM* **9** 297–314

——1967 Nonlinear environments permitting efficient adaptation *Computer and Information Sciences II* (New York: Academic)

——1969 Adaptive plans optimal for payoff-only environments *Proc. 2nd Hawaii Int. Conf. on System Sciences* pp 917–20

——1971 Processing and processors for schemata *Associative information processing* ed E L Jacks (New York: Elsevier) pp 127–46

——1973 Genetic algorithms and the optimal allocation of trials *SIAM J. Comput.* **2** 88–105

——1975 *Adaptation in Natural and Artificial Systems* (Ann Arbor, MI: University of Michigan Press)

Hollstien R B 1971 *Artificial Genetic Adaptation in Computer Control Systems* PhD Thesis, University of Michigan

Kim J-H and Jeon J-Y 1996 Evolutionary programming-based high-precision controller design *Evolutionary Programming V—Proc. 5th Ann. Conf. on Evolutionary Programming (1996)* ed L J Fogel, P J Angeline and T Bäck (Cambridge, MA: MIT Press)

Klockgether J and Schwefel H-P 1970 Two-phase nozzle and hollow core jet experiments *Proc. 11th Symp. on Engineering Aspects of Magnetohydrodynamics* ed D G Elliott (Pasadena, CA: California Institute of Technology) pp 141–8

Koza J R 1992 *Genetic Programming* (Cambridge, MA: MIT Press)

Koza J R and Andre D 1996 Evolution of iteration in genetic programming *Evolutionary Programming V—Proc. 5th Ann. Conf. on Evolutionary Programming (1996)* ed L J Fogel, P J Angeline and T Bäck (Cambridge, MA: MIT Press)

Kursawe F 1991 A variant of evolution strategies for vector optimization *Parallel Problem Solving from Nature—Proc. 1st Workshop PPSN I (Lecture Notes in Computer Science 496) (Dortmund, 1991)* ed H-P Schwefel and R Männer (Berlin: Springer) pp 193–7

——1992 Naturanaloge Optimierverfahren—Neuere Entwicklungen in der Informatik *Studien zur Evolutorischen Ökonomik II (Schriften des Vereins für Socialpolitik 195 II)* ed U Witt (Berlin: Duncker and Humblot) pp 11–38

Land M and Belew R K 1995 Towards a self-replicating language for computation *Evolutionary Programming IV—Proc. 4th Ann. Conf on Evolutionary Programming (San Diego, CA, 1995)* ed J R McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press) pp 403–13

Larsen R W and Herman J S 1992 A comparison of evolutionary programming to neural networks and an application of evolutionary programming to a navy mission planning problem *Proc. 1st Ann. Conf. on Evolutionary Programming (La Jolla, CA, 1992)* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 127–33

Lichtfuß H J 1965 *Evolution eines Rohrkrümmers* Dipl.-Ing. Thesis, Technical University of Berlin, Hermann Föttinger Institute for Hydrodynamics

Lohmann R 1991 Application of evolution strategy in parallel populations *Parallel Problem Solving from Nature—Proc. 1st Workshop PPSN I (Dortmund, 1991) (Lecture Notes in Computer Science 496)* ed H-P Schwefel and R Männer (Berlin: Springer) pp 198–208

——1992 Structure evolution and incomplete induction *Parallel Problem Solving from Nature 2 (Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier–North-Holland) pp 175–85

Lutter B E and Huntsinger R C 1969 Engineering applications of finite automata *Simulation* **13** 5–11

Männer R and Manderick B (eds) 1992 *Parallel Problem Solving from Nature 2 (Brussels, 1992)* (Amsterdam: Elsevier–North-Holland)

Matyas J 1965 Random optimization *Automation Remote Control* **26** 244–51

——1967 Das zufällige Optimierungsverfahren und seine Konvergenz *Proc. 5th Int. Analogue Computation Meeting (Lausanne, 1967)* **1** 540–4

McDonnell J R 1992 Training neural networks with weight constraints *Proc. 1st Ann. Conf. on Evolutionary Programming (La Jolla, CA, 1992)* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 111–9

McDonnell J R, Andersen B D, Page W C and Pin F 1992 Mobile manipulator configuration optimization using evolutionary programming *Proc. 1st Ann. Conf. on Evolutionary Programming (La Jolla, CA, 1992)* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 52–62

McDonnell J R and Waagen D 1994 Evolving recurrent perceptrons for time-series prediction *IEEE Trans. Neural Networks* **NN-5** 24–38

Michalewicz Z *et al* (eds) 1994 *Proc. 1st IEEE Conf. on Evolutionary Computation (Orlando, FL, 1994)* (Piscataway, NJ: IEEE)

Ostermeier A, Gawelczyk A and Hansen N 1994 Step-size adaptation based on non-local use of selection information *Parallel Problem Solving from Nature—PPSN III Int. Conf. on Evolutionary Computation (Jerusalem, 1994) (Lecture notes in Computer Science 866)* ed Y Davidor, H-P Schwefel and R Männer (Berlin: Springer) pp 189–98

Page W C, Andersen B D and McDonnell J R 1992 An evolutionary programming approach to multi-dimensional path planning *Proc. 1st Ann. Conf. on Evolutionary Programming (La Jolla, CA, 1992)* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 63–70

Porto V W 1992 Alternative methods for training neural networks *Proc. 1st Ann. Conf. on Evolutionary Programming (La Jolla, CA, 1992)* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 100–10

Porto V W, Fogel D B and Fogel L J 1995 Alternative neural network training methods *IEEE Expert* **10** 16–22

Rappl G 1984 *Konvergenzraten von Random-Search-Verfahren zur globalen Optimierung* PhD Thesis, Bundeswehr University

Rastrigin L A 1965 *Random Search in Optimization Problems for Multiparameter Systems (translated from the Russian original: Sluchainyi poisk v zadachakh optimisatsii mnogoarametricheskikh sistem, Zinatne, Riga)* Air Force System Command Foreign Technology Division FTD-HT-67-363

Rawlins G J E (ed) 1991 *Foundations of Genetic Algorithms* (San Mateo, CA: Morgan Kaufmann)

Rechenberg I 1965 *Cybernetic Solution Path of an Experimental Problem* Royal Aircraft Establishment Library Translation 1122

——1973 *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution* (Stuttgart: Frommann–Holzboog)

——1978 Evolutionsstrategien *Simulationsmethoden in der Medizin und Biologie* ed B Schneider and U Ranft (Berlin: Springer) pp 83–114

——1994 *Evolutionsstrategie '94* (Stuttgart: Frommann–Holzboog)

Rizki M M, Tamburino L A and Zmuda M A 1995 Evolution of morphological recognition systems *Evolutionary Programming IV—Proc. 4th Ann. Conf on Evolutionary Programming (San Diego, CA, 1995)* ed J R McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press) pp 95–106

Rosenberg R 1967 *Simulation of Genetic Populations with Biochemical Properties* PhD Thesis, University of Michigan

Rudolph G 1991 Global optimization by means of distributed evolution strategies *Parallel Problem Solving from Nature—Proc. 1st Workshop PPSN I (Dortmund, 1991) (Lecture Notes in Computer Science 496)* ed H-P Schwefel and R Männer (Berlin: Springer) pp 209–13

——1992 Parallel approaches to stochastic global optimization *Parallel Computing: from Theory to Sound Practice, Proc. Eur. Workshop on Parallel Computing* ed W Joosen and E Milgrom (Amsterdam: IOS) pp 256–67

——1994 An evolutionary algorithm for integer programming *Parallel Problem Solving from Nature—PPSN III Int. Conf. on Evolutionary Computation (Jerusalem, 1994) (Lecture notes in Computer Science 866)* ed Y Davidor, H-P Schwefel and R Männer (Berlin: Springer) pp 139–48

Rudolph G and Sprave J 1995 A cellular genetic algorithm with self-adjusting acceptance threshold *Proc. 1st IEE/IEEE Int. Conf. on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA '95) (Sheffield, 1995)* (London: IEE) pp 365–72

Sampson J R 1981 *A Synopsis of the Fifth Annual Ann Arbor Adaptive Systems Workshop* Department of Computing and Communication Science, Logic of Computers Group Technical Report University of Michigan

Saravanan N, Fogel D B and Nelson K M 1995 A comparison of methods for self-adaptation in evolutionary algorithms *BioSystems* **36** 157–66

Satterthwaite F E 1959a Random balance experimentation *Technometrics* **1** 111–37

——1959b *REVOP or Random Evolutionary Operation* Merrimack College Technical Report 10-10-59

Schaffer J D (ed) 1989 *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, WA, 1989)* (San Mateo, CA: Morgan Kaufmann)

Schütz M and Sprave J 1996 Application of parallel mixed-integer evolution strategies with mutation rate pooling *Evolutionary Programming V—Proc. 5th Ann. Conf. on Evolutionary Programming (1996)* ed L J Fogel, P J Angeline and T Bäck (Cambridge, MA: MIT Press)

Schwefel H-P 1965 *Kybernetische Evolution als Strategie der experimentellen Forschung in der Strömungstechnik* Dipl.-Ing. Thesis, Technical University of Berlin, Hermann Föttinger Institute for Hydrodynamics

——1968 *Experimentelle Optimierung einer Zweiphasendüse Teil I* AEG Research Institute Project MHD-Staustrahlrohr 11034/68 Technical Report 35

——1974 *Adaptive Mechanismen in der biologischen Evolution und ihr Einflußauf die Evolutionsgeschwindigkeit* Technical University of Berlin Working Group of Bionics and Evolution Techniques at the Institute for Measurement and Control Technology Technical Report Re 215/3

——1975 *Binäre Optimierung durch somatische Mutation* Working Group of Bionics and Evolution Techniques at the Institute of Measurement and Control Technology of the Technical University of Berlin and the Central Animal Laboratory of the Medical Highschool of Hannover Technical Report

——1977 *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie (Interdisciplinary Systems Research 26)* (Basle: Birkhäuser)

——1981 *Numerical Optimization of Computer Models* (Chichester: Wiley)

——1987 Collective phenomena in evolutionary systems *Problems of Constancy and Change—the Complementarity of Systems Approaches to Complexity, Papers Presented at the 31st Ann. Meeting Int. Society Gen. Syst. Res.* vol 2, ed P Checkland and I Kiss (Budapest: International Society for General System Research) pp 1025–33

——1995 *Evolution and Optimum Seeking* (New York: Wiley)

Schwefel H-P and Bäck T 1995 Evolution strategies II: theoretical aspects *Genetic Algorithms in Engineering and Computer Science Proc. 1st Short Course EUROGEN-95* ed G Winter, J Périaux, M Galán and P Cuesta (New York: Wiley) pp 127–40

Schwefel H-P and Männer R (eds) 1991 *Parallel Problem Solving from Nature—Proc. 1st Workshop PPSN I (Dortmund, 1991) (Lecture Notes in Computer Science 496)* (Berlin: Springer)

Schwefel H-P and Rudolph G 1995 Contemporary evolution strategies *Advances in Artificial Life—Proc. 3rd Eur. Conf. on Artificial Life (ECAL'95) (Lecture Notes in Computer Science 929)* ed F Morán, A Moreno, J J Merelo and P Chacón (Berlin: Springer) pp 893–907

Sebald A V and Fogel D B 1992 Design of fault-tolerant neural networks for pattern classification *Proc. 1st Ann. Conf. on Evolutionary Programming (La Jolla, CA, 1992)* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 90–9

Sebald A V, Schlenzig J and Fogel D B 1992 Minimax design of CMAC encoded neural controllers for systems with variable time delay *Proc. 1st Ann. Conf. on Evolutionary Programming (La Jolla, CA, 1992)* ed D B Fogel and W Atmar (La Jolla, CA: Evolutionary Programming Society) pp 120–6

Smith S F 1980 *A Learning System Based on Genetic Adaptive Algorithms* PhD Thesis, University of Pittsburgh

Spendley W, Hext G R and Himsworth F R 1962 Sequential application of simplex designs in optimisation and evolutionary operation *Technometrics* **4** 441–61

Sprave J 1994 Linear neighborhood evolution strategy *Proc. 3rd Ann. Conf. on Evolutionary Programming (San Diego, CA, 1994)* ed A V Sebald and L J Fogel (Singapore: World Scientific) pp 42–51

Stewart E C, Kavanaugh W P and Brocker D H 1967 Study of a global search algorithm for optimal control *Proc. 5th Int. Analogue Computation Meeting (Lausanne, 1967)* vol 1, pp 207–30

Takeuchi A 1980 Evolutionary automata—comparison of automaton behavior and Restle's learning model *Information Sci.* **20** 91–9

Wetzel A 1983 *Evaluation of the Effectiveness of Genetic Algorithms in Combinatorial Optimization* unpublished manuscript, University of Pittsburgh

PART B

FUNDAMENTAL CONCEPTS OF EVOLUTIONARY COMPUTATION

B1

Evolutionary Algorithms and Their Standard Instances

Contents

B1.1 Introduction

Thomas Bäck

Abstract

Within this introduction to Chapter B1, we present a general outline of an evolutionary algorithm, which is consistent with all mainstream instances of evolutionary computation and summarizes the major features common to evolutionary computation approaches; that is, a population of individuals, recombination and/or mutation, and selection. Some of the differences between genetic algorithms, evolution strategies, and evolutionary programming are briefly mentioned to provide a short overview of the features that are most emphasized by these different approaches. Furthermore, the basic characteristics of genetic programming and classifier systems are also outlined.

B1.1.1 General outline of evolutionary algorithms

Since they are gleaned from the model of organic evolution, all basic instances of evolutionary algorithms share a number of common properties, which are mentioned here to characterize the prototype of a general evolutionary algorithm:

(i) Evolutionary algorithms utilize the collective learning process of a population of individuals. Usually, each individual represents (or encodes) a search point in the space of potential solutions to a given problem. Additionally, individuals may also incorporate further information; for example, *strategy parameters* of the evolutionary algorithm. C1.3.2, C3.2.2

(ii) Descendants of individuals are generated by randomized processes intended to model *mutation* and *recombination*. Mutation corresponds to an erroneous self-replication of individuals (typically, small modifications are more likely than large ones), while recombination exchanges information between two or more existing individuals. C3.2 C3.3

(iii) By means of evaluating individuals in their environment, a measure of quality or fitness value can be assigned to individuals. As a minimum requirement, a comparison of individual fitness is possible, yielding a binary decision (better or worse). According to the fitness measure, the *selection* process favors better individuals to reproduce more often than those that are relatively worse. C2

These are just the most general properties of evolutionary algorithms, and the instances of evolutionary algorithms as described in the following sections of this chapter use the components in various different ways and combinations. Some basic differences in the utilization of these principles characterize the mainstream instances of evolutionary algorithms; that is, *genetic algorithms*, *evolution strategies*, and *evolutionary programming*. See D B Fogel (1995) and Bäck (1996) for a detailed overview of similarities and differences of these instances and Bäck and Schwefel (1993) for a brief comparison. B1.2 B1.3 B1.4

- Genetic algorithms (originally described by Holland (1962, 1975) at Ann Arbor, Michigan, as so-called adaptive or reproductive plans) emphasize *recombination (crossover)* as the most important search operator and apply *mutation* with very small probability solely as a 'background operator.' They also use a probabilistic selection operator *(proportional selection)* and often rely on a *binary representation* of individuals. C3.3 C3.2 C2.2, C1.2

- Evolution strategies (developed by Rechenberg (1965, 1973) and Schwefel (1965, 1977) at the Technical University of Berlin) use normally distributed mutations to modify *real-valued vectors* and emphasize *mutation* and *recombination* as essential operators for searching in the search space and in the strategy parameter space at the same time. The *selection operator* is deterministic, and parent and offspring population sizes usually differ from each other.
- Evolutionary programming (originally developed by Lawrence J Fogel (1962) at the University of California in San Diego, as described in Fogel *et al* (1966) and refined by David B Fogel (1992) and others) emphasizes mutation and does not incorporate the recombination of individuals. Similarly to evolution strategies, when approaching real-valued optimization problems, evolutionary programming also works with normally distributed mutations and extends the evolutionary process to the strategy parameters. The *selection operator* is probabilistic, and presently most applications are reported for search spaces involving real-valued vectors, but the algorithm was originally developed to evolve *finite-state machines*.

In addition to these three mainstream methods, which are described in detail in the following sections, *genetic programming*, *classifier systems*, and *hybridizations* of evolutionary algorithms with other techniques are considered in this chapter. As an introductory remark, we only mention that genetic programming applies the evolutionary search principle to automatically develop computer programs in suitable *languages* (often LISP, but others are possible as well), while classifier systems search the space of production rules (or sets of rules) of the form '`IF <condition> THEN <action>`'.

A variety of different representations of individuals and corresponding operators are presently known in evolutionary algorithm research, and it is the aim of Part C (Evolutionary Computation Models) to present all these in detail. Here, we will use Part C as a construction kit to assemble the basic instances of evolutionary algorithms.

As a general framework for these basic instances, we define I to denote an arbitrary space of individuals $\boldsymbol{a} \in I$, and $F : I \to \mathbb{R}$ to denote a real-valued fitness function of individuals. Using μ and λ to denote parent and offspring population sizes, $P(t) = (\boldsymbol{a}_1(t), \ldots, \boldsymbol{a}_\mu(t)) \in I^\mu$ characterizes a population at generation t. Selection, mutation, and recombination are described as operators $s : I^\lambda \to I^\mu$, $m : I^\kappa \to I^\lambda$, and $r : I^\mu \to I^\kappa$ that transform complete populations. By describing all operators on the population level (though this is counterintuitive for mutation), a high-level perspective is adopted, which is sufficently general to cover different instances of evolutionary algorithms. For mutation, the operator can of course be reduced to the level of single individuals by defining m through a multiple application of a suitable operator $m' : I \to I$ on individuals.

These operators typically depend on additional sets of parameters Θ_s, Θ_m, and Θ_r which are characteristic for the operator and the representation of individuals. Additionally, an initialization procedure generates a population of individuals (typically at random, but an initialization with known starting points should of course also be possible), an evaluation routine determines the fitness values of the individuals of a population, and a termination criterion is applied to determine whether or not the algorithm should stop.

Putting all this together, a basic evolutionary algorithm reduces to the simple recombination–mutation–selection loop as outlined below:

Input: $\mu, \lambda, \Theta_\iota, \Theta_r, \Theta_m, \Theta_s$
Output: $\boldsymbol{a}^*$, the best individual found during the run, or
P^*, the best population found during the run.

```
1   t ← 0;
2   P(t) ← initialize(μ);
3   F(t) ← evaluate(P(t), μ);
4   while (ι(P(t), Θ_ι) ≠ true) do
5       P'(t)  ← recombine(P(t), Θ_r);
6       P''(t) ← mutate(P'(t), Θ_m);
7       F(t)   ← evaluate(P''(t), λ);
8       P(t+1) ← select(P''(t), F(t), μ, Θ_s);
9       t      ← t + 1;
    od
```

After initialization of t (line 1) and the population $P(t)$ of size μ (line 2) as well as its fitness evaluation (line 3), the while-loop is entered. The termination criterion ι might depend on a variety of parameters, which are summarized here by the argument Θ_ι. Similarly, recombination (line 5), mutation

(line 6), and selection (line 8) depend on a number of algorithm-specific additional parameters. While $P(t)$ consists of μ individuals, $P'(t)$ and $P''(t)$ are assumed to be of size κ and λ, respectively. Of course, $\lambda = \kappa = \mu$ is allowed and is the default case in genetic algorithms. The setting $\kappa = \mu$ is also often used in evolutionary programming (without recombination), but it depends on the application and the situation is quickly changing. Either recombination or mutation might be absent from the main loop, such that $\kappa = \mu$ (absence of recombination) or $\kappa = \lambda$ (absence of mutation) is required in these cases. The selection operator selects μ individuals from $P''(t)$ according to the fitness values $\boldsymbol{F}(t)$, t is incremented (line 9), and the body of the main loop is repeated.

The input parameters of this general evolutionary algorithm include the population sizes μ and λ as well as the parameter sets Θ_t, Θ_r, Θ_m, and Θ_s of the basic operators. Notice that we allow recombination to equal the identity mapping; that is, $P''(t) = P'(t)$ is possible.

The following sections of this chapter present the common evolutionary algorithms as particular instances of the general scheme.

