**Natural Computing Series**

Anthony Brabazon Michael O'Neill Seán McGarraghy Natural Computing Algorithms

**Natural Computing Series**

Series Editors: G. Rozenberg Th. Bäck A.E. Eiben J.N. Kok H.P. Spaink

Leiden Center for Natural Computing

Advisory Board: S. Amari G. Brassard K.A. De Jong C.C.A.M. Gielen T. Head L. Kari L. Landweber T. Martinetz Z. Michalewicz M.C. Mozer E. Oja G. Paun ̆

J. Reif H. Rubin A. Salomaa M. Schoenauer H.-P. Schwefel C. Torras D. Whitley E. Winfree J.M. Zurada

More information about this series at h ttp://www.springer.com/series/ 4190

Anthony Brabazon • Michael O’Neill Seán McGarraghy

Natural Computing Algorithms

Anthony Brabazon Michael O’Neill Natural Computing Research

& Applications Group School of Business University College Dublin Dublin, Ireland

Seán McGarraghy

*Series Editors* G. Rozenberg (Managing Editor)

Th. Bäck, J.N. Kok, H.P. Spaink Leiden Center for Natural Computing Leiden University Leiden, The Netherlands

A.E. Eiben VU University Amsterdam The Netherlands

Natural Computing Research

& Applications Group School of Business University College Dublin Dublin, Ireland

Natural Computing Research

& Applications Group School of Business University College Dublin Dublin, Ireland

ISSN 1619-7127 Natural Computing Series ISBN 978-3-662-43630-1 ISBN 978-3-662-43631-8 (eBook) DOI 10.1007/978-3-662-43631-8

Library of Congress Control Number: 20159 5

1433

Springer Heidelberg New York Dordrecht London © Springer-Verlag Berlin Heidelberg 2015 This work is subject to copyright. All rights are reserved by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed. The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use. The publisher, the authors and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, express or implied, with respect to the material contained herein or for any errors or omissions that may have been made.

Printed on acid-free paper

Springer-Verlag GmbH Berlin Heidelberg is part of Springer Science+Business Media (www.springer.com)

To Maria, Kevin and Rose Tony

To Gr ́ainne, Aoife, Michael, Caoimhe, my father John & to the memory of my mother Jane Mike

To Milena, Martin and Alex, and my mother Mary; and to the memory of my father Michael Seán

**Preface**

The field of natural computing has been the focus of a substantial research ef- fort in recent decades. One particular strand of this concerns the development of computational algorithms using metaphorical inspiration from systems and phenomena that occur in the natural world. These naturally inspired comput- ing algorithms have proven to be successful problem solvers across domains as varied as management science, telecommunications, business analytics, bioin- formatics, finance, marketing, engineering, architecture and design, to name but a few. This book provides a comprehensive introduction to natural com- puting algorithms.

The book is divided into eight main parts, each of which provides an inte- grated discussion of a range of related natural computing algorithms. The first part covers a family of algorithms which are inspired by processes of evolution (evolutionary computing) as well as introducing pivotal concepts in the design of natural computing algorithms such as choice of representation, diversity generation mechanisms, and the selection of an appropriate fitness function. The second part illustrates a selection of algorithms which are inspired by the social behaviour of individuals (social computing) ranging from flocking behaviours to the food foraging behaviours of several organisms, including bats, insects and bacteria. The third part introduces a number of algorithms whose workings are inspired by the operation of our central nervous system (neurocomputing). The fourth part of the book discusses optimisation and classification algorithms which are metaphorically derived from the workings of our immune system (immunocomputing). The fifth part provides an intro- duction to developmental and grammatical computing, where the creation of a model or structure results from a development process which is typically governed by a set of rules, a ‘grammar’. Physical computing is described in the sixth part of the book, with the primary emphasis being placed on the use of quantum inspired representations in natural computing. Two emerging paradigms in natural computing, chemical computing and plant inspired al- gorithms are introduced in part seven of the book. The closing chapter of the book looks towards the future of natural computing algorithms.

VII

VIII Preface

Of course, given the diverse range of natural computing algorithms which have been developed, we have had to make difficult decisions as to which algorithms to include and which to omit from our coverage. We have generally focussed on the better known algorithms in each part, supplemented by some less known algorithms which we personally found interesting!

This book will be of particular interest to academics and practitioners in computer science, informatics, management science and other application domains who wish to learn more about natural computing. The book is also a suitable accompaniment for a course on natural computing and could be used with graduate and final-year undergraduate students. More generally, the book will be of interest to anyone interested in natural computing approaches to optimisation, clustering, classification, prediction and model induction.

In order to make the book as accessible as possible, no prior knowledge of natural computing is assumed; nor do we assume that the reader has an extensive background in mathematics, optimisation or statistics. In discussing each family of algorithms we concentrate on providing a clear description of the main components of each algorithm and we also include material on the relevant natural background so that readers can fully appreciate the hidden computation that takes place in nature. As the book is intended to provide an introduction to a wide range of algorithms we do not focus on an exhaus- tive theoretical discussion concerning each of the algorithmic families, as a comprehensive discussion of any of them would require a book in its own right.

Anthony Brabazon Michael O’Neill Dublin, March 2015 Seán McGarraghy

**Acknowledgment**

When first conceived, we had little idea that the project of writing this book would take nearly eight years to complete. Over the course of that time, the field of natural computing has seen multiple advances and many people have consequently influenced this work. In particular, we would like to acknowledge a number of people who have directly, and indirectly through their encour- agement, contributed to this project.

The greatest single influence has come from our students over the years, both those taking courses that we have taught, and our graduate research stu- dents. Students on the Natural Computing (COMP30290), Natural Comput- ing Applications (COMP41190), Numerical Analytics and Software (MIS40530), Network Software Modelling (MIS40550), Data Mining & Applications (MIS40970), and Advanced Specialist Course (Finance) (FIN40960) modules at University College Dublin helped to road test material from the book and many changes were made as a result of their feedback.

The countless conversations and interactions with members of the UCD Natural Computing Research & Applications Group (http://ncra.ucd.ie) have continually challenged and informed our understanding of the world of natural computing and we wish to thank all members (past and present) of this amazing team of researchers who have kept us inspired.

We would also like to thank the School of Business at University College Dublin (http://www.smurfitschool.ie/), and our colleagues in the School, for their support during this project. The Complex & Adaptive Systems Lab- oratory (http://casl.ucd.ie) at University College Dublin also provided support for this project by fostering the interdisciplinary environment in which our group collaborates and through the provision of physical space for our research team. Anthony Brabazon and Michael O’Neill also acknowl- edge the support of their research activities provided by Science Foundation Ireland (Grant number 08/SRC/FM1389 – Financial Mathematics and Com- putation Research Cluster, Grant number 08/IN.1/I1868 – Evolution in Dy- namic Environments with Grammatical Evolution (EDGE), and Grant num- ber 13/IA/1850 – Applications of Evolutionary Design (AppED)).

IX

X Acknowledgment

We are very grateful to the anonymous referees for their thorough review of drafts of this book and for their insightful, thought provoking comments. They have notably strengthened this work. For their proof reading, we acknowledge the assistance of Drs. Eoin Murphy, Miguel Nicolau and James McDermott, and we also acknowledge the assistance of Dr. Wei Cui for the production of some of the diagrams in the book. Despite the valiant efforts of all the proof readers and ourselves, errors and omissions will undoubtedly occur, and we accept responsibility for these.

We thank Diego Perez and Miguel Nicolau for their research on the ap- plication of Grammatical Evolution (GE) to the Mario AI challenge, which was adapted as a working example of a GE mapping in Chap. 19. We thank Margaret O’Connor for sharing her knowledge on plants, and Owen, Niall and Peggy for listening and keeping Mike sane and firmly grounded at all times. Thanks also to those who in their various ways encouraged our journey into natural computing including Eamonn Walsh and Robin Matthews.

We especially extend our thanks to Ronan Nugent of Springer-Verlag for his on-going support of this project, for his advice on drafts of the manuscript, and especially for his patience with a project that took far longer than we originally planned.

We each extend a special thank you to our families. Without your love, support, patience, and understanding, this project would never have been completed.