References

Bäck T 1996 *Evolutionary Algorithms in Theory and Practice* (New York: Oxford University Press)

Bäck T and Schwefel H-P 1993 An overview of evolutionary algorithms for parameter optimization *Evolutionary Computation* **1**(1) 1–23

Fogel D B 1992 *Evolving Artificial Intelligence* PhD Thesis, University of California, San Diego

——1995 *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence* (Piscataway, NJ: IEEE)

Fogel L J 1962 Autonomous automata *Industr. Res.* **4** 14–9

Fogel L J, Owens A J and Walsh M J 1966 *Artificial Intelligence through Simulated Evolution* (New York: Wiley)

Holland J H 1962 Outline for a logical theory of adaptive systems *J. ACM* **3** 297–314

——1975 *Adaptation in Natural and Artificial Systems* (Ann Arbor, MI: University of Michigan Press)

Rechenberg I 1965 Cybernetic solution path of an experimental problem *Library Translation No 1122* Royal Aircraft Establishment, Farnborough, UK

——1973 *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution* (Stuttgart: Frommann-Holzboog)

Schwefel H-P 1965 *Kybernetische Evolution als Strategie der experimentellen Forschung in der Strömungstechnik* Diplomarbeit, Technische Universität, Berlin

——1977 *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie* Interdisciplinary Systems Research, vol 26 (Basel: Birkhäuser)

Further reading

The introductory section to evolutionary algorithms certainly provides the right place to mention the most important books on evolutionary computation and its subdisciplines. The following list is not intended to be complete, but only to guide the reader to the literature.

1. Bäck T 1996 *Evolutionary Algorithms in Theory and Practice* (New York: Oxford University Press)

   A presentation and comparison of evolution strategies, evolutionary programming, and genetic algorithms with respect to their behavior as parameter optimization methods. Furthermore, the role of mutation and selection in genetic algorithms is discussed in detail, arguing that mutation is much more useful than usually claimed in connection with genetic algorithms.

2. Goldberg D E 1989 *Genetic Algorithms in Search, Optimization, and Machine Learning* (Reading, MA: Addison-Wesley)

   An overview of genetic algorithms and classifier systems, discussing all important techniques and operators used in these subfields of evolutionary computation.

3. Rechenberg I 1994 *Evolutionsstrategie '94* Werkstatt Bionik und Evolutionstechnik, vol 1 (Stuttgart: Frommann-Holzboog)

   A description of evolution strategies in the form used by Rechenberg's group in Berlin, including a reprint of (Rechenberg 1973).

4. Schwefel H-P 1995 *Evolution and Optimum Seeking* Sixth-Generation Computer Technology Series (New York: Wiley)

   The most recent book on evolution strategies, covering the (μ, λ)-strategy and all aspects of self-adaptation of strategy parameters as well as a comparison of evolution strategies with classical optimization methods.

5. Fogel D B 1995 *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence* (Piscataway, NJ: IEEE)

   The book covers all three main areas of evolutionary computation (i.e. genetic algorithms, evolution strategies, and evolutionary programming) and discusses the potential for using simulated evolution to achieve machine intelligence.

6. Michalewicz Z 1994 *Genetic Algorithms + Data Structures = Evolution Programs* (Berlin: Springer)

   Michalewicz also takes a more general view at evolutionary computation, thinking of evolutionary heuristics as a principal method for search and optimization, which can be applied to any kind of data structure.

7. Kinnear K E 1994 *Advances in Genetic Programming* (Cambridge, MA: MIT Press)

   This collection of articles summarizes the state of the art in genetic programming, emphasizing other than LISP-based approaches to genetic programming.

8. Koza J R 1992 *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (Cambridge, MA: MIT Press)

9. Koza J R 1994 *Genetic Programming II* (Cambridge, MA: MIT Press)

   The basic books for genetic programming using LISP programs, demonstrating the feasibility of the method by presenting a variety of application examples from diverse fields.

B1.2 Genetic algorithms

Larry J Eshelman

Abstract

This section gives an overview of genetic algorithms (GAs), describing the canonical GAs proposed by John Holland and developed by his first students, as well as later variations. Whereas many of the GA variations are distinguished by the methods used for selection, GAs as a class, including later variations, are distinguished from other evolutionary algorithms by their reliance upon crossover. Selection and crossover are discussed in some detail, and representation and parallelization are discussed briefly.

B1.2.1 Introduction

Genetic algorithms (GAs) are a class of evolutionary algorithms first proposed and analyzed by John Holland (1975). There are three features which distinguish GAs, as first proposed by Holland, from other evolutionary algorithms: (i) the representation used—*bitstrings*; (ii) the method of selection— *proportional selection*; and (iii) the primary method of producing variations—*crossover*. Of these three features, however, it is the emphasis placed on crossover which makes GAs distinctive. Many subsequent GA implementations have adopted alternative methods of selection, and many have abandoned bitstring representations for other representations more amenable to the problems being tackled. Although many alternative methods of crossover have been proposed, in almost every case these variants are inspired by the spirit which underlies Holland's original analysis of GA behavior in terms of the processing of schemata or building blocks. It should be pointed out, however, that the *evolution strategy* paradigm has added crossover to its repertoire, so that the distinction between classes of evolutionary algorithms has become blurred (Bäck *et al* 1991).

C1.2
C2.2, C3.3
B1.3

We shall begin by outlining what might be called the canonical GA, similar to that described and analyzed by Holland (1975) and Goldberg (1987). We shall introduce a framework for describing GAs which is richer than needed but which is convenient for describing some variations with regard to the method of selection. First we shall introduce some terminology. The individual structures are often referred to as chromosomes. They are the genotypes that are manipulated by the GA. The evaluation routine decodes these structures into some phenotypical structure and assigns a fitness value. Typically, but not necessarily, the chromosomes are bitstrings. The value at each locus on the bitstring is referred to as an allele. Sometimes the individuals loci are also called genes. At other times genes are combinations of alleles that have some phenotypical meaning, such as parameters.

B1.2.2 Genetic algorithm basics and some variations

An initial population of individual structures $P(0)$ is generated (usually randomly) and each individual is evaluated for fitness. Then some of these individuals are selected for mating and copied (select_repro) to the mating buffer $C(t)$. In Holland's original GA, individuals are chosen for mating probabilistically, assigning each individual a probability proportional to its observed performance. Thus, better individuals are given more opportunities to produce offspring (reproduction with emphasis). Next the genetic operators (usually mutation and crossover) are applied to the individuals in the mating buffer, producing offspring $C'(t)$. The rates at which mutation and crossover are applied are an implementation decision.

If the rates are low enough, it is likely that some of the offspring produced will be identical to their parents. Other implementation details are how many offspring are produced by crossover (one or two), and how many individuals are selected and paired in the mating buffer. In Holland's original description, only one pair is selected for mating per cycle. The pseudocode for the genetic algorithm is as follows:

```
begin
    t = 0;
    initialize P(t);
    evaluate structures in P(t);
    while termination condition not satisfied do
    begin
        t = t + 1;
        select_repro C(t) from P(t-1);
        recombine and mutate structures in C(t) forming C'(t);
        evaluate structures in C'(t);
        select_replace P(t) from C'(t) and P(t-1);
    end
end
```

After the new offspring have been created via the genetic operators the two populations of parents and children must be merged to create a new population. Since most GAs maintain a fixed-sized population M, this means that a total of M individuals need to be selected from the parent and child populations to create a new population. One possibility is to use all the children generated (assuming that the number is not greater than M) and randomly select (without any bias) individuals from the old population to bring the new population up to size M. If only one or two new offspring are produced, this in effect means randomly replacing one or two individuals in the old population with the new offspring. (This is what Holland's original proposal did.) On the other hand, if the number of offspring created is equal to M, then the old parent population is completely replaced by the new population.

There are several opportunities for biasing selection: selection for reproduction (or mating) and selection from the parent and child populations to produce the new population. The GAs most closely associated with Holland do all their biasing at the reproduction selection stage. Even among these GAs, however, there are a number of variations. If reproduction with emphasis is used, then the probability of an individual being chosen is a function of its observed fitness. A straightforward way of doing this would be to total the fitness values assigned to all the individuals in the parent population and calculate the probability of any individual being selected by dividing its fitness by the total fitness. One of the properties of this way of assigning probabilities is that the GA will behave differently on functions that seem to be equivalent from an optimization point of view such as $y = ax^2$ and $y = ax^2 + b$. If the b value is large in comparison to the differences in the value produced by the ax^2 term, then the differences in the probabilities for selecting the various individuals in the population will be small, and selection pressure will be very weak. This often happens as the population converges upon a narrow range of values. One way
C2.2.2 of avoiding this behavior is to *scale* the fitness function, typically to the worst individual in the population (De Jong 1975). Hence the measure of fitness used in calculating the probability for selecting an individual is not the individual's absolute fitness, but its fitness relative to the worst individual in the population.

Although scaling can eliminate the problem of not enough selection pressure, often GAs using fitness proportional selection suffer from the opposite problem—too much selection pressure. If an individual is found which is much better than any other, the probability of selecting this individual may become quite high (especially if scaling to the worst is used). There is the danger that many copies of this individual will be placed in the mating buffer, and this individual (and its similar offspring) will rapidly take over the population (premature convergence). One way around this is to replace fitness proportional selection with
C2.4 *ranked selection* (Whitley 1989). The individuals in the parent population are ranked, and the probability of selection is a linear function of rank rather than fitness, where the 'steepness' of this function is an adjustable parameter.

C2.3 Another popular method of performing selection is *tournament selection* (Goldberg and Deb 1991). A small subset of individuals is chosen at random, and then the best individual (or two) in this set is (are) selected for the mating buffer. Tournament selection, like rank selection, is less subject to rapid takeover by good individuals, and the selection pressure can be adjusted by controlling the size of the subset used.

Another common variation of those GAs that rely upon reproduction selection for their main source of selection bias is to maintain one copy of the best individual found so far (De Jong 1975). This is referred to as the *elitist strategy*. It is actually a method of biased parent selection, where the best member of the parent population is chosen and all but one of the M members of the child population are chosen. Depending upon the implementation, the selection of the child to be replaced by the best individual from the parent population may or may not be biased. C2.7.4

A number of GA variations make use of biased replacement selection. Whitley's GENITOR, for example, creates one child each cycle, selecting the parents using ranked selection, and then replacing the worst member of the population with the new child (Whitley 1989). Syswerda's steady-state GA creates two children each cycle, selecting parents using ranked selection, and then stochastically choosing two individuals to be replaced, with a bias towards the worst individuals in the parent population (Syswerda 1989). Eshelman's CHC uses unbiased reproductive selection by randomly pairing all the members of the parent population, and then replacing the worst individuals of the parent population with the better individuals of the child population. (In effect, the offspring and parent populations are merged and the best M (population size) individuals are chosen.) Since the new offspring are only chosen by CHC if they are better than the members of the parent population, the selection of both the offspring and parent populations is biased (Eshelman 1991).

These methods of replacement selection, and especially that of CHC, resemble the $(\mu+\lambda)$ ES method of selection sometimes originally used by evolution strategies (ESs) (Bäck *et al* 1991). From μ parents λ offspring are produced; the μ parents and λ offspring are merged; and the best μ individuals are chosen to form the new parent population. The other ES selection method, (μ, λ) ES, places all the bias in the child selection stage. In this case, μ parents produce λ offspring ($\lambda > \mu$), and the best μ offspring are chosen to replace the parent population. Mühlenbein's breeder GA also uses this selection mechanism (Mühlenbein and Schlierkamp-Voosen 1993). C2.4.4 C2.4.4

Often a distinction is made between *generational* and *steady-state* GAs. Unfortunately, this distinction tends to merge two properties that are quite independent: whether the replacement strategy of the GA is biased or not and whether the GA produces one (or two) versus many (usually M) offspring each cycle. Syswerda's steady-state GA, like Whitley's GENITOR, allows only one mating per cycle and uses a biased replacement selection, but there are also GAs that combine multiple matings per cycle with biased replacement selection (CHC) as well as a whole class of ESs ($(\mu+\lambda)$ ES). Furthermore, the GA described by Holland (1975) combined a single mating per cycle and unbiased replacement selection. Of these two features, it would seem that the most significant is the replacement strategy. De Jong and Sarma (1993) found that the main difference between GAs allowing many matings versus few matings per cycle is that the latter have a higher variance in performance. C2.7.3

The choice between a biased and an unbiased replacement strategy, on the other hand, is a major determinant of GA behavior. First, if biased replacement is used in combination with biased reproduction, then the problem of premature convergence is likely to be compounded. (Of course this will depend upon other factors, such as the size of the population, whether ranked selection is used, and, if so, the setting of the selection bias parameter.) Second, the obvious shortcoming of unbiased replacement selection can turn out to be a strength. On the negative side, replacing the parents by the children, with no mechanism for keeping those parents that are better than any of the children, risks losing, perhaps forever, very good individuals. On the other hand, replacing the parents by the children can allow the algorithm to wander, and it may be able to wander out of a local minimum that would trap a GA relying upon biased replacement selection. Which is the better strategy cannot be answered except in the context of the other mechanisms of the algorithm (as well as the nature of the problem being solved). Both Syswerda's steady-state GA and Whitley's GENITOR combine a biased replacement strategy with a mechanism for eliminating children which are duplicates of any member in the parent population. CHC uses unbiased reproductive selection, relying solely upon biased replacement selection as its only source of selection pressure, and uses several mechanisms for maintaining diversity (not mating similar individuals and seeded restarts), which allow it to take advantage of the preserving properties of a deterministic replacement strategy without suffering too severely from its shortcomings.

B1.2.3 Mutation and crossover

All evolutionary algorithms work by combining selection with a mechanism for producing variations. The best known mechanism for producing variations is *mutation*, where one allele of a gene is randomly C3.2

replaced by another. In other words, new trial solutions are created by making small, random changes in the representation of prior trial solutions. If a binary representation is used, then mutation is achieved by 'flipping' bits at random. A commonly used rate of mutation is one over the string length. For example, if the chromosome is one hundred bits long, then the mutation rate is set so that each bit has a probability of 0.01 of being flipped.

Although most GAs use mutation along with crossover, mutation is sometimes treated as if it were a background operator for assuring that the population will consist of a diverse pool of alleles that can be exploited by crossover. For many optimization problems, however, an evolutionary algorithm using mutation without crossover can be very effective (Mathias and Whitley 1994). This is not to suggest that crossover never provides an added benefit, but only that one should not disparage mutation.

The intuitive idea behind crossover is easy to state: given two individuals who are highly fit, but for different reasons, ideally what we would like to do is create a new individual that combines the best features from each. Of course, since we presumably do not know which features account for the good performance (if we did we would not need a search algorithm), the best we can do is to recombine features at random. This is how crossover operates. It treats these features as building blocks scattered throughout the population and tries to recombine them into better individuals via crossover. Sometimes crossover will combine the worst features from the two parents, in which case these children will not survive for long. But sometimes it will recombine the best features from two good individuals, creating even better individuals, provided these features are compatible.

Suppose that the representation is the classical bitstring representation: individual solutions in our population are represented by binary strings of zeros and ones of length L. A GA creates new individuals via crossover by choosing two strings from the parent population, lining them up, and then creating two new individuals by swapping the bits at random between the strings. (In some GAs only one individual is created and evaluated, but the procedure is essentially the same.) Holland originally proposed that the swapping be done in segments, not bit by bit. In particular, he proposed that a single locus be chosen
C3.3 at random and all bits after that point be swapped. This is known as *one-point crossover*. Another common form of crossover is two-point crossover which involves choosing two points at random and swapping the corresponding segments from the two parents defined by the two points. There are of course many possible variants. The best known alternative to one- and two-point crossover is *uniform crossover*. Uniform crossover randomly swaps individual bits between the two parents (i.e. exchanges between the parents the values at loci chosen at random).

B2.5 Following Holland, GA behavior is typically analyzed in terms of *schemata*. Given a space of structures represented by bitstrings of length L, schemata represent partitions of the search space. If the bitstrings of length L are interpreted as vectors in a L-dimensional hypercube, then schemata are hyperplanes of the space. A schema can be represented by a string of L symbols from the set $0, 1, \#$ where # is a 'wildcard' matching either 0 or 1. Each string of length L may be considered a sample from the partition defined by a schema if it matches the schema at each of the defined positions (i.e. the non-# loci). For example, the string 011001 instantiates the schema 01##0#. Each string, in fact, instantiates 2^L schemata.

Two important schema properties are order and defining length. The order of a schema is the number of defined loci (i.e. the number of non-# symbols). For example the schema #01##1### is an order 3 schema. The defining length is the distance between the loci of the first and last defined positions. The defining length of the above schema is four since the loci of the first and last defined positions are 2 and 6.

From the hyperplane analysis point of view, a GA can be interpreted as focusing its search via crossover upon those hyperplane partition elements that have on average produced the best-performing individuals. Over time the search becomes more and more focused as the population converges since the degree of variation used to produce new offspring is constrained by the remaining variation in the population. This is because crossover has the property that Radcliffe refers to as respect—if two parents are instances of the same schema, the child will also be an instance (Radcliffe 1991). If a particular schema conveys high fitness values to its instances, then the population is likely to converge on the defining bits of this schema. Once it so converges, all offspring will be instances of this schema. This means that as the population converges, the search becomes more and more focused on smaller and smaller partitions of the search space.

It is useful to contrast crossover with mutation in this regard. Whereas mutation creates variations by flipping bits randomly, crossover is restricted to producing variations at those loci on which the population

has not yet converged. Thus crossover, and especially bitwise versions of crossover, can be viewed as a form of adaptive mutation, or convergence-controlled variation (CCV).

The standard explanation of how GAs operate is often referred to as the *building block hypothesis*. B2.5.3 According to this hypothesis, GAs operate by combining small building blocks into larger building blocks. The intuitive idea behind recombination is that by combining features (or building blocks) from two good parents crossover will often produce even better children; for example, a mother with genes for sharp teeth and a father with genes for sharp claws will have the potential of producing some children who have both features. More formally, the building blocks are the schemata discussed above.

Loosely interpreted, the building block hypothesis is another way of asserting that GAs operate through a process of CCV. The building block hypothesis, however, is often given a stronger interpretation. In particular, crossover is seen as having the added value of being able to recombine middle-level building blocks that themselves cannot be built from lower-level building blocks (where *level* refers to either the defining length or order, depending on the crossover operator). We shall refer to this explanation as to how GAs work as the strict building block hypothesis (SBBH), and contrast it with the weaker convergence-controlled variation hypothesis (CCVH).

To differentiate these explanations, it is useful to compare crossover with an alternative mechanism for achieving CCV. Instead of pairing individuals and swapping segments or bits, a more direct method of generating CCVs is to use the distribution of the allele values in the population to generate new offspring. This is what Syswerda's bitwise simulated crossover (BSC) algorithm does (Syswerda 1993). In effect, the distribution of allele values is used to generate a vector of allele probabilities, which in turn is used to generate a string of ones and zeros. Baluja's PBIL goes one step further and eliminates the population, and simply keeps a probability vector of allele values, using an update rule to modify it based on the fitness of the samples generated (Baluja 1995).

The question is, if one wants to take advantage of CCV with its ability to adapt, why use crossover, understood as involving pairwise mating, rather than one of these *poolwise* schemes? One possible answer is that the advantage is only one of implementation. The pairwise implementation does not require any centralized bookkeeping mechanism. In other words, crossover (using pairwise mating) is simply nature's way of implementing a decentralized version of CCV.

A more theoretically satisfying answer is that pairwise mating is better able to preserve essential linkages among the alleles. One manifestation of this is that there is no obvious way to implement a segment-based version of poolwise mating, but this point also applies if we compare poolwise mating with only crossover operators that operate at the bit level, such as uniform crossover. If two allele values are associated in some individual, the probability of these values being associated in the children is much higher for pairwise mating than poolwise. To see this consider an example. Suppose the population size is 100, and that an individual of average fitness has some unique combination of allele values, say all ones in the first three positions. This individual will have a 0.01 probability (one out of 100) of being selected for mating, assuming it is of average fitness. If uniform crossover is being used, with a 0.5 probability of swapping the values at each locus, and one offspring is being produced per mating, then the probability of the three allele values being propagated without disruption has a lower bound of 0.125 (0.5^3). This is assuming the worst-case scenario that every other member in the population has all zeros in the first three positions (and ignoring the possibility of mating this individual with a copy of itself). Thus, the probability of propagating this schema is 0.00125 ($0.01 * 0.125$). On the other hand, if BSC is being used, then the probability of propagating this schema is much lower. Since there is only one instance of this individual in the population, there is only one chance in 100 of propagating each allele and only 0.000 001 (0.01^3) of propagating all three.

Ultimately, one is faced with a tradeoff: the enhanced capability of pairwise mating to propagate difficult-to-find schemata is purchased at the risk of increased hitchhiking; that is, the population may prematurely converge on bits that do not convey additional fitness but happen to be present in the individuals that are instances of good schemata. According to both the CCVH and the SBBH, crossover must not simply preserve and propagate good schemata, but must also recombine them with other good schemata. Recombination, however, requires that these good schemata be tried in the context of other schemata. In order to determine which schemata are the ones contributing to fitness, we must test them in many different contexts, and this involves prying apart the defining positions that contribute to fitness from those that are spurious, but the price for this reduced hitchhiking is higher disruption (the breaking up of the good schemata). This price will be too high if the algorithm cannot propagate critical, highly valued, building blocks or, worse yet, destroys them in the next crossover cycle.

This tradeoff applies not only to the choice between poolwise and pairwise methods of producing variation, but also to the choice between various methods of crossover. Uniform crossover, for example, is less prone to hitchhiking than two-point crossover, but is also more disruptive, and poolwise mating schemes are even more disruptive than uniform crossover. In Holland's original analysis this tradeoff between preserving the good schemata while performing vigorous recombination is downplayed by using a segment-based crossover operator such as one- or two-point crossover and assuming that the important building blocks are of short defining length. Unfortunately, for the types of problem to which GAs are supposedly ideally suited—those that are highly complex with no tractable analytical solution—there is no *a priori* reason to assume that the problem will, or even can, be represented so that important building blocks will be those with short defining length. To handle this problem Holland proposed an inversion operator that could reorder the loci on the string, and thus be capable of finding a representation that had building blocks with short defining lengths. The inversion operator, however, has not proven sufficiently effective in practice at recoding strings on the fly. To overcome this linkage problem, Goldberg has
C4.2.4 proposed what he calls *messy GAs*, but, before discussing messy GAs, it will be helpful to describe a class of problems that illustrate these linkage issues: deceptive problems.

B2.7.1 *Deception* is a notion introduced by Goldberg (1987). Consider two incompatible schemata, A and B. A problem is deceptive if the average fitness of A is greater than B even though B includes a string that has a greater fitness than any member of A. In practice this means that the lower-order building blocks lead the GA away from the global optimum. For example, consider a problem consisting of five-bit segments for which the fitness of each is determined as follows (Liepins and Vose 1991). For each *one* the segment receives a point, and thus five points for all *ones*, but for all *zeros* it receives a value greater than five. For problems where the value of the optimum is between five and eight the problem is fully deceptive (i.e. all relevant lower-order hyperplanes lead toward the deceptive attractor). The total fitness is the sum of the fitness of the segments.

It should be noted that it is probably a mistake to place too much emphasis on the formal definition of deception (Grefenstette 1993). What is really important is the concept of being misled by the lower-order building blocks. Whereas the formal definition of deception stresses the average fitness of the hyperplanes taken over the entire search space, selection only takes into account the observed average fitness of hyperplanes (those in the actual population). The interesting set of problems is those that are misleading in that manipulation of the lower-order building blocks is likely to lead the search away from the middle-level building blocks that constitute the optimum solution, whether these middle-level building blocks are deceptive in the formal sense or not. In the above class of functions, even when the value of the optimum is greater than eight (and so not fully deceptive), but still not very large, e.g. ten, the problem is solvable by a GA using segment-based crossover, very difficult for a GA using bitwise uniform crossover, and all but impossible for a poolwise-based algorithm like BSC.

As long as the deceptive problem is represented so that the loci of the positions defining the building blocks are close together on the string, it meets Holland's original assumption that the important building blocks are of short defining length. The GA will be able to exploit this information using one- or two-point crossover—the building blocks will have a low probability of being disrupted, but will be vigorously recombined with other building blocks along the string. If, on the other hand, the bits constituting the deceptive building blocks are maximally spread out on the chromosome, then a crossover operator such as one- or two-point crossover will tend to break up the good building blocks. Of course, maximally spreading the deceptive bits along the string is the extreme case, but bunching them together is the opposite extreme.

Since one is not likely to know enough about the problem to be able to guarantee that the building blocks are of short defining length, segmented crossover loses its advantage over bitwise crossover. It is true that bitwise crossover operators are more disruptive, but there are several solutions to this problem. First, there are bitwise crossover operators that are much less disruptive than the standard uniform crossover operator (Spears and De Jong 1991, Eshelman and Schaffer 1995). Second, the problem of preservation can often be ameliorated by using some form of replacement selection so that good individuals survive until they are replaced by better individuals (Eshelman and Schaffer 1995). Thus a disruptive form of crossover such as uniform crossover can be used and good schemata can still be preserved. Uniform crossover will still make it difficult to propagate these high-order, good schemata once they are found, but, provided the individuals representing these schemata are not replaced by better individuals that represent incompatible schemata, they will be preserved and may eventually be able to propagate their schemata on to their offspring. Unfortunately, this proviso is not likely to be met by any but low-order deceptive problems. Even for deceptive problems of order five, the difficulty of propagating optimal schemata is

such that the suboptimal schemata tend to crowd out the optimum ones.

Perhaps the ultimate GA for tackling deceptive problems is Goldberg's messy GA (mGA) (Goldberg *et al* 1991). Whereas in more traditional GAs the manipulation of building blocks is implicit, mGAs explicitly manipulate the building blocks. This is accomplished by using variable-length strings that may be underspecified or overspecified; that is, some bit positions may not be defined, and some positions may have conflicting specifications. This is what makes mGAs messy.

These strings constitute the building blocks. They consist of a set of position–value pairs. Overspecified strings are evaluated by a simple conflict resolution strategy such as first-come-first-served rules. Thus, ((1 0) (2 1) (1 1) (3 0)) would be interpreted as 010, ignoring the third pair, since the first position has already been defined. Underspecified strings are interpreted by filling in the missing values using a competitive template, a locally optimal structure. For example, if the locally optimal structure, found by testing one bit at a time, is 111, then the string ((1 0) (3 0)) would be interpreted by filling in the value for the (missing) second position with the value of the second position in the template. The resulting 010 string would then be evaluated.

mGAs have an outer and an inner loop. The inner loop consists of three phases: the initialization, primordial, and juxtaposition phases. In the initialization phase all substrings of length k are created and evaluated, i.e. all combinations of strings with k defining positions (where k is an estimate of the highest order of deception in the problem). As was explained above the missing values are filled in using the competitive template. (As will be explained below, the template for the k level of the outer loop is the solution found at the $k-1$ level.)

In the primordial phase, selection is applied to the population of individuals produced during the initialization phase without any operators. Thus the substrings that have poor evaluations are eliminated and those with good evaluations have multiple copies in the resulting population.

In the juxtapositional phase selection in conjunction with cut and splice operators is used to evolve improved variations. Again, the competitive template is used for filling in missing values, and the first-come-first-served rule is used for handling overspecified strings created by the splice operator. The cut and splice operators act much like one-point crossover in a traditional GA, keeping in mind that the strings are of variable length and may be underspecified or overspecified.

The outer loop is over levels. It starts at the level of $k = 1$, and continues through each level until it reaches a user-specified stopping criterion. At each level, the solution found at the previous level is used as the competitive template.

One of the limitations of mGAs as originally conceived is that the initialization phase becomes extremely expensive as the mGA progresses up the levels. A new variant of the mGA speeds up the process by eliminating the need to process all the variants in the initialization stage (Goldberg *et al* 1993). The initialization and primordial phases of the original mGA are replaced by a 'probabilistically complete initialization' procedure. This procedure is divided into several steps. During the first step strings of nearly length L are evaluated (using the template to fill in the missing values). Then selection is applied to these strings without any operators (much as was done in the primordial phase of the original mGA, but for only a few generations). Then the algorithm enters a filtering step where some of the genes in the strings are deleted, and the shortened strings are evaluated using the competitive template. Then selection is applied again. This process is repeated until the resulting strings are of length k. Then the mGA goes into the juxtaposition stage like the original mGA. By replacing the original initialization and primordial stages with stepwise filtering and selection, the number of evaluations required is drastically reduced for problems of significant size. (Goldberg *et al* (1993) provide analytical methods for determining the population and filtering reduction constants.) This new version of the mGA is very effective at solving loosely linked deceptive problems, i.e. those problems where the defining positions of the deceptive segments are spread out along the bitstring.

mGAs were designed to operate according to the SBBH, and deceptive problems illustrate that there are problems where being able to manipulate building blocks can provide an added value over CCV. It still is an open question, however, as to how representative deceptive problems are of the types of real-world problem that GAs might encounter. No doubt, many difficult real-world problems have deceptive or misleading elements in them. If they did not, they could be easily solved by local search methods. However it does not necessarily follow that such problems can be solved by a GA that is good at solving deceptive problems. The SBBH assumes that the misleading building blocks will exist in the initial population, that they can be identified early in the search before they are lost, and that the problem can be solved incrementally by combining these building blocks, but perhaps the building blocks that have

misleading alternatives have little meaning until late in the search and so cannot be expected to survive in the population.

Even if the SBBH turns out not to be as useful an hypothesis as originally supposed, the increased propagation capabilities of pairwise mating may give a GA (using pairwise mating) an advantage over a poolwise CCV algorithm. To see why this is the case it is useful to define the prototypical individual for a given population: for each locus we assign a one or a zero depending upon which value is most frequent in the population (randomly assigning a value if they are equally frequent). Suppose the population contains some maverick individual that is quite far from the prototypical individual although it is near the optimum (as measured by Hamming distance) but is of only average fitness. Since an algorithm using a poolwise method of producing offspring will tend to produce individuals that are near the prototypical individual, such an algorithm is unlikely to explore the region around the maverick individual. On the other hand, a GA using pairwise mating is more likely to explore the region around the maverick individual, and so more likely to discover the optimum. Ironically, pairwise mating is, in this respect, more mutation-like than poolwise mating. While pairwise mating retains the benefits of CCV, it less subject to the majoritarian tendencies of poolwise mating.

B1.2.4 Representation

Although GAs typically use a bitstring representation, GAs are not restricted to bitstrings. A number of
C1.3 early proponents of GAs developed GAs that use other representations, such as *real-valued parameters*
C1.4 (Davis 1991, Janikow and Michalewicz 1991, Wright 1991), *permutations* (Davis 1985, Goldberg and
C1.6 Lingle 1985, Grefenstette *et al* 1985), and *treelike hierarchies* (Antonisse and Keller 1987). Koza's
B1.5.1 *genetic programming* (GP) paradigm (Koza 1992) is a GA-based method for evolving programs, where the data structures are LISP S-expressions, and crossover creates new LISP S-expressions (offspring) by exchanging subtrees from the two parents.

G9.5 In the case of combinatorial problems such as the *traveling salesman problem* (TSP), a number of order-based or sequencing crossover operators have been proposed. The choice of operator will depend upon one's goal. If the goal is to solve a TSP, then preserving adjacency information will be the priority, which suggests a crossover operator that operates on common edges (links between cities shared by the
F1.5 two parents) (Whitley *et al* 1989). On the other hand, if the goal is to solve a *scheduling* problem, then preserving relative order is likely to be the priority, which suggests an order preserving crossover operator. Syswerda's order crossover operator (Syswerda 1991), for example, chooses several positions at random in the first parent, and then produces a child so that the relative order of the chosen elements in the first parent is imposed upon the second parent.

Even if binary strings are used, there is still a choice to be made as to which binary coding scheme to use for numerical parameters. Empirical studies have usually found that Gray code is superior to the standard power-of-two binary coding (Caruana and Schaffer 1988), at least for the commonly used test problems. One reason is that the latter introduces Hamming cliffs—two numerically adjacent values may have bit representations that are many bits apart (up to $L - 1$). This will be a problem if there is some degree of gradualness in the function, i.e. small changes in the variables usually correspond to small changes in the function. This is often the case for functions with numeric parameters.

As an example, consider a five-bit parameter, with a range from 0 to 31. If it is encoded using the standard binary coding, then 15 is encoded as 01111, whereas 16 is encoded as 10000. In order to move from 15 to 16, all five bits need to be changed. On the other hand, using Gray coding, 15 would be represented as 01000 and 16 as 11000, differing only by 1 bit.

When choosing an alternative representation, it is critical that a crossover operator be chosen that is appropriate for the representation. For example, if real-valued parameters are used, then a possible crossover operator is one that for each parameter uses the parameter values of the two parents to define an interval from which a new parameter is chosen (Eshelman and Schaffer 1993). As the GA makes progress it will narrow the range over which it searches for new parameter values.

If, for the chosen representation and crossover operator, the building blocks are unlikely to be instantiated independently of each other in the population, then a GA may not be appropriate. This problem has plagued finding crossover operators that are good for solving TSPs. The natural building blocks, it would seem, are subtours. However, what counts as a good subtour will almost always depend upon what the other subtours are. In other words, two good, but suboptimal solutions to a TSP may not

have many subtours (other than very short ones) that are compatible with each other so that they can be spliced together to form a better solution. This hurdle is not unique to combinatorial problems.

Given the importance of the representation, a number of researches have suggested methods for allowing the GA to adapt its own coding. We noted earlier that Holland proposed the inversion operator for rearranging the loci in the string. Another approach to adapting the representation is Shaefer's ARGOT system (Shaefer 1987). ARGOT contains an explicit parameterized representation of the mappings from bitstrings to real numbers and heuristics for triggering increases and decreases in resolution and for shifts in the ranges of these mappings. A similar idea is employed by Schraudolph and Belew (1992) who provide a heuristic for increasing the resolution triggered when the population begins to converge. Mathias and Whitley (1994) have proposed what they call delta coding. When the population converges, the numeric representation is remapped so that the parameter ranges are centered around the best value found so far, and the algorithm is restarted. There are also heuristics for narrowing or extending the range.

There are also GAs with mechanisms for dynamically adapting the rate at which GA operators are used or which operator is used. Davis, who has developed a number of nontraditional operators, proposed a mechanism for adapting the rate at which these operators are applied based on the past success of these operators during a run of the algorithm (Davis 1987).

B1.2.5 Parallel genetic algorithms

All evolutionary algorithms, because they maintain a population of solutions, are naturally parallelizable. However, because GAs use crossover, which is a way of sharing information, there are two other variations that are unique to GAs (Gordon and Whitley 1993). The first, most straightforward, method is to simply have one global population with multiple processors for evaluating individual solutions. The second method, often referred to as the *island model* (alternatively, the migration or coarse-grain model), maintains C6.3 separate subpopulations. Selection and crossover take place in each subpopulation in isolation from the other subpopulations. Every so often an individual from one of the subpopulations is allowed to migrate to another subpopulation. This way information is shared among subpopulations.

The third method, often referred to as the *neighborhood model* (alternatively, the diffusion or fine-grain C6.4 model), maintains overlapping neighborhoods. The neighborhood for which selection (for reproduction and replacement) applies is restricted to a region local to each individual. What counts as a neighborhood will depend upon the neighborhood topology used. For example, if the population is arranged upon some type of spherical structure, individuals might be allowed to mate with (and forced to compete with) neighbors within a certain radius.

B1.2.6 Conclusion

Although the above discussion has been in the context of GAs as potential function optimizers, it should be pointed out that Holland's initial GA work was in the broader context of exploring GAs as adaptive systems (De Jong 1993). GAs were designed to be a simulation of evolution, not to solve problems. Of course, evolution has come up with some wonderful designs, but one must not lose sight of the fact that evolution is an opportunistic process operating in an environment that is continuously changing. Simon has described evolution as a process of searching where there is no goal (Simon 1983). This is not to question the usefulness of GAs as function optimizers, but only to emphasize that the perspective of function optimization is somewhat different from that of adaptation, and that the requirements of the corresponding algorithms will be somewhat different.

References

Antonisse H J and Keller K S 1987 Genetic operators for high-level knowledge representations *Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erbaum) pp 69–76

Bäck T, Hoffmeister F and Schwefel H 1991 A survey of evolution strategies *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 2–9

Baluja S 1995 *An Empirical Comparison of Seven Iterative and Evolutionary Function Optimization Heuristics* Carnegie Mellon University School of Computer Science Technical Report CMU-CS-95-193

Caruana R A and Schaffer J D 1988 Representation and hidden bias: Gray vs. binary coding for genetic algorithms *Proc. 5th Int. Conf. on Machine Learning* (San Mateo, CA: Morgan Kaufmann) pp 153–61

Davis L 1985 Applying adaptive algorithms to epistatic domains *Proc. Int. Joint Conference on Artificial Intelligence* pp 162–4

——1987 Adaptive operator probabilities in genetic algorithms *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 61–9

——1991 Hybridization and numerical representation *The Handbook of Genetic Algorithms* ed L Davis (New York: Van Nostrand Reinhold) pp 61–71

De Jong K 1975 *An Analysis of the Behavior of a Class of Genetic Adaptive Systems* Doctoral Thesis, Department of Computer and Communication Sciences, University of Michigan

——1993 Genetic algorithms are not function optimizers *Foundations of Genetic Algorithms 2* ed D Whitley (San Mateo, CA: Morgan Kaufmann) pp 5–17

De Jong K and Sarma J 1993 Generation gaps revisited *Foundations of Genetic Algorithms 2* ed D Whitley (San Mateo, CA: Morgan Kaufmann) pp 19–28

Eshelman L J 1991 The CHC adaptive search algorithm: how to have safe search when engaging in nontraditional genetic recombination *Foundations of Genetic Algorithms* ed G J E Rawlins (San Mateo, CA: Morgan Kaufmann) pp 265–83

Eshelman L J and Schaffer J D 1993 Real-coded genetic algorithms and interval schemata *Foundations of Genetic Algorithms 2* ed D Whitley (San Mateo, CA: Morgan Kaufmann) pp 187–202

——1995 Productive recombination and propagating and preserving schemata *Foundations of Genetic Algorithms 3* ed D Whitley (San Mateo, CA: Morgan Kaufmann) pp 299–313

Goldberg D E 1987 Simple genetic algorithms and the minimal, deceptive problem *Genetic Algorithms and Simulated Annealing* ed L Davis (San Mateo, CA: Morgan Kaufmann) pp 74–88

——1989 *Genetic Algorithms in Search, Optimization, and Machine Learning* (Reading, MA: Addison-Wesley)

Goldberg D E and Deb K 1991 A comparative analysis of selection schemes used in genetic algorithms *Foundations of Genetic Algorithms* ed G J E Rawlins (San Mateo, CA: Morgan Kaufmann) pp 69–93

Goldberg D E, Deb K, Kargupta H and Harik G 1993 Rapid, accurate optimization of difficult problems using fast messy genetic algorithms *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 56–64

Goldberg D E, Deb K and Korb B 1991 Don't worry, be messy *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 24–30

Goldberg D E and Lingle R L 1985 Alleles, loci, and the traveling salesman problem *Proc. 1st Int. Conf. on Genetic Algorithms (Pittsburgh, PA, 1985)* ed J J Grefenstette (Hillsdale, NJ: Erbaum) pp 154–9

Gordon V S and Whitley 1993 Serial and parallel genetic algorithms and function optimizers *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann) pp 177–83

Grefenstette J J 1993 Deception considered harmful *Foundations of Genetic Algorithms 2* ed D Whitley (San Mateo, CA: Morgan Kaufmann) pp 75–91

Grefenstette J J, Gopal R, Rosmaita B J and Van Gucht D 1985 Genetic algorithms for the traveling salesman problem *Proc. 1st Int. Conf. on Genetic Algorithms (Pittsburgh, PA, 1985)* ed J J Grefenstette (Hillsdale, NJ: Erbaum) pp 160–8

Holland J H 1975 *Adaptation in Natural and Artificial Systems* (Ann Arbor, MI: University of Michigan Press)

Janikow C Z and Michalewicz Z 1991 An experimental comparison of binary and floating point representations in genetic algorithms *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 31–6

Koza J 1992 *Genetic Programming: on the Programming of Computers by Means of Natural Selection and Genetics* (Cambridge, MA: MIT Press)

Liepins G E and Vose M D 1991 Representational issues in genetic optimization *J. Exp. Theor. AI* **2** 101–15

Mathias K E and Whitley L D 1994 Changing representations during search: a comparative study of delta coding *Evolutionary Comput.* **2**

Mühlenbein H and Schlierkamp-Voosen 1993 The science of breeding and its application to the breeder genetic algorithm *Evolutionary Comput.* **1**

Radcliffe N J 1991 Forma analysis and random respectful recombination *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 222–9

Schaffer J D, Eshelman L J and Offutt D 1991 Spurious correlations and premature convergence in genetic algorithms *Foundations of Genetic Algorithms* ed G J E Rawlins (San Mateo, CA: Morgan Kaufmann) pp 102–12

Schraudolph N N and Belew R K 1992 Dynamic parameter encoding for genetic algorithms *Machine Learning* **9** 9–21

Shaefer C G 1987 The ARGOT strategy: adaptive representation genetic optimizer technique *Genetic Algorithms and Their Applications: Proc. 2nd Int. Conf. on Genetic Algorithms (Cambridge, MA, 1987)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum) pp 50–8

Simon H A 1983 *Reason in Human Affairs* (Stanford, CA: Stanford University Press)

Spears W M and De Jong K A 1991 On the virtues of parameterized uniform crossover *Proc. 4th Int. Conf. on Genetic Algorithms (San Diego, CA, 1991)* ed R K Belew and L B Booker (San Mateo, CA: Morgan Kaufmann) pp 230–6

Syswerda G 1989 Uniform crossover in genetic algorithms *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 2–9

——1991 Schedule optimization using genetic algorithms *Handbook of Genetic Algorithms* ed L Davis (New York: Van Nostrand Reinhold) pp 332–49

——1993 Simulated crossover in genetic algorithms *Foundations of Genetic Algorithms 2* ed D Whitley (San Mateo, CA: Morgan Kaufmann) pp 239–55

Whitley D 1989 The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 116–21

Whitley D, Starkweather T and Fuquay D 1989 Scheduling problems and traveling salesmen: the genetic edge recombination operator *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 116–21

Wright A 1991 Genetic algorithms for real parameter optimization *Foundations of Genetic Algorithms* ed G J E Rawlins (San Mateo, CA: Morgan Kaufmann) pp 205–18

B1.3 Evolution strategies

Günter Rudolph

Abstract

This section provides a description of evolution strategies (ESs) as a special instance of evolutionary algorithms. After the presentation of the archetype of ESs, accompanied with some historical remarks, contemporary ESs (the standard instances) are described in detail. Finally, the design and potential fields of application of nested ESs are discussed.