Anthony Brabazon Michael O’Neill Seán McGarraghy

**Contents**

**1 Introduction** ............................................... 1 1.1 Natural Computing Algorithms: An Overview . . . . . . . . . . . . . . . 2 1.1.1 Biologically Inspired Algorithms . . . . . . . . . . . . . . . . . . . . . 2 1.1.2 Families of Naturally Inspired Algorithms . . . . . . . . . . . . 9 1.1.3 Physically Inspired Algorithms . . . . . . . . . . . . . . . . . . . . . . 10 1.1.4 Plant Inspired Algorithms . . . . . . . . . . . . . . . . . . . . . . . . . . 11 1.1.5 Chemically Inspired Algorithms . . . . . . . . . . . . . . . . . . . . . 11 1.1.6 A Unified Family of Algorithms . . . . . . . . . . . . . . . . . . . . . 11 1.1.7 How Much Natural Inspiration? . . . . . . . . . . . . . . . . . . . . . 12 1.2 Structure of the Book . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

**Part I Evolutionary Computing**

**2 Introduction to Evolutionary Computing** .................. 17 2.1 Evolutionary Algorithms . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18

**3 Genetic Algorithm**......................................... 21 3.1 Canonical Genetic Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 21 3.1.1 A Simple GA Example . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23 3.2 Design Choices in Implementing a GA . . . . . . . . . . . . . . . . . . . . . 24 3.3 Choosing a Representation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25 3.3.1 Genotype to Phenotype Mapping . . . . . . . . . . . . . . . . . . . . 26 3.3.2 Genotype Encodings . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26 3.3.3 Representation Choice and the Generation of Diversity . 28 3.4 Initialising the Population . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29 3.5 Measuring Fitness . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29 3.6 Generating Diversity . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31 3.6.1 Selection Strategy . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31 3.6.2 Mutation and Crossover . . . . . . . . . . . . . . . . . . . . . . . . . . . . 35 3.6.3 Replacement Strategy . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 39XI

XII Contents

3.7 Choosing Parameter Values . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40 3.8 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41

**4 Extending the Genetic Algorithm**.......................... 43 4.1 Dynamic Environments . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 43 4.1.1 Strategies for Dynamic Environments . . . . . . . . . . . . . . . . 44 4.1.2 Diversity . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 44 4.2 Structured Population GAs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48 4.3 Constrained Optimisation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 50 4.4 Multiobjective Optimisation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 53 4.5 Memetic Algorithms . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 57 4.6 Linkage Learning . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 59 4.7 Estimation of Distribution Algorithms . . . . . . . . . . . . . . . . . . . . . 61 4.7.1 Population-Based Incremental Learning . . . . . . . . . . . . . . 62 4.7.2 Univariate Marginal Distribution Algorithm . . . . . . . . . . 63 4.7.3 Compact Genetic Algorithm . . . . . . . . . . . . . . . . . . . . . . . . 65 4.7.4 Bayesian Optimisation Algorithm . . . . . . . . . . . . . . . . . . . 66 4.8 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 71

**5 Evolution Strategies and Evolutionary Programming** ...... 73 5.1 The Canonical ES Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 74 5.1.1 (1+1)-ES ...................................... 74 5.1.2 (μ + λ)-ES and (μ, λ)-ES .......................... 75 5.1.3 Mutation in ES . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 75 5.1.4 Adaptation of the Strategy Parameters . . . . . . . . . . . . . . . 76 5.1.5 Recombination . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 78 5.2 Evolutionary Programming . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 80 5.3 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 82

**6 Differential Evolution** ...................................... 83 6.1 Canonical Differential Evolution Algorithm . . . . . . . . . . . . . . . . . 83 6.2 Extending the Canonical DE Algorithm . . . . . . . . . . . . . . . . . . . . 88 6.2.1 Selection of the Base Vector . . . . . . . . . . . . . . . . . . . . . . . . 88 6.2.2 Number of Vector Differences . . . . . . . . . . . . . . . . . . . . . . . 88 6.2.3 Alternative Crossover Rules . . . . . . . . . . . . . . . . . . . . . . . . 89 6.2.4 Other DE Variants . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 89 6.3 Discrete DE . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 90 6.4 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 92

**7 Genetic Programming** ..................................... 95 7.1 Genetic Programming . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 95 7.1.1 GP Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 97 7.1.2 Function and Terminal Sets . . . . . . . . . . . . . . . . . . . . . . . . . 98 7.1.3 Initialisation Strategy . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 100 7.1.4 Diversity-Generation in GP . . . . . . . . . . . . . . . . . . . . . . . . . 102

**Part II Social Computing**

**8 Particle Swarm Algorithms** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 117 8.1 Social Search . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 118 8.2 Particle Swarm Optimisation Algorithm . . . . . . . . . . . . . . . . . . . . 118 8.2.1 Velocity Update . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 120 8.2.2 Velocity Control . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 123 8.2.3 Neighbourhood Structure . . . . . . . . . . . . . . . . . . . . . . . . . . . 124 8.3 Comparing PSO and Evolutionary Algorithms . . . . . . . . . . . . . . 125 8.4 Maintaining Diversity in PSO . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 127 8.4.1 Simple Approaches to Maintaining Diversity . . . . . . . . . . 129 8.4.2 Predator–Prey PSO . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 130 8.4.3 Charged Particle Swarm . . . . . . . . . . . . . . . . . . . . . . . . . . . . 132 8.4.4 Multiple Swarms . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 134 8.4.5 Speciation-Based PSO . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 135 8.5 Hybrid PSO Algorithms . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 136 8.6 Discrete PSO . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 137 8.6.1 BinPSO . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 137 8.6.2 Angle-Modulated PSO . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 138 8.7 Evolving a PSO Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 139 8.8 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 139

**9 Ant Algorithms**. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 141 9.1 A Taxonomy of Ant Algorithms . . . . . . . . . . . . . . . . . . . . . . . . . . . 142 9.2 Ant Foraging Behaviours . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 142 9.3 Ant Algorithms for Discrete Optimisation . . . . . . . . . . . . . . . . . . 144 9.3.1 Graph structure . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 144 9.3.2 Ant System . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 147 9.3.3 MAX-MIN Ant System . . . . . . . . . . . . . . . . . . . . . . . . . . 151 9.3.4 Ant Colony System . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 152

Contents XIII

7.2 Bloat in GP . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 105 7.3 More Complex GP Architectures . . . . . . . . . . . . . . . . . . . . . . . . . . 105 7.3.1 Functions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 105 7.3.2 ADF Mutation and Crossover . . . . . . . . . . . . . . . . . . . . . . . 108 7.3.3 Memory . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 108 7.3.4 Looping . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 109 7.3.5 Recursion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 111 7.4 GP Variants . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 112 7.4.1 Linear and Graph GP . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 112 7.4.2 Strongly Typed GP . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 112 7.4.3 Grammar-Based GP . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 112 7.5 Semantics and GP . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 113 7.6 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 113

XIV Contents

9.3.5 Ant Multitour Systems . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 153 9.3.6 Dynamic Optimisation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 154 9.4 Ant Algorithms for Continuous Optimisation. . . . . . . . . . . . . . . . 155 9.5 Multiple Ant Colonies . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 157 9.6 Hybrid Ant Foraging Algorithms . . . . . . . . . . . . . . . . . . . . . . . . . . 159 9.7 Ant-Inspired Clustering Algorithms . . . . . . . . . . . . . . . . . . . . . . . . 160 9.7.1 Deneubourg Model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 161 9.7.2 Lumer and Faieta Model . . . . . . . . . . . . . . . . . . . . . . . . . . . 162 9.7.3 Critiquing Ant Clustering . . . . . . . . . . . . . . . . . . . . . . . . . . 166 9.8 Classification with Ant Algorithms . . . . . . . . . . . . . . . . . . . . . . . . 167 9.9 Evolving an Ant Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 169 9.10 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 170

**10 Other Foraging Algorithms** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 171 10.1 Honeybee Dance Language . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 171 10.2 Honeybee Foraging . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 172 10.2.1 The Honeybee Recruitment Dance . . . . . . . . . . . . . . . . . . . 172 10.3 Designing a Honeybee Foraging Optimisation Algorithm . . . . . . 173 10.3.1 Bee System Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 174 10.3.2 Artificial Bee Colony Algorithm . . . . . . . . . . . . . . . . . . . . . 175 10.3.3 Honeybee Foraging and Dynamic Environments . . . . . . . 178 10.4 Bee Nest Site Selection . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 180 10.4.1 Bee Nest Site Selection Optimisation Algorithm . . . . . . . 182 10.5 Honeybee Mating Optimisation Algorithm . . . . . . . . . . . . . . . . . . 184 10.6 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 186