B1.3.1 The archetype of evolution strategies

Minimizing the total drag of three-dimensional slender bodies in a turbulent flow was, and still is, a general goal of research in institutes of hydrodynamics. Three students (Peter Bienert, Ingo Rechenberg, and Hans-Paul Schwefel) met each other at such an institute, the Hermann Föttinger Institute of the Technical University of Berlin, in 1964. Since they were amazed not only by aerodynamics, but also by cybernetics, they hit upon the idea to solve the analytically (and at that time also numerically) intractable form design problem with the help of some kind of robot. The robot should perform the necessary experiments by iteratively manipulating a flexible model positioned at the outlet of a wind tunnel. An *experimentum crucis* was set up with a two-dimensional foldable plate. The iterative search strategy—first performed by hand, a robot was developed later on by Peter Bienert—was expected to end up with a flat plate: the form with minimal drag. But it did not, since a one-variable-at-a-time as well as a discrete gradient-type strategy always got stuck in a local minimum: an S-shaped folding of the plate. Switching to small random changes that were only accepted in the case of improvements—an idea of Ingo Rechenberg—brought the breakthrough, which was reported at the joint annual meeting of WGLR and DGRR in Berlin, 1964 (Rechenberg 1965). The interpretation of binomially distributed changes as mutations and of the decision to step back or not as selection (on 12 June 1964) was the seed for all further developments leading to evolution strategies (ESs) as they are known today. So much about the birth of the *ES*.

It should be mentioned that the domain of the decision variables was not fixed or even restricted to real variables at that time. For example, the experimental optimization of the shape of a supersonic two-phase nozzle by means of mutation and selection required discrete variables and mutations (Klockgether and Schwefel 1970) whereas first numerical experiments with the early ES on a Zuse Z 23 computer (Schwefel 1965) employed discrete mutations of real variables. The apparent fixation of ESs to Euclidean search spaces nowadays is probably due to the fact that Rechenberg (1973) succeeded in analyzing the simple version in Euclidean space with continuous mutation for several test problems.

Within this setting the archetype of ESs takes the following form. An individual a consisting of an element $\boldsymbol{X} \in \mathbb{R}^n$ is mutated by adding a normally distributed random vector $\boldsymbol{Z} \sim N(0, \mathbf{I}_n)$ that is multiplied by a scalar $\sigma > 0$ ($\mathbf{I}_n$ denotes the unit matrix with rank n). The new point is accepted if it is better than or equal to the old one, otherwise the old point passes to the next iteration. The selection decision is based on a simple comparison of the objective function values of the old and the new point. Assuming that the objective function $f : \mathbb{R}^n \to \mathbb{R}$ is to be minimized, the simple ES, starting at some point $\boldsymbol{X}_0 \in \mathbb{R}^n$, is determined by the following iterative scheme:

$$\boldsymbol{X}_{t+1} = \begin{cases} \boldsymbol{X}_t + \sigma_t \boldsymbol{Z}_t & \text{if} \quad f(\boldsymbol{X}_t + \sigma_t \boldsymbol{Z}_t) \leq f(\boldsymbol{X}_t) \\ \boldsymbol{X}_t & \text{otherwise} \end{cases} \tag{B1.3.1}$$

where $t \in \mathbb{N}_0$ denotes the iteration counter and where $(Z_t : t \geq 0)$ is a sequence of independent and identically distributed standard normal random vectors.

The general algorithmic scheme (B1.3.1) was not a novelty: Schwefel (1995, pp 94–5), presents a survey of forerunners and related versions of (B1.3.1) since the late 1950s. Most methods differed in the mechanism to adjust the parameter σ_t, that is used to control the strength of the mutations (i.e. the length of the mutation steps in n-dimensional space). Rechenberg's solution to control parameter σ_t is known as the 1/5 success rule: Increase σ_t if the relative frequency of successful mutations over some period in the past is larger than 1/5, otherwise decrease σ_t. Schwefel (1995, p 112), proposed the following implementation. Let $t \in \mathbb{N}$ be the generation (or mutation) counter and assume that $t \geq 10n$.

(i) If $t \bmod n = 0$ then determine the number s of successful mutations that have occurred during the steps $t - 10n$ to $t - 1$.
(ii) If $s < 2n$ then multiply the step lengths by the factor 0.85.
(iii) If $s > 2n$ then divide the step lengths by the factor 0.85.

First ideas to extend the simple ES (B1.3.1) can be found in the book by Rechenberg (1973, pp 78–86). The population consists of $\mu > 1$ parents. Two parents are selected at random and recombined by multipoint crossover and the resulting individual is finally mutated. The offspring is added to the population. The selection operation chooses the μ best individuals out of the $\mu + 1$ in total to serve as parents of the next iteration. Since the search space was binary, this ES was exactly the same evolutionary algorithm as
C2.7.1 became known later under the term *steady-state genetic algorithm*. The usage of this algorithmic scheme for Euclidean search spaces poses the problem of how to control the step length control parameter σ_t. Therefore, the 'steady-state' ES is no longer in use.

B1.3.2 Contemporary evolution strategies

The general algorithmic frame of contemporary ESs is easily presented by the symbolic notation introduced in Schwefel (1977). The abbreviation $(\mu + \lambda)$ ES denotes an ES that generates λ offspring from μ parents and selects the μ best individuals from the $\mu + \lambda$ individuals (parents and offspring) in total. This notation can be used to express the simple ES by $(1 + 1)$ ES and the 'steady-state' ES by $(\mu + 1)$ ES. Since the latter is not in use it is convention that the abbreviation $(\mu + \lambda)$ ES always refers to an ES parametrized according to the relation $1 \leq \mu \leq \lambda < \infty$.

The abbreviation (μ, λ) ES denotes an ES that generates λ offspring from μ parents but selects the μ best individuals only from the λ offspring. As a consequence, λ must be necessarily at least as large as μ. However, since the parameter setting $\mu = \lambda$ represents nothing more than a random walk, it is convention that the abbreviation (μ, λ) ES always refers to an ES parametrized according to the relation $1 \leq \mu < \lambda < \infty$.

Apart from the population concept contemporary ESs differ from early ESs in that an individual consists of an element $x \in \mathbb{R}^n$ of the search space plus several individual parameters controlling the individual mutation distribution. Usually, mutations are distributed according to a multivariate normal distribution with zero mean and some covariance matrix $\mathbf{C}$ that is symmetric and positive definite. Unless matrix $\mathbf{C}$ is a diagonal matrix, the mutations in each coordinate direction are *correlated* (Schwefel 1995, p 240). It was shown in Rudolph (1992) that a matrix is symmetric and positive definite if and only if it is decomposable via $\mathbf{C} = (\mathbf{ST})'\mathbf{ST}$ where $\mathbf{S}$ is a diagonal matrix with positive diagonal entries and

$$\mathbf{T} = \prod_{i=1}^{n-1} \prod_{j=i+1}^{n} \mathbf{R}_{ij}(\omega_k) \tag{B1.3.2}$$

is an orthogonal matrix built by a product of $n(n-1)/2$ elementary rotation matrices $\mathbf{R}_{ij}$ with angles $\omega_k \in (0, 2\pi]$. An elementary rotation matrix $\mathbf{R}_{ij}(\omega)$ is a unit matrix where four specific entries are replaced by $r_{ii} = r_{jj} = \cos\omega$ and $r_{ij} = -r_{ji} = -\sin\omega$.

As a consequence, $n(n-1)/2$ angles and n scaling parameters are sufficient to generate arbitrary correlated normal random vectors with zero mean and covariance matrix $\mathbf{C} = (\mathbf{ST})'\mathbf{ST}$ via $Z = \mathbf{T}'\mathbf{S}'N$, where N is a standard normal random vector (since matrix multiplication is associative, random vector Z can be created in $O(n^2)$ time by multiplication from right to left).

There remains, however, the question of how to choose and adjust these individual strategy parameters. The idea that a population-based ES could be able to adapt σ_t individually by including these parameters

in the mutation–selection process came up early (Rechenberg 1973, pp 132–7). Although first experiments with the $(\mu + 1)$ ES provided evidence that this approach works in principle, the first really successful implementation of the idea of *self-adaptation* was presented by Schwefel (1977) and it is based on the observation that a surplus of offspring (i.e. $\lambda > \mu$) is a good advice to establish self-adaptation of individual parameters. C7.1

To start with a simple case let $\mathbf{C} = \sigma^2 \mathbf{I}_n$. Thus, the only parameter to be self-adapted for each individual is the step length control parameter σ. For this purpose let the the genome of each individual be represented by the tuple $(\boldsymbol{X}, \sigma) \in \mathbb{R}^n \times \mathbb{R}_+$, that undergoes the genetic operators. Now mutation is a two-stage operation:

$$\sigma_{t+1} = \sigma_t \exp(\tau Z_\tau)$$
$$\boldsymbol{X}_{t+1} = \boldsymbol{X}_t + \sigma_{t+1} \boldsymbol{Z}$$

where $\tau = n^{-1/2}$ and Z_τ is a standard normal random variable whereas $\boldsymbol{Z}$ is a standard normal random vector. This scheme can be extended to the general case with $n\,(n+1)/2$ parameters.

(i) Let $\omega \in (0, 2\pi]^{n(n-1)/2}$ denote the angles that are necessary to build the orthogonal rotation matrix $\mathbf{T}(\omega)$ via (B1.3.2). The mutated angles $\omega_{t+1}^{(i)}$ are obtained by

$$\omega_{t+1}^{(i)} = (\omega_t^{(i)} + \varphi Z_\omega^{(i)}) \bmod 2\pi$$

where $\varphi > 0$ and the independent random numbers $Z_\omega^{(i)}$ with $i = 1, \ldots, n\,(n-1)/2$ are standard normally distributed.

(ii) Let $\sigma \in \mathbb{R}_+^n$ denote the standard deviations that are represented by the diagonal matrix $\mathbf{S}(\sigma) = \operatorname{diag}(\sigma^{(1)}, \ldots, \sigma^{(n)})$. The mutated standard deviations are obtained as follows. Draw a standard normally distributed random number Z_τ. For each $i = 1, \ldots, n$ set

$$\sigma_{t+1}^{(i)} = \sigma_t^{(i)} \exp(\tau Z_\tau + \eta Z_\sigma^{(i)})$$

where $(\tau, \eta) \in \mathbb{R}_+^2$ and the independent random numbers $Z_\sigma^{(i)}$ are standard normally distributed. Note that Z_τ is drawn only once.

(iii) Let $\boldsymbol{X} \in \mathbb{R}^n$ be the object variables and $\boldsymbol{Z}$ be a standard normal random vector. The mutated object variable vector is given by

$$\boldsymbol{X}_{t+1} = \boldsymbol{X}_t + \mathbf{T}(\omega_{t+1})\mathbf{S}(\sigma_{t+1})\boldsymbol{Z}.$$

According to Schwefel (1995) a good heuristic for the choice of the constants appearing in the above mutation operation is

$$(\varphi, \tau, \eta) = (5\pi/180,\ (2n)^{-1/2},\ (4n)^{-1/4})$$

but recent extensive simulation studies (Kursawe 1996) revealed that the above recommendation is not the best choice—especially in the case of multimodal objective functions it seems to be better to use weak selection pressure (μ/λ not too small) and a parametrization obeying the relation $\tau > \eta$. As a consequence, a final recommendation cannot be given here, yet.

As soon as $\mu > 1$, the decision variables as well as the internal strategy parameters can be recombined with usual recombination operators. Notice that there is no reason to employ the same recombination operator for the angles, standard deviations, and object variables. For example, one could apply *intermediate recombination* to the angles as well as standard deviations and *uniform crossover* to the object variables. With this choice recombination of two parents works as follows. Choose two parents $(\boldsymbol{X}, \sigma, \omega)$ and $(\boldsymbol{X}', \sigma', \omega')$ at random. Then the preliminary offspring resulting from the recombination process is C3.3

$$\left(\mathbf{U}\boldsymbol{X} + (\mathbf{I} - \mathbf{U})\boldsymbol{X}',\ \frac{\sigma + \sigma'}{2},\ \frac{(\omega + \omega') \bmod 4\pi}{2}\right)$$

where $\mathbf{I}$ is the unit matrix and $\mathbf{U}$ is a random diagonal matrix whose diagonal entries are either zero or one with the same probability. Note that the angles must be adjusted to the interval $(0, 2\pi]$.

After these preparations a sketch of a contemporary ES can be presented:

Generate μ initial parents of the type $(\boldsymbol{X}, \sigma, \omega)$ and determine their objective function values $f(\boldsymbol{X})$.
`repeat`
  `do` λ `times:`
    Choose $\rho \geq 2$ parents at random.
    Recombine their angles, standard deviations, and object variables.
    Mutate the angles, standard deviations, and object variables of the preliminary offspring obtained via recombination.
    Determine the offspring's objective function value.
    Put the offspring into the offspring population.
  `end do`
  Select the μ best individuals either from the offspring population
  or from the union of the parent and offspring population.
  The selected individuals represent the new parents.
`until` some stopping criterion is satisfied.

It should be noted that there are other proposals to adapt σ_t. In the case of a $(1, \lambda)$ ES with $\lambda = 3k$ and $k \in \mathbb{N}$, Rechenberg (1994), p 47, devised the following rule: Generate k offspring with σ_t, k offspring with $c\,\sigma_t$ and k offspring with σ_t / c for some $c > 0$ ($c = 1.3$ is recommended for $n \leq 100$, for larger n the constant c should decrease).

Further proposals, that are however still in an experimental state, try to derandomize the adaptation process by exploiting information gathered in preceding iterations (Ostermeier *et al* 1995). This approach is related to (deterministic) variable metric (or quasi-Newton) methods, where the Hessian matrix is approximated iteratively by certain update rules. The inverse of the Hessian matrix is in fact the optimal choice for the covariance matrix $\mathbf{C}$. A large variety of update rules is given by the *Oren–Luenberger class* (Oren and Luenberger 1974) and it might be useful to construct probabilistic versions of these update rules, but it should be kept in mind that ESs are designed to tackle difficult nonconvex problems and not convex ones: the usage of such techniques increases the risk that ESs will be attracted by local optima.

Other ideas that have not yet achieved a standard include the introduction of an additional age parameter κ for individuals in order to have intermediate forms of selection between the $(\mu + \lambda)$ ES with $\kappa = \infty$ and the (μ, λ) ES with $\kappa = 1$ (Schwefel and Rudolph 1995), as well as the huge variety of ESs whose population possesses a spatial structure. Since the latter is important for parallel implementations and applies to other evolutionary algorithms as well the description is omitted here.

B1.3.3 Nested evolution strategies

The shorthand notation $(\mu \dot{+} \lambda)$ ES was extended by Rechenberg (1978) to the expression

$$[\,\mu' \dot{+} \lambda'\,(\mu \dot{+} \lambda)^{\gamma}\,]^{\gamma'}\ \text{ES}$$

with the following meaning. There are μ' populations of μ parents. These are used to generate (e.g. by merging) λ' initial populations of μ individuals each. For each of these λ' populations a $(\mu \dot{+} \lambda)$ ES is run for γ generations. The criterion to rank the λ' populations after termination might be the average fitness of the individuals in each population. This scheme is repeated γ' times. The obvious generalization to higher levels of nesting is described by Rechenberg (1994), where it is also attempted to develop a shorthand notation to specify the parametrization completely.

This nesting technique is of course not limited to ESs: other evolutionary algorithms and even mixtures of them can be used instead. In fact, the somewhat artificial distinction between ESs, genetic algorithms, and evolutionary programs becomes more and more blurred when higher concepts enter the scene. Finally, some fields of application of nested evolutionary algorithms will be described briefly.

Alternative method to control internal parameters. Herdy (1992) used λ' subpopulations, each of them possessing its own different and fixed step size σ. Thus, there is no step size control at the level of individuals. After γ generations the improvements (in terms of fitness) achieved by each subpopulation is compared to each other and the best μ' subpopulations are selected. Then the process repeats with slightly modified values of σ. Since subpopulations with a near-optimal step size will achieve larger

improvements, they will be selected (i.e. better step sizes will survive), resulting in an alternative method to control the step size.

Mixed-integer optimization. Lohmann (1992) considered optimization problems in which the decision variables are partially discrete and partially continuous. The nested approach worked as follows. The ESs in the inner loop were optimizing over the continuous variables while the discrete variables were held fixed. After termination of the inner loop, the evolutionary algorithm in the outer loop compared the fitness values achieved in the subpopulations, selected the best ones, mutated the discrete variables and passed them as fixed parameters to the subpopulations in the inner loop.

It should be noted that this approach to mixed-integer optimization may cause some problems. In essence, a Gauß–Seidel-like optimization strategy is realized, because the search alternates between the subspace of discrete variables and the subspace of continuous variables. Such a strategy must fail whenever simultaneous changes in discrete *and* continuous variables are necessary to achieve further improvements.

Minimax optimization. Sebald and Schlenzig (1994) used nested optimization to tackle minimax problems of the type

$$\min_{x \in X}\{\max_{y \in Y}\{f(x, y)\}\}$$

where $X \subseteq \mathbb{R}^n$ and $Y \subseteq \mathbb{R}^m$. Equivalently, one may state the problem as follows:

$$\min\{g(x) : x \in X\} \qquad \text{where} \quad g(x) = \max\{f(x, y) : y \in Y\}.$$

The evolutionary algorithm in the inner loop maximizes $f(x, y)$ with fixed parameters x, while the outer loop is responsible to minimize $g(x)$ over the set X.

Other applications of this technique are imaginable. An additional aspect touches the evident degree of independence of executing the evolutionary algorithms in the inner loop. As a consequence, nested evolutionary algorithms are well suited for parallel computers.

References

Herdy M 1992 Reproductive isolation as strategy parameter in hierachically organized evolution strategies *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 207–17

Klockgether J and Schwefel H-P 1970 Two-phase nozzle and hollow core jet experiments *Proc. 11th Symp. on Engineering Aspects of Magnetohydrodynamics* ed D Elliott (Pasadena, CA: California Institute of Technology) pp 141–8

Kursawe F 1996 Breeding evolution strategies—first results, talk presented at Dagstuhl lectures *Applications of Evolutionary Algorithms (March 1996)*

Lohmann R 1992 Structure evolution and incomplete induction *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 175–85

Oren S and Luenberger D 1974 Self scaling variable metric (SSVM) algorithms, Part II: criteria and sufficient conditions for scaling a class of algorithms *Management Sci.* **20** 845–62

Ostermeier A, Gawelczyk A and Hansen N 1995 A derandomized approach to self-adaptation of evolution strategies *Evolut. Comput.* **2** 369–80

Rechenberg I 1965 *Cybernetic solution path of an experimental problem* Library Translation 1122, Royal Aircraft Establishment, Farnborough, UK

——1973 *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution* (Stuttgart: Frommann-Holzboog)

——1978 Evolutionsstrategien *Simulationsmethoden in der Medizin und Biologie* ed B Schneider and U Ranft (Berlin: Springer) pp 83–114

——1994 *Evolutionsstrategie '94* (Stuttgart: Frommann-Holzboog)

Rudolph G 1992 On correlated mutations in evolution strategies *Parallel Problem Solving from Nature, 2 (Proc. 2nd Int. Conf. on Parallel Problem Solving from Nature, Brussels, 1992)* ed R Männer and B Manderick (Amsterdam: Elsevier) pp 105–14

Schwefel H-P 1965 *Kybernetische Evolution als Strategie der experimentellen Forschung in der Strömungstechnik* Diplomarbeit, Technical University of Berlin

——1977 *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie* (Basel: Birkhäuser)

——1995 *Evolution and Optimum Seeking* (New York: Wiley)

Schwefel H-P and Rudolph G 1995 Contemporary evolution strategies *Advances in Artificial Life* ed F Morana *et al* (Berlin: Springer) pp 893–907

Sebald A V and Schlenzig J 1994 Minimax design of neural net controllers for highly uncertain plants *IEEE Trans. Neural Networks* **NN-5** 73–82

B1.4 Evolutionary programming

V William Porto

Abstract

This section describes the basic concepts of evolutionary programming (EP) as originally introduced by Fogel, with extensions by numerous other researchers. EP is distinguished from other forms of evolutionary computation, such as genetic algorithms, in that it simulates evolution emphasizing the phenotypic relationship between parent and offspring, rather than the genetic relationship. Emphasis is placed on the use of one or more mutation operations which generate diversity among the population of solutions while maintaining a high degree of correlation between parent and offspring behavior. Recent efforts in the areas of pattern recognition, system identification, parameter optimization, and automatic control are presented.

B1.4.1 Introduction

Evolutionary programming (EP) is one of a class of paradigms for simulating evolution which utilizes the concepts of *Darwinian evolution* to iteratively generate increasingly appropriate solutions (organisms) A2.1 in light of a static or dynamically changing environment. This is in sharp contrast to earlier research into artificial intelligence research which largely centered on the search for simple heuristics. Instead of developing a (potentially) complex set of rules which were derived from human experts, EP evolves a set of solutions which exhibit optimal behavior with regard to an environment and desired payoff function. In a most general framework, EP may be considered an optimization technique wherein the algorithm iteratively optimizes behaviors, parameters, or other constructs. As in all optimization algorithms, it is important to note that the point of optimality is completely independent of the search algorithm, and is solely determined by the adaptive topography (i.e. response surface) (Atmar 1992).

In its standard form, the basic evolutionary program utilizes the four main components of all evolutionary computation (EC) algorithms: initialization, variation, evaluation (scoring), and selection. At the basis of this, as well as other EC algorithms, is the presumption that, at least in a statistical sense, learning is encoded phylogenically versus ontologically in each member of the population. 'Learning' is a byproduct of the evolutionary process as successful individuals are retained through stochastic trial and error. Variation (e.g. mutation) provides the means for moving solutions around on the search space, preventing entrapment in local minima. The evaluation function directly measures fitness, or equivalently the behavioral error, of each member in the population with regard to the environment. Finally, the selection process probabilistically culls suboptimal solutions from the population, providing an efficient method for searching the topography.

The basic EP algorithm starts with a population of trial solutions which are initialized by random, heuristic, or other appropriate means. The size of the population, μ, may range over a broadly distributed set, but is in general larger than one. Each of these trial solutions is evaluated with regard to the specified fitness function. After the creation of a population of initial solutions, each of the *parent* members is altered through application of a mutation process; in strict EP, recombination is not utilized. Each parent member i generates λ_i progeny which are replicated with a stochastic error mechanism (mutation). The fitness or behavioral error is assessed for all offspring solutions with the selection process performed by one of several general techniques including: (i) the best μ solutions are retained to become the parents for

C2.7.4, C2.3 the next generation (*elitist*), or (ii) μ of the best solutions are statistically retained (*tournament*), or (iii)
C2.2 *proportional-based selection*. In most applications, the size of the population remains constant, but there is no restriction in the general case. The process is halted when the solution reaches a predetermined quality, a specified number of iterations has been achieved, or some other criterion (e.g. sufficient convergence) stops the algorithm.

B1.2 EP differs philosophically from other evolutionary computational techniques such as *genetic algorithms* (GAs) in a crucial manner. EP is a top-down versus bottom-up approach to optimization. It is important to note that (according to neo-Darwinism) selection operates only on the phenotypic expressions of a genotype; the underlying coding of the phenotype is only affected indirectly. The realization that a sum of optimal parts rarely leads to an optimal overall solution is key to this philosophical difference. GAs rely on the identification, combination, and survival of 'good' building blocks (schemata) iteratively combining to form larger 'better' building blocks. In a GA, the coding structure (genotype) is of primary importance as it contains the set of optimal building blocks discovered through successive iterations. The
B2.5.3 *building block hypothesis* is an implicit assumption that the fitness is a separable function of the parts of the genome. This successively iterated local optimization process is different from EP, which is an entirely global approach to optimization. Solutions (or organisms) in an EP algorithm are judged solely on their fitness with respect to the given environment. No attempt is made to partition credit to individual
B1.3 components of the solutions. In EP (and in *evolution strategies* (ESs)), the variation operator allows for simultaneous modification of all variables at the same time. Fitness, described in terms of the behavior of each population member, is evaluated directly, and is the sole basis for survival of an individual in the population. Thus, a crossover operation designed to recombine building blocks is not utilized in the general forms of EP.

B1.4.2 History

A2.3.2 The *genesis* of EP was motivated by the desire to generate an alternative approach to artificial intelligence. Fogel (1962) conceived of using the simulation of evolution to develop artificial intelligence which did not rely on heuristics, but instead generated organisms of increasing intellect over time. Fogel (1964, Fogel *et al* 1966) made the observation that intelligent behavior requires the ability of an organism to make correct predictions within its environment, while being able to translate these predictions into a suitable response
C1.5 for a given goal. This early work focused on evolving *finite-state machines* (see the articles by Mealy (1955), and Moore (1957) for a discussion of these automata) which provided a most generic testbed for this approach. A finite-state machine (figure B1.4.1) is a mechanism which operates on a finite set (i.e. alphabet) of input symbols, possesses a finite number of internal states, and produces output symbols from a finite alphabet. As in all finite-state machines, the corresponding input–output symbol pairs and state transitions from every state define the specific behavior of the machine.

In a series of experiments (Fogel *et al* 1966), an environment was simulated by a sequence of symbols from a finite-length alphabet. The problem was defined as follows: evolve an algorithm which would operate on the sequence of symbols previously observed in a manner that would produce an output symbol which maximizes the benefit to the algorithm in light of the next symbol to appear in the environment, relative to a well-defined payoff function.

EP was originally defined by Fogel (1964) in the following manner. A population of *parent* finite-state machines, after initialization, is exposed to the sequence of symbols (i.e. environment) which have been observed up to the current time. As each input symbol is presented to each parent machine, the output symbol is observed (predicted) and compared to the next input symbol. A predefined payoff function provides a means for measuring the worth of each prediction. After the last prediction is made, some function of the sequence of payoff values is used to indicate the overall fitness of each machine. Offspring machines are created by randomly mutating each parent machine. As defined above, there are five possible resulting modes of random mutation for a finite-state machine. These are: (i) change an output symbol; (ii) change a state transition; (iii) add a state; (iv) delete an existing state; and (v) change the initial state. Other mutations were proposed but results of experiments with these mutations were not described by Fogel *et al* (1966). To prevent the possibility of creating null machines, the deletion of a state and the changing of the initial state were allowed only when a parent machine had more than one state.

Mutation operators are chosen with respect to a specified probability distribution which may be uniform, or another desired distribution. The number of mutation operations applied to each offspring is also determined with respect to a specified probability distribution function (e.g. Poisson) or may be

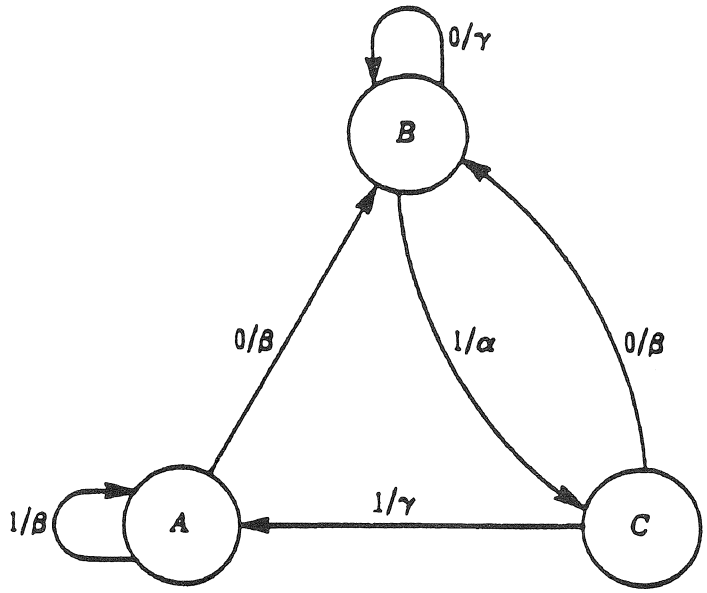


Figure B1.4.1. A simple finite-state machine diagram. Input symbols are shown to the left of the slash. Output symbols are to the right of the slash. The finite-state machine is presumed to start in state A.

fixed *a priori*. Each of the mutated offspring machines is evaluated over the existing environment (set of input–output symbol pairs) in the same manner as the parent machines.

After offspring have been created through application of the mutation operator(s) on the members of the parent population, the machines providing the greatest payoff with respect to the payoff function are retained to become parent members for the next generation. Typically, one offspring is created for each parent, and half of the total machines are retained in order to maintain a constant population size. The process is iterated until it is required to make an actual prediction of the next output symbol in the environment, which has yet to be encountered. This is analogous to the presentation of a naive exemplar to a previously trained neural network. Out of the entire population of machines, only the best machine, in terms of its overall worth, is chosen to generate the new output symbol. Fogel originally proposed selection of machines which score in the top half of the entire population, i.e. a nonregressive selection mechanism. Although discussed as a possibility to increase variance, the retention of lesser-quality machines was not incorporated in these early experiments.

Since the topography (response surface) is changed after each presentation of a symbol, the fitness of the evolved machines must change to reflect the payoff from the previous prediction. This prevents evolutionary stagnation as the adaptive topography is experiencing continuous change. As is evidenced in nature, the complexity of the representation is increased as the finite-state machines learn to recognize more subtle features in the experienced sequence of symbols.

B1.4.2.1 Early foundations

Fogel (see Fogel 1964, Fogel *et al* 1966) used EP on a series of successively more difficult prediction tasks. These experiments ranged from simple two-symbol cyclic sequences, eight-symbol cyclic sequences degraded by addition of noise, and sequences of symbols generated by other finite-state machines to nonstationary sequences and sequences taken from the article by Flood (1962).

In one example, the capability for predicting the 'primeness', i.e. whether or not a number is prime, was tested. A nonstationary sequence of symbols was generated by classifying each of the monotonically increasing set of integers as prime (with symbol 1) or nonprime (with symbol 0). The payoff function consisted of an all-or-none function where one point was provided for each correct prediction. No points or penalty were assessed for incorrect predictions. A small penalty term was added to maximize parsimony, through the subtraction of 0.01 multiplied by the number of states in the machine. This complexity penalty was added due to the limited memory available on the computers at that time. After presentation of 719 symbols, the iterative process was halted with the best machine possessing one state, with both

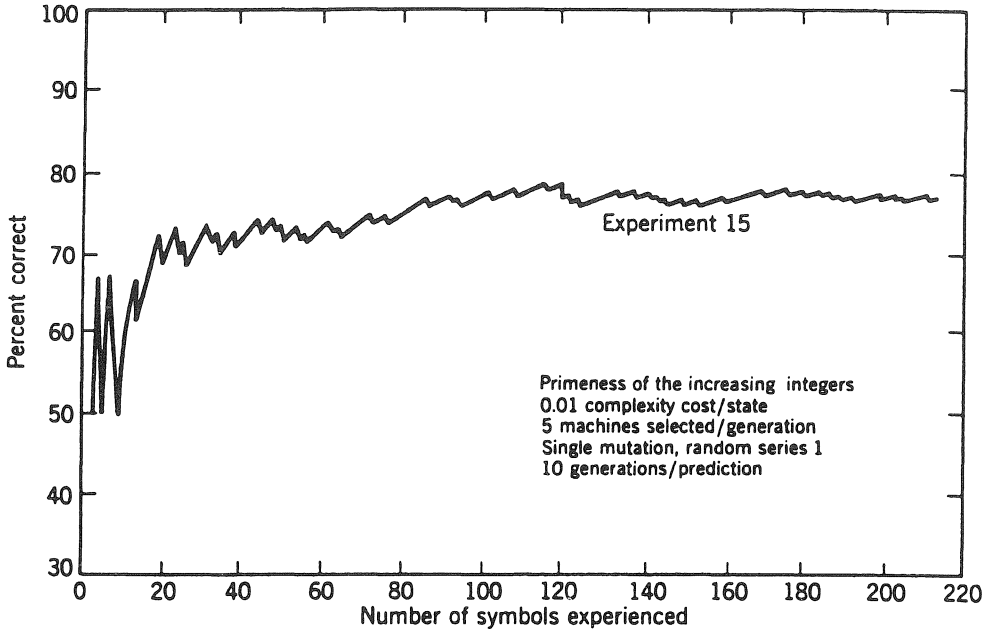


Figure B1.4.2. A plot showing the convergence of EP on finite-state machines evolved to predict primeness of numbers.

output symbols being zero. Figure B1.4.2 indicates the prediction score achieved in this nonstationary environment. Because prime numbers become increasingly infrequent (Burton 1976), the asymptotic worth of this machine, given the defined payoff function, approaches 100%.

After initial, albeit qualified, success with this experiment, the goal was altered to provide a greater payoff for correct prediction of a rare event. Correct prediction of a prime was worth one plus the number of nonprimes preceding it. For the first 150 symbols, 30 correct predictions were made (primes predicted as primes), 37 false positives (nonprimes predicted as primes), and five primes were missed. On predicting the 151st through 547th symbols there were 65 correct predictions of primes, and 67 false positives. Of the first 35 prime numbers, five were missed, but of the next 65 primes, none were missed. Fogel *et al* (1966) indicated that the machines demonstrated the capability to quickly recognize numbers which are divisible by two and three as being nonprime, and that some capability to recognize divisibility by five as being indicative of nonprimes was also evidenced. Thus, the machines generated evidence of learning a definition of primeness without prior knowledge of the explicit nature of a prime number, or any ability to explicitly divide.

Fogel and Burgin (1969) researched the use of EP in game theory. In a number of experiments, EP was consistently able to discover the globally optimal strategy in simple two-player, zero-sum games involving a small number of possible plays. This research also showed the ability of the technique to outperform human subjects in more complicated games. Several extensions were made to the simulations to address nonzero-sum games (e.g. pursuit evasion.) A three-dimensional model was constructed where EP was used to guide an *interceptor* towards a moving target. Since the target was, in most circumstances, allowed a greater degree of maneuverability, the success or failure of the interceptor was highly dependent upon the learned ability to predict the position of the target without *a priori* knowledge of the target's dynamics.

A different aspect of EP was researched by Walsh *et al* (1970) where EP was used for prediction as a precursor to automatic control. This research concentrated on decomposing a finite-state machine into submachines which could be executed in parallel to obtain the overall output of the evolved system. A primary goal of this research was to maximize parsimony in the evolving machines. In these experiments, finite-state machines containing seven and eight states were used as the generating function for three output symbols. The performance of three human subjects was compared to the evolved models when predicting the next symbol in the respective environments. In these experiments, EP was consistently able to outperform the human subjects.

B1.4.2.2 Extensions

The basic EP paradigm may be described by the following EP algorithm:

$t := 0$;
initialize $P(0) := \{a'_1(0), a'_2(0), \ldots, a'_\mu(0)\}$
evaluate $P(0) : \{\Phi(a'_1(0)), \Phi(a'_2(0)), \ldots, \Phi(a'_\mu(0))\}$
iterate
{
mutate: $P'(t) := m_{\Theta_m}(P(t))$
evaluate: $P'(t) : \{\Phi(a'_1(t)), \Phi(a'_2(t)), \ldots, \Phi(a'_\lambda(t))\}$
select: $P(t+1) := s_{\Theta_s}(P'(t) \cup Q)$
$t := t + 1$;
}

where:

a' is an individual member in the population
$\mu \geq 1$ is the size of the parent population
$\lambda \geq 1$ is the size of the offspring population
$P(t) := \{a'_1(t), a'_2(t), \ldots, a'_\mu(t)\}$ is the population at time t
$\Phi : I \rightarrow \Re$ is the fitness mapping
m_{Θ_m} is the mutation operator with controlling parameters Θ_m
s_{Θ_s} is the selection operator $\ni s_{\Theta_s} : \left(I^\lambda \cup I^{\mu+\lambda}\right) \rightarrow I^\mu$
$Q \in \{\emptyset, P(t)\}$ is a set of individuals additionally accounted for in the selection step, i.e. parent solutions.

Other than on initialization, the search space is generally unconstrained; constraints are utilized for generation and initialization of starting parent solutions. Constrained optimization may be addressed through imposition of the requirement that (i) the *mutation operator* is formulated to only generate legitimate solutions (often impossible) or (ii) a *penalty function* is applied to offspring mutations lying outside the constraint bounds in such a manner that they do not become part of the next generation. The objective function explicitly defines the fitness values which may be scaled to positive values (although this is not a requirement, it is sometimes performed to alter the range for ease of implementation).

C3.2.4 C5.2

In early versions of EP applied to continuous parameter optimization (Fogel 1992) the mutation operator is Gaussian with a zero mean and variance obtained for each component of the object variable vector as the square root of a linear transform of the fitness value $\varphi(\boldsymbol{x})$.

$$x_i(k+1) := x_i(k) + \sqrt{\beta_i(k)\varphi(x_i(k) + \gamma_i)} + N_i(0, 1)$$

where $\boldsymbol{x}(k)$ is the object variable vector, β is the proportionality constant, and γ is an offset parameter. Both β and γ must be set externally for each problem. $N_i(0, 1)$ is the ith independent sample from a Gaussian distribution with zero mean and unit variance.

Several extensions to the finite-state machine formulation of Fogel *et al* (1966) have been offered to address continuous optimization problems as well as to allow for various degrees of parametric self-adaptation (Fogel 1991a, 1992, 1995). There are three main variants of the basic paradigm, identified as follows:

(i) original EP, where continuous function optimization is performed without any self-adaptation mechanism;
(ii) *continuous* EP where new individuals in the population are inserted directly without iterative generational segmentation (i.e. an individual becomes part of the existing (surviving) population without waiting for the conclusion of a discrete generation; this is also known as *steady-state selection* in GAs and $(\mu + 1)$ *selection* in ES);
(iii) *self-adaptive* EP, which augments the solution vectors with one or more parameters governing the mutation process (e.g. variances, covariances) to permit self-adaptation of these parameters through the same iterative mutation, scoring, and selection process. In addition, self-adaptive EP may also be continuous in the sense of (ii) above.

C2.7.3 B1.3 C7.1

The original EP is an extension of the formulation of Fogel *et al* (1966) wherein continuous-valued functions replace the discrete alphabets of finite-state machines. The continuous form of EP was investigated by Fogel and Fogel (1993). To properly simulate this algorithmic variant, it is necessary to insert new population members by asynchronous methods (e.g. event-driven interrupts in a true multitasking, real-time operating system). Iterative algorithms running on a single central processing unit (CPU) are much more prevalent, since they are easier to program on today's computers, hence most implementations of EP are performed on a generational (epoch-to-epoch) basis.