**11 Bacterial Foraging Algorithms** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 187 11.1 Bacterial Behaviours . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 187 11.1.1 Quorum Sensing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 187 11.1.2 Sporulation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 188 11.1.3 Mobility. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 188 11.2 Chemotaxis in E. Coli Bacteria . . . . . . . . . . . . . . . . . . . . . . . . . . . . 189 11.3 Bacterial Foraging Optimisation Algorithm . . . . . . . . . . . . . . . . . 190 11.3.1 Basic Chemotaxis Model . . . . . . . . . . . . . . . . . . . . . . . . . . . 191 11.3.2 Chemotaxis Model with Social Communication . . . . . . . . 192 11.4 Dynamic Environments . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 198 11.5 Classification Using a Bacterial Foraging Metaphor . . . . . . . . . . 198 11.6 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 199

**12 Other Social Algorithms** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 201 12.1 Glow Worm Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 201 12.2 Bat Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 206 12.2.1 Bat Vocalisations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 206 12.2.2 Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 207 12.2.3 Discussion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 210

**Part III Neurocomputing**

**13 Neural Networks for Supervised Learning** . . . . . . . . . . . . . . . . . 221 13.1 Biological Inspiration for Neural Networks . . . . . . . . . . . . . . . . . . 221 13.2 Artificial Neural Networks . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 222 13.2.1 Neural Network Architectures . . . . . . . . . . . . . . . . . . . . . . . 222 13.3 Structure of Supervised Neural Networks . . . . . . . . . . . . . . . . . . . 224 13.3.1 Activation and Transfer Functions . . . . . . . . . . . . . . . . . . . 226 13.3.2 Universal Approximators . . . . . . . . . . . . . . . . . . . . . . . . . . . 228 13.4 The Multilayer Perceptron . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 228 13.4.1 MLP Transfer Function . . . . . . . . . . . . . . . . . . . . . . . . . . . . 230 13.4.2 MLP Activation Function . . . . . . . . . . . . . . . . . . . . . . . . . . 230 13.4.3 The MLP Projection Construction and Response

Regions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 231 13.4.4 Relationship of MLPs to Regression Models. . . . . . . . . . . 233 13.4.5 Training an MLP . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 234 13.4.6 Overtraining . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 237 13.4.7 Practical Issues in Modelling with and Training MLPs . 239 13.4.8 Stacking MLPs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 243 13.4.9 Recurrent Networks . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 244 13.5 Radial Basis Function Networks . . . . . . . . . . . . . . . . . . . . . . . . . . . 246 13.5.1 Kernel Functions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 246 13.5.2 Radial Basis Functions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 247 13.5.3 Intuition Behind Radial Basis Function Networks . . . . . . 248 13.5.4 Properties of Radial Basis Function Networks . . . . . . . . . 249 13.5.5 Training Radial Basis Function Networks . . . . . . . . . . . . . 250 13.5.6 Developing a Radial Basis Function Network . . . . . . . . . . 251 13.6 Support Vector Machines . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 252 13.6.1 SVM Method . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 258 13.6.2 Issues in Applications of SVM . . . . . . . . . . . . . . . . . . . . . . 258 13.7 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 259

**14 Neural Networks for Unsupervised Learning** . . . . . . . . . . . . . . . 261 14.1 Self-organising Maps . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 262 14.2 SOM Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 264 14.3 Implementing a SOM Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . 266 14.4 Classification with SOMs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 271

Contents XV

12.3 Fish School Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 211 12.3.1 Fish School Search . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 212 12.3.2 Summary. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 214 12.4 Locusts . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 215 12.4.1 Locust Swarm Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . 216 12.5 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 218

XVI Contents

14.5 Self-organising Swarm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 272 14.6 SOSwarm and SOM . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 275 14.7 Adaptive Resonance Theory . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 276 14.7.1 Unsupervised Learning for ART . . . . . . . . . . . . . . . . . . . . . 277 14.7.2 Supervised Learning for ARTs . . . . . . . . . . . . . . . . . . . . . . 279 14.7.3 Weaknesses of ART Approaches . . . . . . . . . . . . . . . . . . . . . 279 14.8 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 279

**15 Neuroevolution** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 281 15.1 Direct Encodings . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 282 15.1.1 Evolving Weight Vectors . . . . . . . . . . . . . . . . . . . . . . . . . . . 282 15.1.2 Evolving the Selection of Inputs . . . . . . . . . . . . . . . . . . . . . 283 15.1.3 Evolving the Connection Structure . . . . . . . . . . . . . . . . . . 283 15.1.4 Other Hybrid MLP Algorithms . . . . . . . . . . . . . . . . . . . . . 286 15.1.5 Problems with Direct Encodings . . . . . . . . . . . . . . . . . . . . 287 15.2 NEAT . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 289 15.2.1 Representation in NEAT . . . . . . . . . . . . . . . . . . . . . . . . . . . 290 15.2.2 Diversity Generation in NEAT . . . . . . . . . . . . . . . . . . . . . . 291 15.2.3 Speciation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 292 15.2.4 Incremental Evolution . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 296 15.3 Indirect Encodings. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 297 15.4 Other Hybrid Neural Algorithms . . . . . . . . . . . . . . . . . . . . . . . . . . 297 15.5 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 297

**Part IV Immunocomputing**

**16 Artificial Immune Systems**. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 301 16.1 The Natural Immune System . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 302 16.1.1 Components of the Natural Immune System . . . . . . . . . . 302 16.1.2 Innate Immune System . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 302 16.1.3 Adaptive Immune System . . . . . . . . . . . . . . . . . . . . . . . . . . 304 16.1.4 Danger Theory . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 309 16.1.5 Immune Network Theory . . . . . . . . . . . . . . . . . . . . . . . . . . . 309 16.1.6 Optimal Immune Defence . . . . . . . . . . . . . . . . . . . . . . . . . . 310 16.2 Artificial Immune Algorithms . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 310 16.3 Negative Selection Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 310 16.4 Dendritric Cell Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 315 16.5 Clonal Expansion and Selection Inspired Algorithms . . . . . . . . . 320 16.5.1 CLONALG Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 320 16.5.2 B Cell Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 322 16.5.3 Real-Valued Clonal Selection Algorithm . . . . . . . . . . . . . . 323 16.5.4 Artificial Immune Recognition System . . . . . . . . . . . . . . . 325 16.6 Immune Programming . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 330 16.7 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 331

**Part V Developmental and Grammatical Computing**

**17 An Introduction to Developmental and Grammatical**

**Computing** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 335 17.1 Developmental Computing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 335 17.2 Grammatical Computing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 336 17.3 What Is a Grammar? . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 337 17.3.1 Types of Grammar . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 338 17.3.2 Formal Grammar Notation . . . . . . . . . . . . . . . . . . . . . . . . . 340 17.4 Grammatical Inference . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 341 17.5 Lindenmayer Systems . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 341 17.6 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 343

**18 Grammar-Based and Developmental Genetic**

**Programming** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 345 18.1 Grammar-Guided Genetic Programming . . . . . . . . . . . . . . . . . . . . 346 18.1.1 Other Grammar-Based Approaches to GP . . . . . . . . . . . . 351 18.2 Developmental GP . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 351 18.2.1 Genetic L-System Programming . . . . . . . . . . . . . . . . . . . . . 351 18.2.2 Binary GP . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 352 18.2.3 Cellular Encoding . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 354 18.2.4 Analog Circuits . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 354 18.2.5 Other Developmental Approaches to GP . . . . . . . . . . . . . 354 18.3 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 356

**19 Grammatical Evolution** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 357 19.1 A Primer on Gene Expression . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 358 19.2 Extending the Biological Analogy to GE . . . . . . . . . . . . . . . . . . . 360 19.3 Example GE Mapping . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 361 19.4 Search Engine . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 368 19.4.1 Genome Encoding . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 368 19.4.2 Mutation and Crossover Search Operators . . . . . . . . . . . . 368 19.4.3 Modularity . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 370 19.4.4 Search Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 371 19.5 Genotype–Phenotype Map . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 372 19.6 Grammars . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 372 19.7 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 373

**20 Tree-Adjoining Grammars and Genetic Programming**

. . . . . . 375 20.1 Tree-Adjoining Grammars . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 377 20.2 TAG3P . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 377 20.3 Developmental TAG3P . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 379 20.4 TAGE . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 379 20.5 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 381