Self-adaptive EP is an important extension of the algorithm in that it successfully overcomes the need for explicit user-tuning of the parameters associated with mutation. Global convergence may be obtained even in the presence of suboptimal parameterization, but available processing time is most often a precious resource and any mechanism for optimizing the convergence rate is helpful. As proposed by Fogel (1992, 1995), EP can self-adapt the variances for each individual in the following way:

$$x_i(k+1) := x_i(k) + \upsilon_i(k) * N_i(0,1)$$
$$\upsilon_i(k+1) := \upsilon_i(k) + [\sigma \upsilon_i(k)]^{1/2} * N_i(0,1).$$

The variable σ ensures that the variance υ_i remains nonnegative. Fogel (1992) suggests a simple rule wherein $\forall \upsilon_i(k) \leq 0$, $\upsilon_i(k)$ is set to ξ, a value close to but not identically equal to zero (to allow some degree of mutation). The sequence of updating the object variable x_i and variance υ_i was proposed to occur in opposite order from that of ESs (Bäck and Schwefel 1993, Rechenberg 1965, Schwefel 1981). Gehlhaar and Fogel (1996) provide evidence favoring the ordering commonly found in ES.

Further development of this theme led Fogel (1991a, 1992) to extend the procedure to alter the correlation coefficients between components of the object vector. A symmetric correlation coefficient matrix $\mathbf{P}$ is incorporated into the evolutionary paradigm in addition to the self-adaptation of the standard deviations. The components of $\mathbf{P}$ are initialized over the interval $[-1, 1]$ and mutated by perturbing each component, again, through the addition of independent realizations from a Gaussian random distribution. Bounding limits are placed upon the resultant mutated variables wherein any mutated coefficient which exceeds the bounds $[-1, 1]$ is reset to the upper or lower limit, respectively. Again, this methodology is similar to that of Schwefel (1981), as perturbations of both the standard deviations and rotation angles (determined by the covariance matrix $\mathbf{P}$) allow adaptation to arbitrary contours on the error surface. This self- adaptation through the incorporation of correlated mutations across components of each parent object vector provides a mechanism for expediting the convergence rate of EP.

C2.3 Fogel (1988) developed different selection operators which utilized *tournament competition* between solution organisms. These operators assigned a number of wins for each solution organism based on a set of individual competitions (using fitness scores as the determining factor) among each solution and each of the q competitors randomly selected from the total population.

B1.4.3 Current directions

Since the explosion of research into evolutionary algorithms in the late 1980s and early 1990s, EP has been applied to a wide range of problem domains with considerable success. Application areas in the current literature include training, construction, and optimization of neural networks, optimal routing (in two, three, and higher dimensions), drug design, bin packing, automatic control, game theory, and optimization of intelligently interactive behaviors of autonomous entities, among many others. Beginning in 1992, annual conferences on EP have brought much of this research into the open where these and other applications as well as basic research have expanded into numerous interdisciplinary realms.

D1 Notable within a small sampling of the current research is the work in *neural network design*. Early efforts (Porto 1989, Fogel *et al* 1990, McDonnell 1992, and others) focused on utilizing EP for training neural networks to prevent entrapment in local minima. This research showed not only that EP was well suited to training a range of network topologies, but also that it was often more efficient than conventional (e.g. gradient-based) methods and was capable of finding optimal weight sets while escaping local minima points. Later research (Fogel 1992, Angeline *et al* 1994, McDonnell and Waagen 1993) involved simultaneous evolution of both the weights and structure of feedforward and feedback networks. Additional research into the areas of using EP for robustness training (Sebald and Fogel 1992), and for

D2 designing *fuzzy neural networks* for feature selection, pattern clustering, and classification (Brotherton and Simpson 1995) have been very successful as well as instructive.

EP has been also used to solve optimal routing problems. The *traveling salesman problem* (TSP), one of many in the class of nondeterministic-polynomial-time- (NP-) complete (see Aho *et al* 1974) problems, has been studied extensively. Fogel (1988, 1993) demonstrated the capability of EP to address such problems. A representation was used wherein each of the cities to be visited was listed in order, with candidate solutions being permutations of this listing. A population of random tours is scored with respect to their Euclidean length. Each of the tours is mutated using one of many possible inversion operations (e.g. select two cities in the tour at random and reverse the order of the segment defined by the two cities) to generate an offspring. All of the offspring are then scored, with either elitist or stochastic competition used to cull lower-scoring members from the population. Optimization of the tours was quite rapid. In one such experiment with 1000 cities uniformly distributed, the best tour (after only 4×10^7 function evaluations) was estimated to be within 5–7% of the optimal tour length. Thus, excellent solutions were obtained after searching only an extremely small portion of the total potential search space. G9.5

EP has also been utilized in a number of medical applications. For example, the issue of optimizing drug design was researched by Gehlhaar *et al* (1995). EP was utilized to perform a conformational and position search within the binding site of a protein. The search space of small molecules which could potentially dock with the crystallographically determined binding site was explored iteratively guided by a database of crystallographic protein–ligand complexes. Geometries were constrained by known physical (in three dimensions) and chemical bounds. Results demonstrated the efficacy of this technique as it was orders of magnitude faster in finding suitable ligands than previous hands-on methodologies. The probability of successfully predicting the proper binding modes for these ligands was estimated at over 95% using nominal values for the crystallographic binding mode and number of docks attempted. These studies have permitted the rapid development of several candidate drugs which are currently in clinical trials.

The issue of utilizing EP to *control systems* has been addressed widely (Fogel and Fogel 1990, Fogel 1991a, Page *et al* 1992, and many others). Automatic control of fuzzy heating, ventilation, and air conditioning (HVAC) controllers was addressed by Haffner and Sebald (1993). In this study, a nonlinear, multiple-input, multiple-output (MIMO) model of a HVAC system was used and controlled by a fuzzy controller designed using EP. Typical fuzzy controllers often use trial and error methods to determine parameters and transfer functions, hence they can be quite time consuming with a complex MIMO HVAC system. These experiments used EP to design the membership functions and values (later studies were extended to find rules as well as responsibilities of the primary controller) to automate the tuning procedure. EP worked in an overall search space containing 76 parameters, 10 controller inputs, seven controller outputs, and 80 rules. Simulation results demonstrated that EP was quite effective at choosing the membership functions of the control laboratory and corridor pressures in the model. The synergy of combining EP with fuzzy set constructs proved quite fruitful in reducing the time required to design a stable, functioning HVAC system. F1.3

Game theory has always been at the forefront of artificial intelligence research. One interesting game, the iterated prisoner's dilemma, has been studied by numerous investigators (Axelrod 1987, Fogel 1991b, Harrald and Fogel 1996, and others). In this two-person game, each player can choose one of two possible behavioral policies: defection or cooperation. Defection implies increasing one's own reward at the expense of the opposing player, while cooperation implies increasing the reward for both players. If the game is played over a single iteration, the dominant move is defection. If the players' strategies depend on the results of previous iterations, mutual cooperation may possibly become a rational policy, whereas if the sequence of strategies is not correlated, the game degenerates into a series of single plays with defection being the end result. Each player must choose to defect or cooperate on each trial. Table B1.4.1 describes a general form of the payoff function in the prisoner's dilemma.

In addition, the payoff matrix defining the game is subject to the following constraints (Rapoport 1966):

$$2\gamma_1 > \gamma_2 + \gamma_3$$
$$\gamma_3 > \gamma_1 > \gamma_4 > \gamma_2.$$

Both neural network approaches (Harrald and Fogel 1996) and finite-state machine approaches (Fogel 1991b) have been applied to this problem. Finite-state machines are typically used where there are discrete choices between cooperation and defection. Neural networks allow for a continuous range of choices between these two opposite strategies. Results of these preliminary experiments using EP, in

Table B1.4.1. A general form of the payoff matrix for the prisoner's dilemma problem. γ_1 is the payoff to each player for mutual cooperation. γ_2 is the payoff for cooperating when the other player defects. γ_3 is the payoff for defecting when the other player cooperates. γ_4 is the payoff to each player for mutual defection. Entries (α, β) indicate payoffs to players A and B, respectively.

		Player B	
		C	D
Player A	C	(γ_1, γ_1)	(γ_2, γ_3)
	D	(γ_3, γ_2)	(γ_4, γ_4)

general, indicated that mutual cooperation is more likely to occur when the behaviors are limited to the extremes (the finite-state machine representation of the problem), whereas in the neural network continuum behavioral representation of the problem, it is easier to slip into a state of mutual defection.

Development of interactively intelligent behaviors was investigated by Fogel *et al* (1996). EP was used to optimize computer-generated force (CGF) behaviors such that they learned new courses of action adaptively as changes in the environment (i.e. presence or absence of opposing side forces) were encountered. The actions of the CGFs were created at the response of an event scheduler which recognized significant changes in the environment as perceived by the forces under evolution. New plans of action were found during these event periods by invoking an evolutionary program. The iterative EP process was stopped when time or CPU limits were met, and relinquished control of the simulated forces back to the CGF simulator after transmitting newly evolved instruction sets for each simulated unit. This process proved quite successful and offered a significant improvement over other rule-based systems.

B1.4.4 Future research

B2.2.5 Important research is currently being conducted into the understanding of the *convergence properties* of EP, as well as the basic mechanisms of different mutation operators and selection mechanisms. Certainly of great interest is the potential for self-adaptation of exogeneous parameters of the mutation operation (meta and Rmeta-EP) , as this not only frees the user from the often difficult task of parameterization, but also provides a built-in, automated mechanism for providing optimal settings throughout a range of problem domains. The number of application areas of this optimization technique is constantly growing. EP, along with the other EC techniques, is being used on previously untenable, often NP-complete, problems which occur quite often in commercial and military problems.

References

Aho A V, Hopcroft J E and Ullman J D 1974 *The Design and Analysis of Computer Algorithms* (Reading, MA: Addison-Wesley) pp 143–5, 318–26

Angeline P, Saunders G and Pollack J 1994 Complete induction of recurrent neural networks *Proc. 3rd Ann. Conf. on Evolutionary Programming (San Diego, CA, 1994)* ed A V Sebald and L J Fogel (Singapore: World Scientific) pp 1–8

Atmar W 1992 On the rules and nature of simulated evolutionary programming *Proc. 1st Ann. Conf. on Evolutionary Programming (La Jolla, CA, 1992)* ed D B Fogel and W Atmar (San Diego, CA: Evolutionary Programming Society) pp 17–26

Axelrod R 1987 The evolution of strategies in the iterated prisoner's dilemma *Genetic Algorithms and Simulated Annealing* ed L Davis (London) pp 32–42

Bäck T and Schwefel H-P 1993 An overview of evolutionary algorithms for parameter optimization *Evolutionary Comput.* **1** 1–23

Brotherton T W and Simpson P K 1995 Dynamic feature set training of neural networks for classification *Evolutionary Programming IV: Proc. 4th Ann. Conf. on Evolutionary Programming (San Diego, CA, 1995)* ed J R McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press) pp 83–94

Burton D M 1976 *Elementary Number Theory* (Boston, MA: Allyn and Bacon) p 136–52

Flood M M 1962 Stochastic learning theory applied to choice experiments with cats, dogs and men *Behavioral Sci.* **7** 289–314

Fogel D B 1988 An evolutionary approach to the traveling salesman problem *Biol. Cybernet.* **60** 139–44

——1991a *System Identification through Simulated Evolution* (Needham, MA: Ginn)

——1991b The evolution of intelligent decision making in gaming *Cybernet. Syst.* **22** 223–36
——1992 *Evolving Artificial Intelligence* PhD Dissertation, University of California
——1993 Applying evolutionary programming to selected traveling salesman problems *Cybernet. Syst.* **24** 27–36
——1995 *Evolutionary Computation, Toward a New Philosophy of Machine Intelligence* (Piscataway, NJ: IEEE)
Fogel D B and Fogel L J 1990 Optimal routing of multiple autonomous underwater vehicles through evolutionary programming *Proc. Symp. on Autonomous Underwater Vehicle Technology* (Washington, DC: IEEE Oceanic Engineering Society) pp 44–7
Fogel D B, Fogel L J and Porto V W 1990 Evolving neural networks *Biol. Cybernet.* **63** 487–93
Fogel G B and Fogel D B 1993 Continuous evolutionary programming: analysis and experiments *Cybernet. Syst.* **26** 79–90
Fogel L J 1962 Autonomous automata *Industrial Res.* **4** 14–9
Fogel L J 1964 *On The Organization of Intellect* PhD Dissertation, University of California
Fogel L J and Burgin G H 1969 *Competitive Goal-Seeking through Evolutionary Programming* Air Force Cambridge Research Labratories Final Report Contract AF19(628)-5927
Fogel L J, Owens A J and Walsh M J 1966 *Artificial Intelligence through Simulated Evolution* (New York: Wiley)
Fogel L J, Porto V W and Owen M 1996 An intelligently interactive non-rule-based computer generated force *Proc. 6th Conf. on Computer Generated Forces and Behavioral Representation* (Orlando, FL: Institute for Simulation and Training STRICOM-DMSO) pp 265–70
Gehlhaar D K and Fogel D B 1996 Tuning evolutionary programming for conformationally flexible molecular docking *Proc. 5th Ann. Conf. on Evolutionary Programming (1996)* ed L J Fogel, P J Angeline and T Bäck (Cambridge, MA: MIT Press) pp 419–29
Gehlhaar D K, Verkhivker G, Rejto P A, Fogel D B, Fogel L J and Freer S T 1995 Docking conformationally flexible small molecules into a protein binding site through evolutionary programming *Evolutionary Programming IV: Proc. 4th Ann. Conf. on Evolutionary Programming (San Diego, CA, 1995)* (Cambridge, MA: MIT Press) pp 615–27
Harrald P G and Fogel D B 1996 Evolving continuous behaviors in the iterated prisoner's dilemma *BioSystems* **37** 135–45
Haffner S B and Sebald A V 1993 Computer-aided design of fuzzy HVAC controllers using evolutionary programming *Proc. 2nd Ann. Conf. on Evolutionary Programming (San Diego, CA, 1993)* ed D B Fogel and W Atmar (San Diego, CA: Evolutionary Programming Society) pp 98–107
McDonnell J M 1992 Training neural networks with weight constraints *Proc. 1st Ann. Conf. on Evolutionary Programming (La Jolla, CA, 1992)* ed D B Fogel and W Atmar (San Diego, CA: Evolutionary Programming Society) pp 111–9
McDonnell J M and Waagen D 1993 Neural network structure design by evolutionary programming *Proc. 2nd Ann. Conf. on Evolutionary Programming (San Diego, CA, 1993)* ed D B Fogel and W Atmar (San Diego, CA: Evolutionary Programming Society) pp 79–89
Mealy G H 1955 A method of synthesizing sequential circuits *Bell Syst. Tech. J.* **34** 1054–79
Moore E F 1957 Gedanken-experiments on sequential machines: automata studies *Annals of Mathematical Studies* vol 34 (Princeton, NJ: Princeton University Press) pp 129–53
Page W C, McDonnell J M and Anderson B 1992 An evolutionary programming approach to multi-dimensional path planning *Proc. 1st Ann. Conf. on Evolutionary Programming (La Jolla, CA, 1992)* ed D B Fogel and W Atmar (San Diego, CA: Evolutionary Programming Society) pp 63–70
Porto V W 1989 Evolutionary methods for training neural networks for underwater pattern classification *24th Ann. Asilomar Conf. on Signals, Systems and Computers* vol 2 pp 1015–19
Rapoport A 1966 *Optimal Policies for the Prisoner's Dilemma* University of North Carolina Psychometric Laboratory Technical Report 50
Rechenberg I 1965 *Cybernetic Solution Path of an Experimental Problem* Royal Aircraft Establishment Translation 1122
Schwefel H-P 1981 *Numerical Optimization of Computer Models* (Chichester: Wiley)
Sebald A V and Fogel D B 1992 Design of fault tolerant neural networks for pattern classification *Proc. 1st Ann. Conf. on Evolutionary Programming (La Jolla, CA, 1992)* ed D B Fogel and W Atmar (San Diego, CA: Evolutionary Programming Society) pp 90–9
Walsh M J, Burgin G H and Fogel L J 1970 *Prediction and Control through the Use of Automata and their Evolution* US Navy Final Report Contract N00014-66-C-0284

Further reading

There are several excellent general references available to the reader interested in furthering his or her knowledge in this exciting area of EC. The following books are a few well-written examples providing a good theoretical background in EP as well as other evolutionary algorithms.

1. Bäck T 1996 *Evolutionary Algorithms in Theory and Practice* (New York: Oxford University Press)
2. Fogel D B 1995 *Evolutionary Computation, Toward a New Philosophy of Machine Intelligence* (Piscataway, NJ: IEEE)
3. Schwefel H-P 1981 *Numerical Optimization of Computer Models* (Chichester: Wiley)
4. Schwefel H-P 1995 *Evolution and Optimization Seeking* (New York: Wiley)

B1.5 Derivative methods

Kenneth E Kinnear, Jr (B1.5.1), *Robert E Smith* (B1.5.2) *and Zbigniew Michalewicz* (B1.5.3)

Abstract

See the individual abstracts for sections B1.5.1–B1.5.3.

B1.5.1 Genetic programming

Kenneth E Kinnear, Jr

Abstract

The fundamental concepts of genetic programming are discussed here. Genetic programming is a form of evolutionary algorithm that is distinguished by a particular set of choices as to representation, genetic operator design, and fitness evaluation.

B1.5.1.1 Introduction

This article describes the fundamental concepts of genetic programming (GP) (Koza 1989, 1992). Genetic programming is a form of evolutionary algorithm which is distinguished by a particular set of choices as to representation, genetic operator design, and fitness evaluation. When examined in isolation, these choices define an approach to evolutionary computation (EC) which is considered by some to be a specialization of the *genetic algorithm* (GA). When considered together, however, these choices define a conceptually different approach to evolutionary computation which leads researchers to explore new and fruitful lines of research and practical applications. B1.2

B1.5.1.2 Genetic programming defined and explained

Genetic programming is implemented as an evolutionary algorithm in which the data structures that undergo adaptation are executable computer programs. Fitness evaluation in genetic programming involves executing these evolved programs. Genetic programming, then, involves an evolution-directed search of the space of possible computer programs for ones which, when executed, will produce the best fitness.

In short, genetic programming breeds computer programs. To create the initial population a large number of computer programs are generated at random. Each of these programs is executed and the results of that execution are used to assign a fitness value to each program. Then a new population of programs, the next generation, is created by directly copying certain selected existing programs, where the selection is based on their fitness values. This population is filled out by creating a number of new offspring programs through genetic operations on existing parent programs which are selected based, again, on their fitness. Then, this new population of programs is again evaluated and a fitness is assigned to each program based on the results of its evaluation. Eventually this process is terminated by the creation and evaluation of a 'correct' program or the recognition of some other specific termination criteria.

More specifically, at the most basic level, genetic programming is defined as a genetic algorithm with some unusual choices made as to the representation of the problem, the genetic operators used to modify that representation, and the fitness evaluation techniques employed.

A specialized representation: executable programs. Any evolutionary algorithm is distinguished by the structures used to represent the problem to be solved. These are the structures which undergo transformation, and in which the potential solutions reside.

C1.2 Originally, most genetic algorithms used *linear strings of bits* as the structures which evolved (Holland 1975), and the representation of the problem was typically the encoding of these bits as numeric or logical parameters of a variety of algorithms. The evolving structures were often used as parameters to human-coded algorithms. In addition, the bitstrings used were frequently of fixed length, which aided in the translation into parameters for the algorithms involved.

More recently, genetic algorithms have appeared with real-valued numeric sequences used as the evolvable structures, still frequently used as parameters to particular algorithms. In recent years, many genetic algorithm implementations have appeared with sequences which are of variable length, sometimes based on the order of the sequences, and which contain more complex and structured information than parameters to existing algorithms.

The representation used by genetic programming is that of an executable program. There is no single form of executable program which is used by all genetic programming implementations, although many

C1.6 implementations use a *tree-structured representation* highly reminiscent of a LISP functional expression. These representations are almost always of a variable size, though for implementation purposes a maximum size is usually specified.

Figure B1.5.1 shows an example of a tree-structured representation for a genetic programming implementation. The specific task for which this is a reasonable representation is the learning of a Boolean function from a set of inputs.

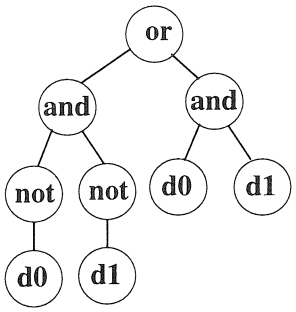


Figure B1.5.1. Tree-structured representation used in genetic programming.

This figure contains two different types of node (as do most genetic programming representations) which are called functions and terminals. Terminals are usually inputs to the program, although they may also be constants. They are the variables which are set to values external to the program itself prior to the fitness evaluation performed by executing the program. In this example d0 and d1 are the terminals. They can take on binary values of either zero or one.

Functions take inputs and produce outputs and possibly produce side-effects. The inputs can be either a terminal or the output of another function. In the above example, the functions are AND, OR, and NOT. Two of these functions are functions of two inputs, and one is a function of one input. Each produces a single output and no side effect.

The fitness evaluation for this particular *individual* is determined by the effectiveness with which it will produce the correct logical output for all of the test cases against which it is tested.

One way to characterize the design of a representation for an application of genetic programming to a particular problem is to view it as the design of a language, and this can be a useful point of view. Perhaps it is more useful, however, to view the design of a genetic programming representation as that of the design of a virtual machine—since usually the execution engine must be designed and constructed as well as the representation or language that is executed.

The representation for the program (i.e. the definition of the functions and terminals) must be designed along with the virtual machine that is to execute them. Rarely are the programs evolved in genetic programming given direct control of the central processor of a computer (although see the article by Nordin (1994)). Usually, these programs are interpreted under control of a virtual machine which defines the functions and terminals. This includes the functions which process the data, the terminals that provide

the inputs to the programs, and any control functions whose purpose is to affect the execution flow of the program.

As part of this virtual machine design task, it is important to note that the output of any function or the value of any terminal may be used as the input to any function. Initially, this often seems to be a trivial problem, but when actually performing the design of the representation and virtual machine to execute that representation, it frequently looms rather large. Two solutions are typically used for this problem. One approach is to design the virtual machine, represented by the choice of functions and terminals, to use only a single data type. In this way, the output of any function or the value of any terminal is acceptable as input to any function. A second approach is to allow more than one data type to exist in the virtual machine. Each function must then be defined to operate on any of the existing data types. Implicit coercions are performed by each function on its input to convert the data type that it receives to one that it is more normally defined to process. Even after handling the data type problem, functions must be defined over the entire possible range of argument values. Simple arithmetic division must be defined to return some value even when division by zero is attempted.

It is important to note that the definition of functions and the virtual machine that executes them is not restricted to functions whose only action is to provide a single output value based on their inputs. Genetic programming functions are often defined whose primary purpose is the actions they take by virtue of their side-effects. These functions must return some value as well, but their real purpose is interaction with an environment external to the genetic programming system.

An additional type of side-effect producing function is one that implements a control structure within the virtual machine defined to execute the genetically evolved program. All of the common programming control constructs such as if–then–else, while–do, for, and others have been implemented as evolvable control constructs within genetic programming systems. Looping constructs must be protected in such a way that they will never loop forever, and usually have an arbitrary limit set on the number of loops which they will execute.

As part of the initialization of a genetic programming run, a large number of individual programs are generated at random. This is relatively straightforward, since the genetic programming system is supplied with information about the number of arguments required by each function, as well as all of the available terminals. Random program trees are generated using this information, typically of a relatively small size. The program trees will tend to grow quickly to be quite large in the absence of some explicit evolutionary pressure toward small size or some simple hard-coded limits to growth (see Section C4.4 for some methods to handle this problem). C4.4

Genetic operators for evolving programs. The second specific design approach that distinguishes genetic programming from other types of genetic algorithm is the design of the genetic operators. Having decided to represent the problem to be solved as a population of computer programs, the essence of an evolutionary algorithm is to evaluate the fitness of the individuals in the population and then to create new members of the population based in some way on the individuals which have the highest fitness in the current population.

In genetic algorithms, *recombination* is typically the key genetic operator employed, with some utility ascribed to mutation as well. In this way, genetic programming is no different from any other genetic algorithm. genetic algorithms usually have genetic material organized in a linear fashion and the recombination, or crossover, algorithm defined for such genetic material is quite straightforward (see Section C3.3.1). The usual representation of genetic programming programs as tree-structured combinations of functions and terminals requires a different form of recombination algorithm. A major step in the invention of genetic programming was the design of a recombination operator which would simply and easily allow the creation of an offspring program tree using as inputs the program trees of two individuals of generally high fitness as parents (Cramer 1985, Koza 1989, 1992). C3.3.1

In any evolutionary algorithm it is vitally important that the fitness of the offspring be related to that of the parents, or else the process degenerates into one of random search across whatever representation space was chosen. It is equally vital that some variation, indeed heritable variation, be introduced into the offspring's fitness, otherwise no improvement toward an optimum is possible.

The tree-structured genetic material usually used in genetic programming has a particularly elegant recombination operator that may be defined for it. In figure B1.5.2, there are two parent program trees, (*a*) and (*b*). They are to be recombined through crossover to create an offspring program tree (*c*). A

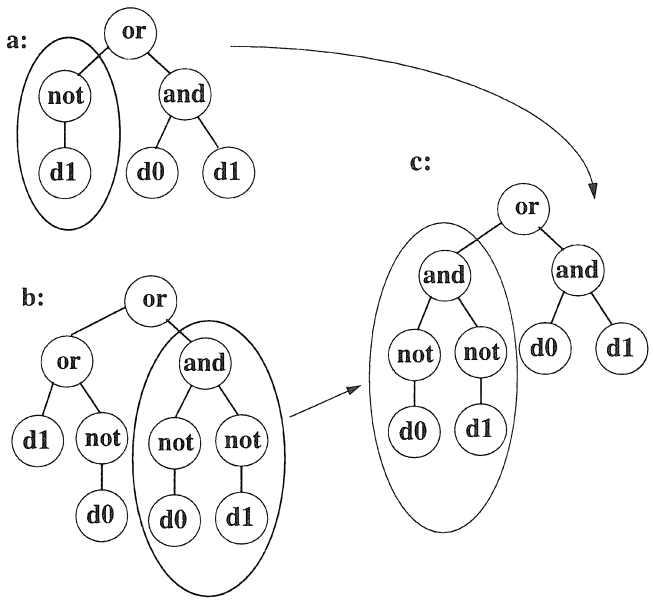


Figure B1.5.2. Recombination in genetic programming.

subtree is chosen in each of the parents, and the offspring is created by inserting the subtree chosen from (*b*) into the place where the subtree was chosen in (*a*). This very simply creates an offspring program tree which preserves the same constraints concerning the number of inputs to each function as each parent tree. In practice it yields a offspring tree whose fitness has enough relationship to that of its parents to support the evolutionary search process. Variations in this crossover approach are easy to imagine, and are currently the subject of considerable active research in the genetic programming community (D'haeseleer 1994, Teller 1996).

C3.2 *Mutation* is a genetic operator which can be applied to a single parent program tree to create an offspring tree. The typical mutation operator used selects a point inside a parent tree, and generates a new random subtree to replace the selected subtree. This random subtree is usually generated by the same procedure used to generate the initial population of program trees.

Fitness evaluation of genetically evolved programs. Finally, then, the last detailed distinction between genetic programming and a more usual implementation of the genetic algorithm is that of the assignment of a fitness value for a individual.

In genetic programming, the representation of the individual is a program which, when executed under control of a defined virtual machine, implements some algorithm. It may do this by returning some value (as would be the case for a system to learn a specific Boolean function) or it might do this through the performance of some task through the use of functions which have side-effects that act on a simulated (or even the real) world.

The results of the program's execution are evaluated in some way, and this evaluation represents the fitness of the individual. This fitness is used to drive the selection process for copying into the next generation or for the selection of parents to undergo genetic operations yielding offspring. Any selection C2 operator from those presented in Chapter C2 can be used.

There is certainly a desire to evolve programs using genetic programming that are 'general', that is to say that they will not only correctly process the fitness cases on which they are evolved, but will process correctly any fitness cases which could be presented to them. Clearly, in the cases where there are infinitely many possible cases, such as evolving a general sorting algorithm (Kinnear 1993), the evolutionary process can only be driven by a very limited number of fitness cases. Many of the lessons from machine learning on the tradeoffs between generality and performance on training cases have been helpful to genetic programming researchers, particularly those from decision tree approaches to machine learning (Iba *et al* 1994).

B1.5.1.3 The development of genetic programming

LISP was the language in which the ideas which led to genetic programming were first developed (Cramer 1985, Koza 1989, 1992). LISP has always been one of the preeminent language choices for implementation where programs need to be treated as data. In this case, programs are data while they are being evolved, and are only considered executable when they are undergoing fitness evaluation.

As genetic programming itself evolved in LISP, the programs that were executed began to look less and less like LISP programs. They continued to be tree structured but soon few if any of the functions used in the evolved programs were standard LISP functions. Around 1992 many people implemented genetic programming systems in C and C++, along with many other programming languages. Today, other than a frequent habit of printing the representation of tree-structured genetic programs in a LISP-like syntax, there is no particular connection between genetic programming and LISP.

There are many public domain implementations of genetic programming in a wide variety of programming languages. For further details, see the reading list at the end of this section.

B1.5.1.4 The value of genetic programming

Genetic programming is defined as a variation on the theme of genetic algorithms through some specific selections of representation, genetic operators appropriate to that representation, and fitness evaluation as execution of that representation in a virtual machine. Taken in isolation, these three elements do not capture the value or promise of genetic programming. What makes genetic programming interesting is the conceptual shift of the problem being solved by the genetic algorithm. A genetic algorithm searches for something, and genetic programming shifts the search from that of parameter discovery for some existing algorithm designed to solve a problem to a search for a program (or algorithm) to solve the problem directly. This shift has a number of ramifications.

- This conceptualization of evolving computer programs is powerful in part because it can change the way that we think about solving problems. Through experience, it has become natural to think about solving problems through a process of human-oriented program discovery. Genetic programming allows us to join this approach to problem solving with powerful EC-based search techniques.

  An example of this is a variation of genetic programming called stack genetic programming (Perkis 1994), where the program is a variable-length linear string of functions and terminals, and the argument passing is defined to be on a stack. The genetic operators in a linear system such as this are much closer to the traditional genetic algorithm operators, but the execution and fitness evaluation (possibly including side-effects) is equivalent to any other sort of genetic programming. The characteristics of stack genetic programming have not yet been well explored but it is clear that it has rather different strengths and weaknesses than does traditional genetic programming.

  Many of the approaches to simulation of adaptive behavior involve simple programs designed to control *animats*. The conceptualization of evolving computer programs as presented by genetic programming fits well with work on evolving adaptive entities (Reynolds 1994, Sims 1994).
- There has been a realization that not only can we evolve programs that are built from human-created functions and terminals, but that the functions from which they are built can evolve as well. Koza's invention of automatically defined functions (ADFs) (Koza 1994) is one such example of this realization. ADFs allow the definitions of certain subfunctions to evolve even while the functions that call them are evolving. For certain classes of problems, ADFs result in considerable increases in performance (Koza 1994, Angeline and Pollack 1993, Kinnear 1994).
- Genetic programming is capable of integrating a wide variety of existing capabilities, and has potential to tie together several complementary subsystems into an overall hybrid system. The functions need not be simple arithmetic or logical operators, but could instead be fast Fourier transforms, GMDH systems, or other complex building blocks. They could even be the results of other evolutionary computation algorithms.
- The genetic operators that create offspring programs from parent programs are themselves programs. These programs can also be evolved either as part of a separate process, or in a coevolutionary way with the programs on which they operate. While any evolutionary computation algorithm could have parameters that affect the genetic operators be part of the evolutionary process, genetic programming provides a natural way to let the operators (defined as programs) evolve directly (Teller 1996, Angeline 1996).

- Genetic programming naturally enhances the possibility for increasingly indirect evolution. As an example of the possibilities, genetic programming has been used to evolve grammars which, when executed, produce the structure of an untrained *neural network*. These neural networks are then trained, and the trained networks are then evaluated on a test set. The results of this evaluation are then used as the fitnesses of the evolved grammars (Gruau 1993). D1

This last example is a step along the path toward modeling embryonic development in genetic programming. The opportunity exists to evolve programs whose results are themselves programs. These resulting programs are then executed and their values or side-effects are evaluated—and become the fitness for the original evolving, program creating programs. The analogy to natural embryonic development is clear, where the genetic material, the genotype, produces through development a body, the phenotype, which then either does or does not produce offspring, the fitness (Kauffman 1993).

Genetic programming is valuable in part because we find it natural to examine issues such as those mentioned above when we think about evolutionary computation from the genetic programming perspective.

B1.5.2 Learning classifier systems

Robert E Smith

Abstract

Learning classifier systems (LCSs) are rule-based machine learning systems that use genetic algorithms as their primary rule discovery mechanism. There is no standard version of the LCS; however, all LCSs share some common characteristics. These characteristics are introduced here by first examining reinforcement learning problems, which are the most frequent application area for LCSs. This introduction provides a motivational framework for the representation and mechanics of LCSs. Also examined are the variations of the LCS scheme.

B1.5.2.1 Introduction

The learning classifier system (LCS) (Goldberg 1989, Holland *et al* 1986) is often referred to as the primary machine learning technique that employs genetic algorithms (GAs). It is also often described as a production system framework with a genetic algorithm as the primary rule discovery method. However, the details of LCS operation vary widely from one implementation to another. In fact, no standard version of the LCS exists. In many ways, the LCS is more of a concept than an algorithm. To explain details of the LCS concept, this article will begin by introducing the type of machine learning problem most often associated with the LCS. This discussion will be followed by a overview of the LCS, in its most common form. Final sections will introduce the more complex issues involved in LCSs.

B1.5.2.2 Types of learning problem

To introduce the LCS, it will be useful to describe types of machine learning problem. Often, in the literature, machine learning problems are described in terms of cognitive psychology or animal behavior. This discussion will attempt to relate the terms used in machine learning to engineering control.

Consider the generic control problem shown in figure B1.5.3. In this problem, inputs from an external control system, combined with uncontrollable disturbances from other sources, change the state of the plant. These changes in state are reflected in the state information provided by the plant. Note that, in general, the state information can be incomplete and noisy.

Consider the *supervised learning* problem shown in figure B1.5.4 (Barto 1990). In this problem, an inverse plant model (or teacher) is available that provides errors directly in terms of control actions. Given this direct error feedback, the parameters of the control system can be adjusted by means of gradient descent, to minimize error in control actions. Note that this is the method used in the neural network backpropagation algorithm.

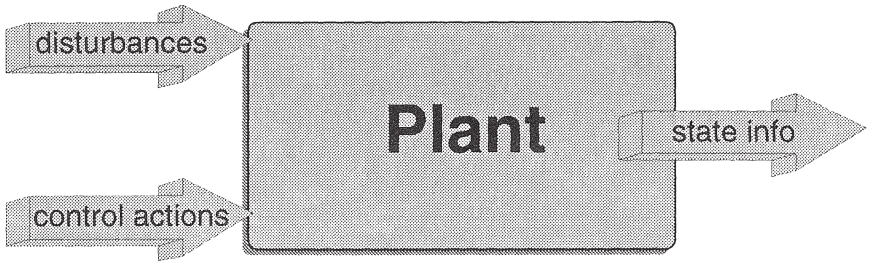


Figure B1.5.3. A generic control problem.

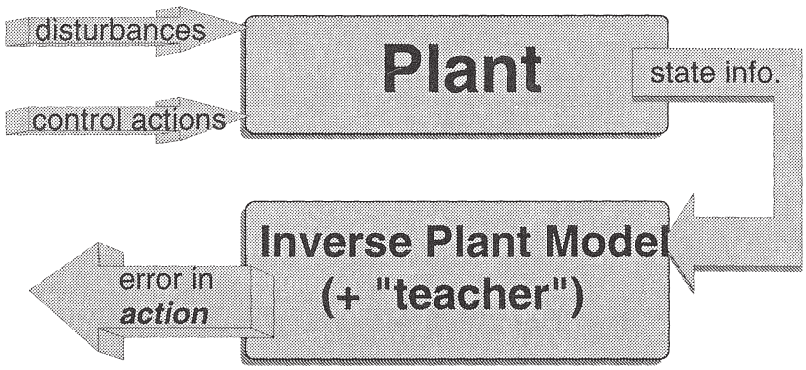


Figure B1.5.4. A supervised learning problem.

Now consider the *reinforcement learning* problem shown in figure B1.5.5 (Barto 1990). Here, no inverse plant model is available. However, a critic is available that indicates error in the state information from the plant. Because error is not directly provided in terms of control actions, the parameters of the controller cannot be directly adjusted by methods such as gradient descent.

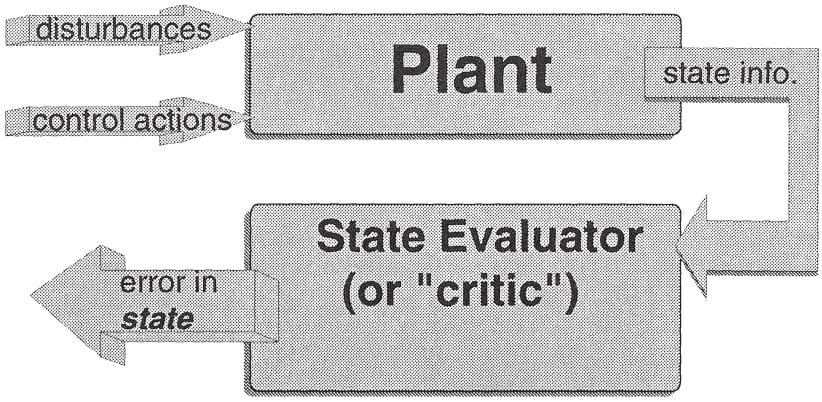


Figure B1.5.5. A reinforcement learning problem.

The remaining discussion will consider the control problem to operate as a *Markov decision problem*. That is, the control problem operates in discrete time steps, the plant is always in one of a finite number of discrete states, and a finite, discrete number of control actions are available. At each time step, the control action alters the probability of moving the plant from the current state to any other state. Note that deterministic environments are a specific case. Although this discussion will limit itself to discrete problems, most of the points made can be related directly to continuous problems.

A characteristic of many reinforcement learning problems is that one may need to consider a sequence of control actions and their results to determine how to improve the controller. One can examine the

implications of this by associating a *reward* or *cost* with each control action. The error in state in figure B1.5.5 can be thought of as a cost. One can consider the long-term effects of an action formally as the *expected, infinite-horizon discounted cost*:

$$\sum_{t=0}^{t=\infty} \lambda^t c_t$$

where $0 \leq \lambda \leq 1$ is the *discount parameter*, and c_t is the cost of the action taken at time t.

To describe a strategy for picking actions, consider the following approach: for each action u associated with a state i, assign a value $Q(i, u)$. A 'greedy' strategy is to select the action associated with the best Q at every time step. Therefore, an optimum setting for the Q-values is one in which a 'greedy' strategy leads to the minimum expected, infinite-horizon discounted cost. *Q-learning* is a method that yields optimal Q-values in restricted situations. Consider beginning with random settings for each Q-value, and updating each Q-value on-line as follows:

$$Q_{t+1}(i, u_t) = Q_t(i, u)(1 - \alpha) + \alpha \left[c_i(u_t) + \lambda \min Q(j, u_{t+1})\right]$$

where $\min Q(j, u_{t+1})$ is the minimum Q available in state j, which is the state arrived in after action u_t is taken in state i (Barto *et al* 1991, Watkins 1989). The parameter α is a learning rate parameter that is typically set to a small value between zero and one. Arguments based on *dynamic programming* and *Bellman optimality* show that if each state–action pair is tried an infinite number of times, this procedure results in optimal Q-values. Certainly, it is impractical to try every state–action pair an infinite number of times. With finite exploration, Q-values can often be arrived at that are approximately optimal. Regardless of the method employed to update a strategy in a reinforcement learning problem, this exploration–exploitation dilemma always exists.

Another difficulty in the Q-value approach is that it requires storage of a separate Q-value for each state–action pair. In a more practical approach, one could store a Q-value for a group of state–action pairs that share the same characteristics. However, it is not clear how state–action pairs should be grouped. In many ways, the LCS can be thought of as a GA-based technique for grouping state–action pairs.