Contents XVII

XVIII Contents

**21 Genetic Regulatory Networks**. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 383 21.1 Artificial Gene Regulatory Model for Genetic Programming . . . 383 21.1.1 Model Output . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 385 21.2 Differential Gene Expression . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 386 21.3 Artificial GRN for Image Compression . . . . . . . . . . . . . . . . . . . . . 389 21.4 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 389

**Part VI Physical Computing**

**22 An Introduction to Physically Inspired Computing** . . . . . . . . 393 22.1 A Brief Physics Primer . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 393 22.1.1 A Rough Taxonomy of Modern Physics . . . . . . . . . . . . . . 393 22.2 Classical Mechanics . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 395 22.2.1 Energy and Momentum . . . . . . . . . . . . . . . . . . . . . . . . . . . . 395 22.2.2 The Hamiltonian . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 396 22.3 Thermodynamics . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 398 22.3.1 Statistical Mechanics . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 400 22.3.2 Ergodicity . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 402 22.4 Quantum Mechanics . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 402 22.4.1 Observation in Quantum Mechanics . . . . . . . . . . . . . . . . . 403 22.4.2 Entanglement and Decoherence . . . . . . . . . . . . . . . . . . . . . 404 22.4.3 Noncommuting Operators . . . . . . . . . . . . . . . . . . . . . . . . . . 405 22.4.4 Tunnelling . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 406 22.4.5 Quantum Statistical Mechanics . . . . . . . . . . . . . . . . . . . . . 406 22.5 Quantum Computing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 407 22.5.1 Two-State Systems and Qubits . . . . . . . . . . . . . . . . . . . . . . 407 22.5.2 Digital Quantum Computers . . . . . . . . . . . . . . . . . . . . . . . . 408 22.5.3 Quantum Information . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 409 22.5.4 Adiabatic Quantum Computation . . . . . . . . . . . . . . . . . . . 410 22.6 Annealing and Spin Glasses . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 411 22.6.1 Ising Spin Glasses . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 412 22.6.2 Quantum Spin Glasses . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 414 22.7 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 415

**23 Physically Inspired Computing Algorithms** . . . . . . . . . . . . . . . . 417 23.1 Simulated Annealing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 417 23.1.1 Search and Neighbourhoods . . . . . . . . . . . . . . . . . . . . . . . . 419 23.1.2 Acceptance of ‘Bad’ Moves . . . . . . . . . . . . . . . . . . . . . . . . . 419 23.1.3 Parameterisation of SA . . . . . . . . . . . . . . . . . . . . . . . . . . . . 420 23.1.4 Extensions of SA . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 421 23.1.5 Concluding Remarks . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 421 23.2 Simulated Quantum Annealing . . . . . . . . . . . . . . . . . . . . . . . . . . . . 422 23.2.1 Implementation of SQA . . . . . . . . . . . . . . . . . . . . . . . . . . . . 424 23.2.2 SQA Application to TSP-Type Problems . . . . . . . . . . . . . 424

Contents XIX

23.3 Constrained Molecular Dynamics Algorithm . . . . . . . . . . . . . . . . 426 23.4 Physical Field Inspired Algorithms. . . . . . . . . . . . . . . . . . . . . . . . . 429 23.4.1 Central Force Optimisation . . . . . . . . . . . . . . . . . . . . . . . . . 429 23.4.2 Gravitational Search Algorithm and Variants . . . . . . . . . 431 23.4.3 Differences Among Physical Field-Inspired Algorithms . 435 23.5 Extremal Optimisation Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . 436 23.6 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 437

**24 Quantum Inspired Evolutionary Algorithms** . . . . . . . . . . . . . . . 439 24.1 Qubit Representation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 439 24.2 Quantum Inspired Evolutionary Algorithms (QIEAs) . . . . . . . . 440 24.3 Binary-Valued QIEA . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 440 24.3.1 Diversity Generation in Binary QIEA . . . . . . . . . . . . . . . . 443 24.4 Real-Valued QIEA . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 446 24.4.1 Initialising the Quantum Population . . . . . . . . . . . . . . . . . 446 24.4.2 Observing the Quantum Chromosomes . . . . . . . . . . . . . . . 447 24.4.3 Crossover Mechanism . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 448 24.4.4 Updating the Quantum Chromosomes . . . . . . . . . . . . . . . 449 24.5 QIEAs and EDAs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 449 24.6 Other Quantum Hybrid Algorithms . . . . . . . . . . . . . . . . . . . . . . . . 450 24.7 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 452

**Part VII Other Paradigms**

**25 Plant-Inspired Algorithms** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 455 25.1 Plant Behaviours . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 455 25.2 Foraging . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 456 25.2.1 Plant Movement and Foraging . . . . . . . . . . . . . . . . . . . . . . 457 25.2.2 Root Foraging . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 459 25.2.3 Predatory Plants . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 461 25.3 Plant-Level Coordination . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 462 25.4 A Taxonomy of Plant-Inspired Algorithms . . . . . . . . . . . . . . . . . . 464 25.5 Plant Propagation Algorithms . . . . . . . . . . . . . . . . . . . . . . . . . . . . 464 25.5.1 Invasive Weed Optimisation Algorithm . . . . . . . . . . . . . . . 464 25.5.2 Paddy Field Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 467 25.5.3 Strawberry Plant Algorithm . . . . . . . . . . . . . . . . . . . . . . . . 468 25.6 Plant Growth Simulation Algorithm . . . . . . . . . . . . . . . . . . . . . . . 469 25.6.1 The Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 472 25.6.2 Variants on the Plant Growth Simulation Algorithm . . . 474 25.7 Root-Swarm Behaviour . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 475 25.7.1 Modelling Root Growth in Real Plants . . . . . . . . . . . . . . . 476 25.7.2 Applying the Root-Swarm Metaphor for Optimisation. . 476 25.8 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 477

XX Contents

**26 Chemically Inspired Algorithms** . . . . . . . . . . . . . . . . . . . . . . . . . . . 479 26.1 A Brief Chemistry Primer . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 479 26.2 Chemically Inspired Algorithms . . . . . . . . . . . . . . . . . . . . . . . . . . . 481 26.2.1 Chemical Reaction Optimisation (CRO) . . . . . . . . . . . . . . 481 26.2.2 Artificial Chemical Reaction Optimisation Algorithm

(ACROA) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 482 26.3 The CRO Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 483 26.3.1 Potential and Kinetic Energy and the Buffer . . . . . . . . . . 483 26.3.2 Types of Collision and Reaction . . . . . . . . . . . . . . . . . . . . . 484 26.3.3 The High-Level CRO Algorithm . . . . . . . . . . . . . . . . . . . . . 485 26.3.4 On-wall Ineffective Collision . . . . . . . . . . . . . . . . . . . . . . . . 489 26.3.5 Decomposition . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 489 26.3.6 Intermolecular Ineffective Collision . . . . . . . . . . . . . . . . . . 490 26.3.7 Synthesis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 492 26.4 Applications of CRO . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 493 26.5 Discussion of CRO . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 494 26.5.1 Potential Future Avenues for Research . . . . . . . . . . . . . . . 496 26.6 Summary . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 498

**Part VIII The Future of Natural Computing Algorithms**

**27 Looking Ahead** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 501 27.1 Open Issues . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 501 27.1.1 Hybrid Algorithms . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 501 27.1.2 The Power and the Dangers of Metaphor . . . . . . . . . . . . . 502 27.1.3 Benchmarks and Scalability . . . . . . . . . . . . . . . . . . . . . . . . . 502 27.1.4 Usability and Parameter-Free Algorithms . . . . . . . . . . . . . 503 27.1.5 Simulation and Knowledge Discovery . . . . . . . . . . . . . . . . 503 27.2 Concluding Remarks . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 503

**References** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 505

**Index** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 547

**1Introduction**

**Fig. 1.1.** The three facets of natural computing: 1) natural systems as computa- tional media (e.g., DNA and molecular computing), 2) simulation of natural systems with the potential for knowledge discovery, and 3) algorithms inspired by the natural world

Although there is no unique definition of the term natural computing, most commonly the field is considered to consist of three main strands of enquiry: see Fig. 1.1. The first strand concerns the use of natural materials and phe- nomena for computational purposes such as DNA and molecular computing (computing in vivo); the second strand concerns the application of computer simulations to replicate natural phenomena in order to better understand those phenomena (e.g., artificial life, agent based modelling and computa- tional biology); and the third strand, which forms the subject matter of this book, is concerned with the development of computational algorithms which draw their metaphorical inspiration from systems and phenomena that occur in the natural world. These algorithms can be applied to a multiplicity of

© Springer-Verlag Berlin Heidelberg 2015 1 A. Brabazon et al., *Natural Computing Algorithms*, Natural Computing Series, DOI 10.1007/978-3-662-43631-8\_1

2 1 Introduction

real-world problems including optimisation, classification, prediction, cluster- ing, design and model induction. The objective of this book is to provide an introduction to a broad range of natural computing algorithms.

**1.1 Natural Computing Algorithms: An Overview**

Natural computing is inherently multidisciplinary as it draws inspiration from a diverse range of fields of study including mathematics, statistics, computer science and the natural sciences of biology, physics and chemistry. As our un- derstanding of natural phenomena has deepened so too has our recognition that many mechanisms in the natural world parallel computational processes and can therefore serve as an inspiration for the design of problem-solving algorithms (defined simply as a set of instructions for solving a problem of interest). In this book we introduce and discuss a range of families of natu- ral computing algorithms. Figure 1.2 provides a high-level taxonomy of the primary methodologies discussed in this book which are grouped under the broad umbrellas of evolutionary computing, social computing, neurocomput- ing, immunocomputing, developmental and grammatical computing, physical computing, and chemical computing.

**1.1.1 Biologically Inspired Algorithms**

An interesting aspect of biological systems at multiple levels of scale (ecosys- tems, humans, bacteria, cells, etc.) which suggests that they may provide good sources of inspiration for solving real-world problems is that the very process of survival is itself a challenging problem! In order to survive successfully or- ganisms need to be able to find resources such as food, water, shelter and mates, whilst simultaneously avoiding predators. It is plausible that mecha- nisms which have evolved in order to assist survivability of organisms in these settings, such as sensing, communication, cognition and mobility could prove particularly useful in inspiring the design of computational algorithms. Virtu- ally all biological systems exist in high-dimensional (i.e., many factors impact on their survival), dynamic environments and hence biologically inspired al- gorithms may be of particular use in problem solving in these conditions.

Although many aspects of biological systems are noteworthy, a few charac- teristics of biological systems which provide food for thought in the design of biologically inspired algorithms include: the importance of the population; the emphasis on robustness (survival) rather than on optimality; and the existence of multilevel adaptive processes.

**Populational Perspective**

In many biologically inspired algorithms the search for good solutions takes place within a population of potential solutions. As in biological settings, indi-

1.1 Natural Computing Algorithms: An Overview 3

**Fig. 1.2.** A taxonomy of the nature inspired algorithms discussed in this book

4 1 Introduction

viduals in a population can be considered an individual hypothesis (or learn- ing trial) in the game of survival. From a species point of view, maintaining a dispersed population of individuals reduces the chance that environmental change will render the entire species extinct.

In contrast, in many traditional (non-natural-computing) optimisation al- gorithms, the paradigm is to generate a single trial solution and then itera- tively improve it. Consider for example, a simple random hill-climbing opti- misation algorithm (Algorithm 1.1) where an individual solution is iteratively improved using a greedy search strategy. In greedy search, any change in a solution which makes it better is accepted. This implies that a hill-climbing algorithm can find a local, but not necessarily the global, optimum (Fig. 1.3). This makes the choice of starting point a critical one. Note that this is so for all hill-climbing algorithms, including those that exploit information found so far (e.g., Newton’s algorithm uses slope information, while the Nelder–Mead algorithm does not need slopes but uses several previous points’ objective function values).

Many of the biologically inspired algorithms described in this book main- tain and successively update a population of potential solutions, which in the ideal case provides a good coverage (or sampling) of the environment in which we are problem-solving, resulting in a form of parallel search. This of course assumes the process that generates the first population disperses the individ- uals in an appropriate manner so as to maximise coverage of the environment. Ideally as search progresses it might be desirable to maintain a degree of dis- persion to avoid premature convergence of the population to local optima. It is the existence of a population which allows these bioinspired algorithms the potential to achieve global search characteristics, and avoid local optima through the populational dispersion of individuals.

**Dispersion and Diversity**

It is important to highlight this point that we should not confuse the no- tions of diversity (of objective function values) and dispersion. Often we

**Algorithm 1.1:** Hill Climbing Algorithm

Randomly generate a solution x; Calculate the objective function value f(x) for the solution;

**repeat**

Replace the current solution with the new one **end until** terminating condition;

Randomly mutate solution; **if** new solution is better than the current solution **then**

1.1 Natural Computing Algorithms: An Overview 5

*Global*

*Local maximum* f(x) *maximum Starting point*

xi

**Fig. 1.3.** A hill-climbing algorithm will find a local optimum. Due to its greedy search strategy it cannot then escape from this as it would require a ‘downhill’ move to traverse to the global optimum

(over)emphasise the value of diversity within a population. From a compu- tational search perspective it is arguably more valuable to focus on the dis- persion (or coverage) of the population. It is possible to have an abundance of diversity within a population; yet, at the same time, the population could be converged on a local optimum. However, a population which has a high value of dispersion is less likely to be converged in this manner. Figure 1.4 illustrates the difference between dispersion and diversity. The importance of dispersion is brought into sharp focus if we expose the population to a chang- ing environment where the location of the optima might change/move over time. A population which maintains dispersion may have a better chance to adapt to an environment where the global optimum moves a relatively far distance from its current location.

**Communication**

Another critical aspect of most of the algorithms described in this book is that the members of the population do not search in isolation. Instead they can communicate information on the quality of their current (or their previous) solution to other members of the population. Communication is interpreted broadly here to mean exchange of information between members of the pop- ulation, and so might take various forms: from chemical signals being left in the environment of a social computing algorithm, which can be sensed by individuals in the population; to the exchange of genes in an evolutionary algorithm. This information is then used to bias the search process towards areas of better solutions as the algorithm iterates.

6 1 Introduction

*Global*

f(x)

*Local optimum optimum*xi

*A population with (a) High Diversity, and (b) High Dispersion*

*Global*

f(x)

*Local optimum optimum*xi

*A population with (a) High Diversity, and (b) Low Dispersion*

**Fig. 1.4.** An illustration of the difference between a dispersed (top) and a diverse (bottom) population. It is possible to have a large amount of diversity (e.g., a wide range of objective function values) but still be converged on a local optimum

**Robustness**

‘Survival in a dynamic environment’ is the primary aim in many biological systems. Therefore the implicit driver for organisms is typically to uncover and implement survival strategies which are ‘good enough’ for current conditions and which are robust to changing environmental conditions. Optimality is a fleeting concept in dynamic environments as (for example), the location of good food resources last week may not hold true next week.

**Adaptiveness**

Adaptiveness (new learning) occurs at multiple levels and timescales in bio- logical systems, ranging from (relatively) slow genetic learning to (relatively) fast lifetime learning. The appropriate balance between speed of adaptation and the importance of memory (or old learning) depends on the nature of

1.1 Natural Computing Algorithms: An Overview 7

the dynamic environment faced by the organism. The more dynamic the en- vironment, the greater the need for adaptive capability and the less useful is memory.

An interesting model of adaptation in biological systems is outlined in Sipper et al. [581] and is commonly referred to as the POE model. This model distinguishes between three levels of organisation in biological systems:

i. phylogeny (P), ii. ontogeny (O), and iii. epigenesis (E).

Phylogeny concerns the adaptation of genetic code over time. As the genome adapts and differentiates, multiple species, or phylogeny, evolve. The primary mechanisms for generating diversity in genetic codes are mutation, and, in the case of sexual reproduction, recombination. The continual gener- ation of diversity in genetic codings facilitates the survival of species, and the emergence of new species, in the face of changing environmental conditions. Much of the research around evolutionary computation to date exists along this axis of adaptation.

Ontogeny refers to the development of a multicellular organism from a zy- gote. While each cell maintains a copy of the original genome, it specialises to perform specific tasks depending on its surroundings (cellular differentiation). In recent years there has been increasing interest in developmental approaches to adaptation with the adoption of models such as artificial genetic regulatory networks.