B1.5.2.3 Learning classifier system introduction

Consider the following method for representing a state–action pair in a reinforcement learning problem: encode a state in binary, and couple it to an action, which is also encoded in binary. In other words, the string

`0 1 1 0 / 0 1 0`

represents one of 16 states and one of eight actions. This string can also be seen as a rule that says 'IF in state `0 1 1 0`, THEN take action `0 1 0`'. In an LCS, such a rule is called a classifier. One can easily associate a Q-value, or other performance measures, with any given classifier.

Now consider generalizing over actions by introducing a 'don't care' character (#) into the state portion of a classifier. In other words, the string

`# 1 1 # / 0 1 0`

is a rule that says 'IF in state `0 1 1 0` OR state `0 1 1 1` OR state `1 1 1 0` OR state `1 1 1 1`, THEN take action `0 1 0`'. The introduction of this generality allows an LCS to represent clusters of states and associated actions. By using the genetic algorithm to search for such strings, one can search for ways of clustering states together, such that they can be assigned joint performance statistics, such as Q values.

Note, however, that Q-learning is not the most common method of credit assignment in LCSs. The most common method is called the *bucket brigade algorithm* for updating a classifier performance statistic called *strength*. Details of the bucket brigade algorithm will be introduced later in this section.

The structure of a typical LCS is shown in figure B1.5.6. This is what is known as a *stimulus–response* LCS, since no internal messages are used as memory. Details of internal message posting in LCSs will be discussed later. In this system, detectors encode state information from an environment into binary messages, which are matched against a list of rules called classifiers. The classifiers used are of the form

IF (condition) THEN (action).

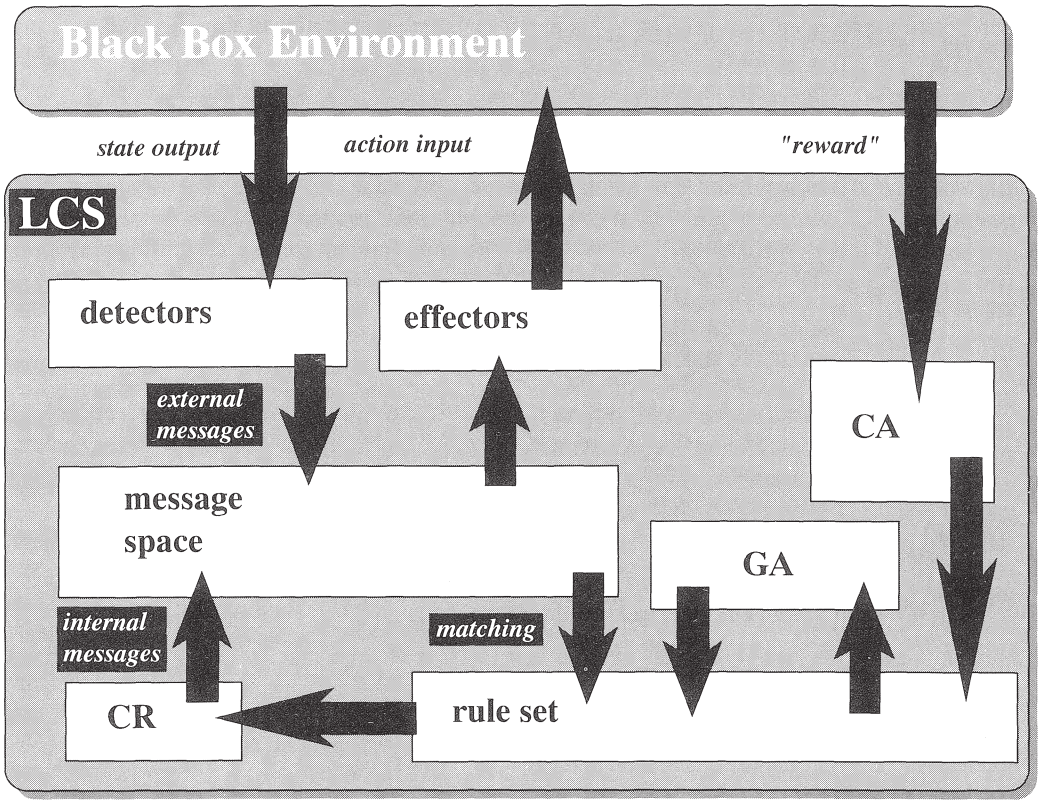


Figure B1.5.6. The structure of an LCS.

The operational cycle of this LCS is:

(i) Detectors post environmental messages on the message list.

(ii) All classifiers are matched against all messages on the message list.

(iii) Fully matched classifiers are selected to act.

(iv) A conflict resolution (CR) mechanism narrows the list of active classifiers to eliminate contradictory actions.

(v) The message list is cleared.

(vi) The CR-selected classifiers post their messages.

(vii) Effectors read the messages from the list, and take appropriate actions in the environment.

(viii) If a reward (or cost) signal is received, it is used by a credit allocation (CA) system to update parameters associated with the individual classifiers (such as the traditional strength measure, Q-like values, or other measures (Booker 1989, Smith 1991)).

B1.5.2.4 'Michigan' and 'Pitt' style learning classifier systems

There are two methods of using the genetic algorithm in LCSs. One is for each genetic algorithm population member to represent an entire set of rules for the problem at hand. This type of LCS is typified by Smith's LS-1 (Smith 1980), which was developed at the University of Pittsburgh. Often, this type of LCS is called the 'Pitt' approach. Another approach is for each genetic algorithm population member to represent a single rule. This type of LCS is typified by the CS-1 of Holland and Reitman (1978), which was developed at the University of Michigan, and is often called the 'Michigan' approach.

In the 'Pitt' approach, crossover and other operators are often employed that change the number of rules in any given population member. The 'Pitt' approach has the advantage of evaluating a complete solution within each genetic algorithm individual. Therefore, the genetic algorithm can converge to a homogeneous population, as in an optimization problem, with the best individual located by the genetic algorithm search acting as the solution. The disadvantage is that each genetic algorithm population member

must be completely evaluated as a rule set. This entails a large computational expense, and may preclude on-line learning in many situations.

In the 'Michigan' approach, one need only evaluate a single rule set, that comprised by the entire population. However, one cannot use the usual genetic algorithm procedures that will converge to a homogeneous population, since one rule is not likely to solve the entire problem. Therefore, one must *coevolve* a set of cooperative rules that jointly solve the problem. This requires a genetic algorithm
C6.1.2 procedure that yields a diverse population at steady state, in a fashion that is similar to *sharing* (Deb and Goldberg 1989, Goldberg and Richardson 1987), or other multimodal genetic algorithm procedures. In some cases simply dividing reward between similar classifiers that fire can yield sharing-like effects (Horn *et al* 1994).

B1.5.2.5 The bucket brigade algorithm (implicit form)

As was noted earlier, the bucket brigade algorithm is the most common form of credit allocation for LCSs. In the bucket brigade, each classifier has a strength, S, which plays a role analogous to a Q-value. The bucket brigade operates as follows:

(i) Classifier A is selected to act at time t.
(ii) Reward r_t is assigned in response to this action.
(iii) Classifier B is selected to act at time $t + 1$.
(iv) The strength of classifier A is updated as follows:

$$S_{\mathrm{A}}^{t+1} = S_{\mathrm{A}}^{t}(1 - \alpha) + \alpha\,[r_t + (\lambda S_{\mathrm{B}})]\,.$$

(v) The algorithm repeats.

Note that this is the *implicit* form of the bucket brigade, first introduced by Wilson (Goldberg 1989, Wilson 1985).

Note that this algorithm is essentially equivalent to Q-learning, but with one important difference. In this case, classifier A's strength is updated with the strength of the classifier that actually acts (classifier B). In Q-learning, the Q-value for the rule at time t is updated with the best Q-valued rule that matches the state at time $t + 1$, whether that rule is selected to act at time $t + 1$ or not. This difference is key to the convergence properties associated with Q-learning. However, it is interesting to note that recent empirical studies have indicated that the bucket brigade (and similar procedures) may be superior to Q-learning in some situations (Rummery and Niranjan 1994, Twardowski 1993).

A wide variety of variations of the bucket brigade exits. Some include a variety of *taxes*, which degrade strength based on the number of times a classifier has matched and fired and the number of generations since the classifier's creation. or other features. Some variations include a variety of methods for using classifier strength in conflict resolution through strength-based *bidding procedures* (Holland *et al* 1986). However, how these techniques fit into the broader context of machine learning, through similar algorithms such as Q-learning, remains a topic of research.

In many LCSs, strength is used as fitness in the genetic algorithm. However, a promising recent study indicates that other measures of classifier utility may be more effective (Wilson 1995).

B1.5.2.6 Internal messages

The LCS discussed to this point has operated entirely in stimulus–response mode. That is, it possesses no internal memory that influences which rule fires. In a more advanced form of the LCS, the action sections of the rule are internal messages that are posted on the message list. Classifiers have a condition that matches environmental messages (those which are posted by the environment) and a condition that matches internal messages (those posted by other classifiers). Some internal messages will cause effectors to fire (causing actions in the environment), and others simply act as internal memory for the LCS.

The operational cycle of a LCS with internal memory is as follows:

(i) Detectors post environmental messages on the message list.
(ii) All classifiers are matched against all messages on the message list.
(iii) Fully matched classifiers are selected to act.
(iv) A conflict resolution (CR) mechanism narrows the list of active classifiers to eliminate contradictory actions, and to cope with restrictions on the number of messages that can be posted.
(v) The message list is cleared.

(vi) The CR-selected classifiers post their messages.

(vii) Effectors read the messages from the list, and take appropriate actions in the environment.

(viii) If a reward (or cost) signal is received, it updates parameters associated with the individual classifiers.

In LCSs with internal messages, the bucket brigade can be used in its original, explicit form. In this form, the next rule that acts is 'linked' to the previous rule through an internal message. Otherwise, the mechanics are similar to those noted above. Once classifiers are linked by internal messages, they can form *rule chains* that express complex sequences of actions.

B1.5.2.7 Parasites

The possibility of rule chains introduced by internal messages, and by 'payback ' credit allocation schemes such as the bucket brigade or Q-learning, also introduces the possibility of rule *parasites*. Simply stated, parasites are rules that obtain fitness through their participation in a rule chain or a sequence of LCS actions, but serve no useful purpose in the problem environment. In some cases, parasite rules can prosper, while actually degrading overall system performance.

A simple example of parasite rules in LCSs is given by Smith (1994). In this study, a simple problem is constructed where the only performance objective is to exploit internal messages as internal memory. Although fairly effective rule sets were evolved in this problem, parasites evolved that exploited the bucket brigade, and the existing rule chains, but that were incorrect for overall system performance. This study speculates that such parasites may be an inevitable consequence in systems that use temporal credit assignment (such as the bucket brigade) and evolve internal memory processing.

B1.5.2.8 Variations of the learning classification system

As was stated earlier, this article only outlines the basic details of the LCS concept. It is important to note that many variations of the LCS exist. These include:

- *Variations in representation and matching procedures.* The {1, 0, #} representation is by no means defining to the LCS approach. For instance, Valenzuela-Rendón (1991) has experimented with a *fuzzy representation* of classifier conditions, actions, and internal messages. Higher-cardinality alphabets are also possible. Other variations include simple changes in the procedures that match classifiers to messages. For instance, sometimes partial matches between messages and classifier conditions are allowed (Booker 1982, 1985). In other systems, classifiers have multiple environmental or internal message conditions. In some suggested variations, multiple internal messages are allowed on the message list at the same time. D2
- *Variations in credit assignment.* As was noted above, a variety of credit assignment schemes can be used in LCSs. The examination of such schemes is the subject of much broader research in the reinforcement learning literature. Alternate schemes for the LCS prominently include *epochal* techniques, where the history of reward (or cost) signals is recorded for some period of time, and classifiers that act during the epoch are updated simultaneously.
- *Variations in discovery operators.* In addition to various versions of the genetic algorithm, LCSs often employ other discovery operators. The most common nongenetic discovery operators are those which create new rules to match messages for which no current rules exist. Such operators are often called *create*, *covering*, or *guessing* operators (Wilson 1985). Other covering operators are suggested that create new rules that suggest actions not accessible in the current rule set (Riolo 1986, Robertson 1988).

B1.5.2.9 Final comments

As was stated in section B1.5.2.1, the LCS remains a concept, more than a specific algorithm. Therefore, some of the details discussed here are necessarily sketchy. However, recent research on the LCS is promising. For a particularly clear examination of a simplified LCS, see a recent article by Wilson (1994). This article also recommends clear avenues for LCS research and development. Interesting LCS *applications* are also appearing in the literature (Smith and Dike 1995). G3.3

Given the robust character of evolutionary computation algorithms, the machine learning techniques suggested by the LCS concept indicate a powerful avenue of future evolutionary computation application.

B1.5.3 Hybrid methods

Zbigniew Michalewicz

Abstract

The concept of a hybrid evolutionary system is introduced here. Included are references to other sections in this handbook in which such hybrid systems are discussed in more detail.

There is some experimental evidence (Davis 1991, Michalewicz 1993) that the enhancement of evolutionary methods by some additional (problem-specific) heuristics, domain knowledge, or existing algorithms can result in a system with outstanding performance. Such enhanced systems are often referred to as *hybrid* evolutionary systems.

Several researchers have recognized the potential of such hybridization of evolutionary systems. Davis (1991, p 56) wrote:

> When I talk to the user, I explain that my plan is to hybridize the genetic algorithm technique and the current algorithm by employing the following three principles:
>
> - *Use the Current Encoding.* Use the current algorithm's encoding technique in the hybrid algorithm.
> - *Hybridize Where Possible.* Incorporate the positive features of the current algorithm in the hybrid algorithm.
> - *Adapt the Genetic Operators.* Create crossover and mutation operators for the new type of encoding by analogy with bit string crossover and mutation operators. Incorporate domain-based heuristics as operators as well.
>
> [...] I use the term *hybrid genetic algorithm* for algorithms created by applying these three principles.

The above three principles emerged as a result of countless experiments of many researchers, who tried to 'tune' their evolutionary algorithms to some problem at hand, that is, to create 'the best' algorithm for a particular class of problems. For example, during the last 15 years, various application-specific variations of evolutionary algorithms have been reported (Michalewicz 1996); these variations included variable-length strings (including strings whose elements were *if–then–else* rules), richer structures than binary strings, and experiments with modified genetic operators to meet the needs of particular applications. Some researchers (e.g. Grefenstette 1987) experimented with incorporating problem-specific knowledge into the initialization routine of an evolutionary system; if a (fast) heuristic algorithm provides individuals of the initial population for an evolutionary system, such a hybrid evolutionary system is guaranteed to do no worse than the heuristic algorithm which was used for the initialization.

Usually there exist several (better or worse) heuristic algorithms for a given problem. Apart from incorporating them for the purpose of initialization, some of these algorithms transform one solution into another by imposing a change in the solution's encoding (e.g. 2-opt step for the *traveling salesman problem*). One can incorporate such transformations into the operator set of evolutionary system, which usually is a very useful addition (see Chapter D3).

G9.5
D3

Note also (see Sections C1.1 and C3.1) that there is a strong relationship between encodings of individuals in the population and operators, hence the operators of any evolutionary system must be chosen carefully in accordance with the selected representation of individuals. This is a responsibility of the developer of the system; again, we would cite Davis (1991, p 58):

C1.1, C3.1

> Crossover operators, viewed in the abstract are operators that combine subparts of two parent chromosomes to produce new children. The adopted encoding technique should support operators of this type, but it is up to you to combine your understanding of the problem, the encoding technique, and the function of crossover in order to figure out what those operators will be. [...]
>
> The situation is similar for mutation operators. We have decided to use an encoding technique that is tailored to the problem domain; the creators of the current algorithm have done this tailoring for us. Viewed in the abstract, a mutation operator is an operator that introduces variations into the chromosome. [...] these variations can be global or local, but they are critical to keeping the genetic pot boiling. You will have to combine your knowledge of the problem,

the encoding technique, and the function of mutation in a genetic algorithm to develop one or more mutation operators for the problem domain.

Very often hybridization techniques make use of local search operators, which are considered as 'intelligent mutations'. For example, the best evolutionary algorithms for the traveling salesman problem use 2-opt or 3-opt procedures to improve the individuals in the population (see e.g. Mühlenbein *et al* 1988). It is not unusual to incorporate gradient-based (or hill-climbing) methods as ways for a local improvement of individuals. It is also not uncommon to combine *simulated annealing* techniques with some evolutionary algorithms (Adler 1993). D3.5

The class of hybrid evolutionary algorithms described so far consists of systems which extend evolutionary paradigm by incorporating additional features (local search, problem-specific representations and operators, and the like). This class also includes also so-called morphogenic evolutionary techniques (Angeline 1995), which include mappings (development functions) between representations that evolve (i.e. evolved representations) and representations which constitutes the input for the evaluation function (i.e. evaluated representations). However, there is another class of evolutionary hybrid methods, where the evolutionary algorithm acts as a separate component of a larger system. This is often the case for various scheduling systems, where the evolutionary algorithm is just responsible for ordering particular items (see, for example, Section F1.5). This is also the case for fuzzy systems, where the evolutionary algorithms may control the membership function (see Chapter D2), or of neural systems, where evolutionary algorithms may optimize the topology or weights of the network (see Chapter D1). F1.5 D2 D1

In this handbook there are several articles which refer (in a more or less explicit way) to the above classes of hybridization. In particular, Chapter C1 describes various representations, Chapter C3 appropriate operators for these representations, and Chapter D3 hybridizations of evolutionary methods with other optimization methods, whereas Chapters D1 and D2 provide an overview of neural–evolutionary and fuzzy–evolutionary systems, respectively. Also, many articles in Part G (Evolutionary Computation in Practice) describe evolutionary systems with hybrid components: it is apparent that hybridization techniques have generated a number of successful optimization algorithms. C1, C3 D3 D1, D2

References

Adler D 1993 Genetic algorithms and simulated annealing: a marriage proposal *Proc. IEEE Int. Conf. on Neural Networks* pp 1104–9

Angeline P J 1995 Morphogenic evolutionary computation: introduction, issues, and examples *Proc. 4th Ann. Conf. on Evolutionary Programming (San Diego, CA, March 1995)* ed J R McDonnell, R G Reynolds and D B Fogel (Cambridge, MA: MIT Press) pp 387–401

——1996 Two self-adaptive crossover operators for genetic programming *Advances in Genetic Programming 2* ed P J Angeline and K E Kinnear Jr (Cambridge, MA: MIT Press)

Angeline P J and Pollack J B 1993 Competitive environments evolve better solutions for complex tasks *Proc. 5th Int. Conf. on Genetic Algorithms (Urbana-Champaign, IL, July 1993)* ed S Forrest (San Mateo, CA: Morgan Kaufmann)

Barto A G 1990 *Some Learning Tasks from a Control Perspective* COINS Technical Report 90-122, University of Massachusetts

Barto A G, Bradtke S J and Singh S P 1991 *Real-time Learning and Control using Asynchronous Dynamic Programming* COINS Technical Report 91-57, University of Massachusetts

Booker L B 1982 Intelligent behavior as an adaptation to the task environment *Dissertations Abstracts Int.* **43** 469B; University Microfilms 8214966

——1985 Improving the performance of genetic algorithms in classifier systems *Proc. Int. Conf. on Genetic Algorithms and Their Applications* pp 80–92

——1989 Triggered rule discovery in classifier systems *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 265–74

Cramer N L 1985 A representation of the adaptive generation of simple sequential programs *Proc. 1st Int. Conf. on Genetic Algorithms (Pittsburgh, PA, July 1985)* ed J J Grefenstette (Hillsdale, NJ: Erlbaum)

Davis L (ed) 1987 *Genetic Algorithms and Simulated Annealing* (Los Altos, CA: Morgan Kaufmann)

——1991 *Handbook of Genetic Algorithms* (New York: Van Nostrand Reinhold)

Deb K and Goldberg D E 1989 An investigation of niche and species formation in genetic function optimization *Proc. 3rd Int. Conf. on Genetic Algorithms (Fairfax, VA, June 1989)* ed J D Schaffer (San Mateo, CA: Morgan Kaufmann) pp 42–50

D'haeseleer P 1994 Context preserving crossover in genetic programming *1st IEEE Conf. on Evolutionary Computation (Orlando, FL, June 1994)* (Piscataway, NJ: IEEE)