Epigenesis is the development of systems which permit the organism to integrate and process large amounts of information from its environment. The development and working of these systems is not completely specified in the genetic code of the organism and hence are referred to as ‘beyond the genetic’ or epigenetic. Examples include the immune, the nervous and the endocrine systems. While the basic structure of these systems is governed by the or- ganism’s genetic code, they are modified throughout the organism’s lifetime as a result of its interaction with the environment. For example, a human’s immune system can maintain a memory of pathogens that it has been exposed to (the acquired immune system). The regulatory mechanism which controls the expression of genes is also subject to epigenetic interference. For exam- ple, chemical modification (e.g., through methylation) of regulatory regions of the genome can have the effect of silencing (turning off or dampening) the expression of a gene(s). The chemical modification can arise due to the en- vironmental state in which the organism lives. So the environment can effect which genes are expressed (or not) thereby indirectly modifying an organism’s genetic makeup to suit the conditions in which it finds itself.

In complex biological organisms, all three levels of organisation are inter- linked. However, in assisting us in thinking about the design of biologically inspired algorithms, it can be useful to consider each level of organisation (and their associated adaptive processes) separately (Fig. 1.5). Summarising

8 1 Introduction

**Fig. 1.5.** Three levels of organisation in biological systems

1.1 Natural Computing Algorithms: An Overview 9

the three levels, Sipper et al. [581] contends that they embed the ideas of evolution, structural development of an individual, and learning through en- vironmental interactions.

Most biologically inspired algorithms draw inspiration from a single level of organisation but it is of course possible to design hybrid algorithms which draw inspiration from more than one level. For example, neuroevolution, dis- cussed in Chap. 15, combines concepts of both evolutionary and lifetime learn- ing, and evolutionary–development (evo–devo) approaches (e.g., see Chap. 21) hybridise phylogenetic and ontogenetic adaptation.

**1.1.2 Families of Naturally Inspired Algorithms**

A brief overview of some of the main families of natural computing algorithms is provided in the following paragraphs. A more detailed discussion of each of these is provided in later chapters.

**Evolutionary Computing**

Evolutionary computation simulates an evolutionary process on a computer in order to breed good solutions to a problem. The process draws high-level inspiration from biological evolution. Initially a population of potential solu- tions are generated (perhaps randomly), and these are iteratively improved over many simulated generations. In successive iterations of the algorithm, fitness based selection takes place within the population of solutions. Better solutions are preferentially selected for survival into the next generation of solutions, with diversity being introduced in the selected solutions in an at- tempt to uncover even better solutions over multiple generations. Algorithms that employ an evolutionary approach include genetic algorithms (GAs), evo- lutionary strategies (ES), evolutionary programming (EP) and genetic pro- gramming (GP). Differential evolution (DE) also draws (loose) inspiration from evolutionary processes.

**Social Computing**

The social models considered in this book are drawn from a swarm metaphor. Two popular variants of swarm models exist, those inspired by the flocking behaviour of birds and fish, and those inspired by the behaviour of social insects such as ants and honey bees. The swarm metaphor has been used to design algorithms which can solve difficult problems by creating a population of problem solvers, and allowing these to communicate their relative success in solving the problem to each other.

10 1 Introduction

**Neurocomputing**

Artificial neural networks (NNs) comprise a modelling methodology whose inspiration arises from a simplified model of the workings of the human brain. NNs can be used to construct models for the purposes of prediction, classifi- cation and clustering.

**Immunocomputing**

The capabilities of the natural immune system are to recognise, destroy and remember an almost unlimited number of foreign bodies, and also to protect the organism from misbehaving cells in the body. Artificial immune systems (AIS) draw inspiration from the workings of the natural immune system to develop algorithms for optimisation and classification.

**Developmental and Grammatical Computing**

A significant recent addition to natural computing methodologies are those inspired by developmental biology (developmental computing) and the use of formal grammars (grammatical computing) from linguistics and computer science. In natural computing algorithms grammars tend to be used in a gen- erative sense to construct sentences in the language specified by the grammar. This generative nature is compatible with a developmental approach, and con- sequently a significant number of developmental algorithms adopt some form of grammatical encoding. As will be seen there is also an overlap between these algorithms and evolutionary computation. In particular, a number of approaches to genetic programming adopt grammars to control the evolving executable structures. This serves to highlight the overlapping nature of nat- ural systems, and that our decomposition of natural computing algorithms into families of inspiration is one of convenience.

**1.1.3 Physically Inspired Algorithms**

Just as biological processes can inspire the design of computational algorithms, inspiration can also be drawn from looking at physical systems and processes. We look at three algorithms which are inspired by the properties of interacting physical bodies such as atoms and molecules, namely simulated annealing, quantum annealing, and the constrained molecular dynamics algorithm. One interesting strand of research in this area is drawn from a quantum metaphor.

**Quantum Inspired Algorithms**

Quantum mechanics seeks to explain the behaviours of natural systems that are observed at very short time or distance scales. An example of a system is a

1.1 Natural Computing Algorithms: An Overview 11

subatomic particle such as a free electron. Two important concepts underlying quantum systems are the superposition of states and quantum entanglement. Recent years have seen the development of a series of quantum inspired hybrid algorithms including quantum inspired evolutionary algorithms, social com- puting, neurocomputing and immunocomputing. A claimed benefit of these algorithms is that because they use a quantum inspired representation, they can potentially maintain a good balance between exploration and exploitation. It is also suggested that they could offer computational efficiencies.

**1.1.4 Plant Inspired Algorithms**

Plants represent some 99% of the eukaryotic biomass of the planet and have been highly successful in colonising many habitats with differing resource po- tential. Just like animals or simpler organisms such as bacteria (Chap. 11), plants have evolved multiple problem-solving mechanisms including complex food foraging mechanisms, environmental-sensing mechanisms, and reproduc- tive strategies. Although plants do not have a brain or central nervous system, they are capable of sensing environmental conditions and taking actions which are ‘adaptive’ in the sense of allowing them to adjust to changing environmen- tal conditions. These features of plants offer potential to inspire the design of computational algorithms and a recent stream of work has seen the de- velopment of a family of plant algorithms. We introduce a number of these algorithms and highlight some current areas of research in this subfield.

**1.1.5 Chemically Inspired Algorithms**

Chemical processes play a significant role in many of the phenomena described in this book, including (for example) evolutionary processes and the workings of the natural immune system. However, so far, chemical aspects of these pro- cesses have been largely ignored in the design of computational algorithms. An emerging stream of study is beginning to remedy this gap in the litera- ture and we describe an optimisation algorithm inspired by the processes of chemical reactions.

**1.1.6 A Unified Family of Algorithms**

Although it is useful to compartmentalise the field of Natural Computing into different subfields, such as Evolutionary and Social Computing, for the purposes of introducing the material, it is important to emphasise that this does not actually reflect the reality of the natural world around us. In na- ture all of these learning mechanisms coexist and interact forming part of a larger natural, complex and adaptive system encompassing physical, chemical, evolutionary, immunological, neural, developmental, grammatical and social processes, which, for example, are embodied in mammals. In much the same

12 1 Introduction

way that De Jong advocated for a unified field of Evolutionary Computa- tion [144], we would favour a unification of all the algorithms inspired by the natural world into the paradigm of Natural Computing and Natural Comput- ing Algorithms. Increasingly we are seeing significant overlaps between the different families of algorithms, and upon real-world application it is common to witness their hybridisation (e.g., neuroevolution, evo–devo etc.). We antic- ipate that the future of natural computing will see the integration of many of these seemingly different approaches into unified software systems working together in harmony.

**1.1.7 How Much Natural Inspiration?**

An obvious question when considering computational algorithms which are inspired by natural phenomena is how accurate does the metaphor need to be? We consider that the true measure of usefulness of a natural computing algorithm is not its degree of veracity with (what we know of) nature, but rather its effectiveness in problem solving; and that an intelligent designer of algorithms should incorporate ideas from nature — while omitting others — so long as these enhance an algorithm’s problem-solving ability. For example, considering quantum inspired algorithms, unless we use quantum computers, which to date are experimental devices and not readily available to the general reader, it is not possible to efficiently simulate effects such as entanglement; hence such algorithms while drawing a degree of inspiration from quantum mechanics must of necessity omit important, even vital, features of the natural phenomenon from which we derive inspiration.

**1.2 Structure of the Book**

The field of natural computing has expanded greatly in recent years beyond its evolutionary and neurocomputing roots to encompass social, immune system, physical and chemical metaphors. Not all of the algorithms discussed in this book are fully explored as yet in terms of their efficiency and effectiveness. However, we have deliberately chosen to include a wide range of algorithms in order to illustrate the diversity of current research into natural computing algorithms.

The remainder of this book is divided into eight parts. Part I starts by pro- viding an overview of evolutionary computation (Chap. 2), and then proceeds to describe the genetic algorithm (Chaps. 3 and 4), evolutionary strategies and evolutionary programming (Chap. 5), differential evolution (Chap. 6), and genetic programming (Chap. 7). Part II focusses on social computing and provides coverage of particle swarm optimisation (Chap. 8), insect al- gorithms (Chaps. 9 and 10), bacterial foraging algorithms (Chap. 11), and other social algorithms (Chap. 12). Part III of the book provides coverage of the main neurocomputing paradigms including supervised learning neural

1.2 Structure of the Book 13

network models such as the multilayer perceptron, recurrent networks, radial basis function networks and support vector machines (Chap. 13), unsuper- vised learning models such as self-organising maps (Chap. 14), and hybrid neuroevolutionary models (Chap. 15). Part IV discusses immunocomputing (Chap. 16). Part V of the book introduces developmental and grammati- cal computing in Chap. 17 and provides detailed coverage of grammar-based approaches to genetic programming in Chap. 18. Two subsequent chapters expose in more detail some of grammar-based genetic programming’s more popular forms, grammatical evolution and TAG3P (Chaps. 19 and 20), fol- lowed by artificial genetic regulatory network algorithms in Chap. 21. Part VI introduces physically inspired computing (Chaps. 22 to 24). Part VII intro- duces some other paradigms that do not fit neatly into the earlier categories, namely, plant-inspired algorithms in Chap. 25, and chemically inspired com- puting in Chap. 26. Finally, Part VIII (Chap. 27) outlines likely avenues of future work in natural computing algorithms.

We hope the reader will enjoy this tour of natural computing algorithms as much as we have enjoyed the discovery (and in some cases rediscovery) of these inspiring algorithms during the writing of this book.

**Evolutionary Computing**

**Part I**

**2Introduction to Evolutionary Computing**

‘Owing to this struggle for life, variations, however slight and from whatever cause proceeding, if they be in any degree profitable to the individuals of a species, in their infinitely complex relations to other organic beings and to their physical conditions of life, will tend to the preservation of such individuals, and will generally be inherited by the offspring. The offspring, also, will thus have a better chance of surviving, for, of the many individuals of any species which are periodically born, but a small number can survive. I have called this principle, by which each slight variation, if useful, is preserved, by the term Natural Selection’. (Darwin, 1859 [127], p. 115)

Biological evolution performs as a powerful problem-solver that attempts to produce solutions that are at least good enough to perform the job of survival in the current environmental context. Since Charles Darwin popularised the theory of Natural Selection, the driving force behind evolution, molecular biol- ogy has unravelled some of the mysteries of the components that underpinned earlier evolutionary ideas. In the twentieth century molecular biologists un- covered the existence of DNA, its importance in determining hereditary traits and later its structure, unlocking the key to the genetic code. The accumu- lation of knowledge about the biological process of evolution, often referred to as neo-Darwinism, has in turn given inspiration to the design of a family of computational algorithms known collectively as evolutionary computation. These evolutionary algorithms take their cues from the biological concepts of natural selection and the fact that the heritable traits are physically encoded on DNA, and can undergo variation through a series of genetic operators such as mutation and crossover.

© Springer-Verlag Berlin Heidelberg 2015 A. Brabazon et al., *Natural Computing Algorithms*, Natural Computing Series, DOI 10.1007/978-3-662-43631-8\_

17

2

18 2 Introduction to Evolutionary Computing

**2.1 Evolutionary Algorithms**

Evolutionary processes represent an archetype, whose application transcends their biological root. Evolutionary processes can be distinguished by means of their four key characteristics, which are [57, 92]:

i. a population of entities, ii. mechanisms for selection, iii. retention of fit forms, and iv. the generation of variety.

In biological evolution, species are positively or negatively selected depending on their relative success in surviving and reproducing in the environment. Differential survival, and variety generation during reproduction, provide the engine for evolution [127, 589] (Fig. 2.1).

***Population***

**Selection**

**Replacement**

***Parents***

**Initialisation**

**Variety Generation *Offspring***

**Fig. 2.1.** Evolutionary cycle

These concepts have metaphorically inspired the field of evolutionary com- putation (EC). Algorithm 2.1 outlines the evolutionary meta-algorithm. There are many ways of operationalising each of the steps in this meta-algorithm; consequently, there are many different, but related, evolutionary algorithms. Just as in biological evolution, the selection step is a pivotal driver of the algo- rithm’s workings. The selection step is biased in order to preferentially select better (or ‘more fit’) members of the current population. The generation of new individuals creates offspring or children which bear some similarity to their parents but are not identical to them. Hence, each individual represents a trial solution in the environment, with better individuals having increased chance of influencing the composition of individuals in future generations. This process can be considered as a ‘search’ process, where the objective is to continually improve the quality of individuals in the population.

**Evolutionary Computation in Computer Science**

The idea of a (computer) simulated evolutionary process dates back to the very dawn of digital computing, being introduced in the writings of Alan Turing in 1948–1950 [636, 637]. One of the earliest published works on the imple- mentation of an evolutionary-like algorithm was by Friedberg in 1958 [203] and followed by [171, 204]. In the earlier of these studies, random and rou- tine changes were made to binary strings representing machine code, with the performance of individual instructions being monitored in a credit assign- ment form of learning. Friedberg’s studies were subsequently compiled into an edited collection providing a snapshot of the seminal papers that gave rise to the field of Evolutionary Computation [192]. Friedberg’s work represents the origin of what is now known as Genetic Programming, which was later popu- larised via the work of Cramer [119], Dickmans et al. [157] and Koza [339] in the 1980s.

During the 1960s and 1970s two significant, independent, lines of re- search developing evolutionary algorithms were undertaken in Europe and the US. The Europeans (Rechenberg and Schwefel) developed Evolution Strate- gies [530, 531, 560, 561] and the Americans (Fogel et al. and Holland) devel- oped Evolutionary Programming [194] and Genetic Algorithms [281]. More recently, Storn and Price have added Differential Evolution [600] to the fam- ily of evolutionary algorithms.

Evolutionary Algorithms

Genetic Algorithm

Evolutionary Strategies

Differential Evolution

Differential Evolution

Genetic Programming

Genetic Programming

Genetic Programming

Evolutionary Programming

Evolutionary Programming

Evolutionary Programming

Evolutionary Programming

**Fig. 2.2.** Main branches of evolutionary computation

**Algorithm 2.1:** Evolutionary Algorithm

Initialise the population of candidate solutions; **repeat**

Select individuals (parents) for breeding from the current population; Generate new individuals (offspring) from these parents; Replace some or all of the current population with the newly generated individuals; **until** terminating condition;

2.1 Evolutionary Algorithms 19

20 2 Introduction to Evolutionary Computing

Although the above strands of research were initially distinct, with each having their own proponents, the lines between all of these evolutionary in- spired approaches is becoming blurred with representations and strategies be- ing used interchangeably between the various algorithms. As such, today it is common to use the term evolutionary algorithm to encompass all of the above approaches [144, 175]. Figure 2.2 provides a taxonomy of the more common branches of evolutionary computation.

In the following chapters in Part I of the book, we will introduce three of the main families of evolutionary inspired algorithms in turn, namely, the genetic algorithm (Chaps. 3 and 4), evolutionary strategies (Chap. 5), and differential evolution (Chap. 6). Following these, genetic programming is then discussed in Chap. 7.

**3Genetic Algorithm**

While the development of the genetic algorithm (GA) dates from the 1960s, this family of algorithms was popularised by Holland in the 1970s [281]. The GA has been applied in two primary areas of research: optimisation, in which GAs represent a population-based optimisation algorithm, and the study of adaptation in complex systems, wherein the evolution of a population of adapting entities is simulated over time by means of a pseudonatural selec- tion process using differential-fitness selection, and pseudogenetic operators to induce variation in the population.

In this chapter we introduce the canonical GA, focussing on its role as an optimising methodology, and discuss the design choices facing a modeller who is seeking to implement a GA.

**3.1 Canonical Genetic Algorithm**

In genetics, a strong distinction is drawn between the genotype and the phe- notype; the former contains genetic information, whereas the latter is the physical manifestation of this information. Both play a role in evolution as the biological processes of diversity generation act on the genotype, while the ‘worth’ or fitness of this genotype in the environment depends on the survival and reproductive success of its corresponding phenotype. Similarly, in the canonical GA a distinction is made between the encoding of a solution (the ‘genotype’), to which simulated genetic operators are applied, and the pheno- type associated with that encoding. These phenotypes can have many diverse forms depending on the application of interest. Unlike traditional optimisation techniques the GA maintains and iteratively improves a population of solu- tion encodings. Evolutionary algorithms, including the GA, can be broadly characterised as [193]:

x[t + 1] = r(v(s(x[t]))) (3.1)

© Springer-Verlag Berlin Heidelberg 2015 A. Brabazon et al., *Natural Computing Algorithms*, Natural Computing Series, DOI 10.1007/978-3-662-43631-8\_ 3

21

22 3 Genetic Algorithm

where x[t] is the population of encodings at timestep t, v(.) is the random variation operator (crossover and mutation), s(.) is the selection for mating operator, and r(.) is the replacement selection operator. Once the initial pop- ulation of encoded solutions has been obtained and evaluated, a reproductive process is applied in which the encodings corresponding to the better-quality, or fitter, solutions have a higher chance of being selected for propagation of their genes into the next generation. Over a series of generations, the better adapted solutions in terms of the given fitness function tend to flourish, and the poorer solutions tend to disappear. Just as biological genotypes encode the results of past evolutionary trials, the population of genotypes in the GA also encode a history (or memory) of the relative success of the resulting phenotypes for the problem of interest.

Therefore, the canonical GA can be described as an algorithm that turns one population of candidate encodings and their corresponding solutions into another using a number of stochastic operators. Selection exploits information in the current population, concentrating interest on high-fitness solutions. The selection process is biased in favour of the encodings corresponding to bet- ter/fitter solutions and better solutions may be selected multiple times. This corresponds to the idea of survival of the fittest. Crossover and mutation per- turb these solutions in an attempt to uncover even better solutions. Mutation does this by introducing new gene values into the population, while crossover allows the recombination of fragments of existing solutions to create new ones. Algorithm 3.1 lists the key steps in the canonical genetic algorithm.

An important aspect of the algorithm is that the evolutionary process operates on the encodings of solutions, rather than directly on the solutions themselves. In determining the fitness of these encodings, they must first be translated into a solution to the problem of interest, the fitness of the solution determined, and finally this fitness is associated with the encoding (Fig. 3.1).

Encoded string (genotype)

Decoding step

Solution (phenotype)

Fitness value

**Fig. 3.1.** Decoding of genotype into a solution in order to calculate fitness

**3.1.1 A Simple GA Example**

To provide additional insight into the workings of the canonical GA, a simple numerical example is now provided. Assume that candidate solutions are en- coded as a binary string of length 8 and the fitness function f(x) is defined as the number of 1s in the bit string (this is known as the OneMax problem). Let n = 4 with pcross = 0.6 and pmut = 0.05. Assume also that the initial population is generated randomly as in Table 3.1.

**Table 3.1.** A sample initial random population

**Candidate String Fitness**

A 10000110 3 B 01101100 4 C 10100000 2 D 01000110 3

Next, a selection process is applied based on the fitness of the candidate solutions. Suppose the first selection draws candidates B and D and the second

**Algorithm 3.1:** Canonical Genetic Algorithm

Determine how the solution is to be encoded as a genotype and define the fitness function; Create an initial population of genotypes; Decode each genotype into a solution and calculate the fitness of each of the n solution candidates in the population;

**repeat**

Select two parents randomly from the mating pool; With probability pcross, perform a crossover process on the encodings of the selected parent solutions, to produce two new (child) solutions; Otherwise, crossover is not performed and the two children are simply copies of their parents; With probability pmut, apply a mutation process to each element of the encodings of the two child solutions; **until** n new child solutions have been created; Replace the old population with the newly created one (this constitutes a generation); **until** terminating condition;

Select n members from the current population of encodings (the parents) in order to create a mating pool; **repeat**

3.1 Canonical Genetic Algorithm 23

24 3 Genetic Algorithm

draws B and A. For each set of parents, the probability that a crossover (or recombination) operator is applied is pcross. Assume that B and D are crossed over between bit position 1 and 2 to produce child candidates E and F (Table 3.2), and that crossover is not applied to B and A.

**Table 3.2.** Crossover applied to individuals B and D from Table 3.1, after the first element of each binary string, to produce the offspring E and F

Crossover is not applied to B and A; hence the child candidates (G and H) are clones of the two parent candidates (Table 3.3).

**Table 3.3.** No crossover is applied to B and D; hence the child candidates G and H are clones of their parents

Finally, the mutation operator is applied to each child candidate with probability pmut. Suppose candidate E is mutated (to a 1) at the third locus, that candidate F is mutated (to a 1) at the seventh locus, and that no other mutations take place. The resulting new population is presented in Table 3.4. By biasing selection for reproduction towards more fit parents, the GA has increased the average fitness of the population in this example from 3 (= the 3+4+2+3

fitness of the best solution 4 ) after the first generation and we can see that F in the second generation is better than that of any solution in the first generation.

**3.2 Design Choices in Implementing a GA**

Although the basic idea of the GA is quite simple, a modeller faces a number of key decisions when looking to apply it to a specific problem:

4 ) to 4 (= 4+5+4+3

Resulting Child **Candidate G Candidate H**

01101100 10000110

Resulting Child **Candidate E Candidate F**

0 1000110 0 1101100

Initial Parent **Candidate B Candidate A**

01101100 10000110

Initial Parent **Candidate B Candidate D**

0 1101100 0 1000110

3.3 Choosing a Representation 25

**Table 3.4.** Population of solution encodings after the mutation operator has been applied

**Candidate String Fitness**

E 01100110 4 F 01101110 5 G 01101100 4 H 10000110 3

• what representation should be used?

• how should the initial population of genotypes be initialised?

• how should fitness be measured?

• how should diversity be generated in the population of genotypes?

Each of these are discussed in the following sections.

**3.3 Choosing a Representation**

In thinking about evolutionary processes, two distinct mapping processes can be distinguished, one between the genotype and the phenotype, and a second between the phenotype and a fitness measure (Fig. 3.2). In applying the GA, the user must select how the problem is to be represented, and there are two aspects to this decision. First, the user must decide how potential solutions (phenotypes) will be encoded onto the genotype. Secondly, the user must decide how individual elements of the genotype will be encoded.

Genotypic space

Phenotypic space

Fitness metric

Fitness metric

**Fig. 3.2.** Mapping from genotypic to phenotypic space with each phenotype in turn being mapped to a fitness measure

26 3 Genetic Algorithm

**3.3.1 Genotype to Phenotype Mapping**

Suppose a modeller is trying to uncover the relationship between a dependent variable and a set of explanatory variables and that she has collected a dataset of sample values. Let us assume that the relationship between the variables is known to be linear of the form y = β0 + β1x1 + β2x2. The object is now to uncover the best values for the model coefficients so that the model fits the dataset as well as possible.

In order to apply the GA for this task, a decision is required as to how these parameters should be represented on the genotype. In this case the genotype can simply consist of three elements, each of which encodes a real number (the values for β0,β1, and β2) and the GA’s task is then to uncover the optimal values for these three parameters. In this case, the mapping from the genotype to the phenotype (Fig. 3.3) is straightforward, with three values being plugged directly into the linear model.

1 2 3.12 23.11 3.93 *y x x* = − + + -3.12 23.11 3.93

**Fig. 3.3.** Mapping from real-valued genotype to produce a linear model

The mapping from genotype to phenotype can of course be more complex. Suppose the task was to evolve a classifier of the form:

IF [xi (<, >) VALUE1]

(AND, OR) [xj (<, >) VALUE2] THEN (Class 1) ELSE(Class 2)

Here the phenotype takes the form of a compound conditional statement and the task of GA in uncovering the best classifier is to decide which two explana- tory variables to include from the dataset, whether each of these needs to be greater than or less than a trigger value, and how the two logic statements should be compounded. The number of (integer) choices for xi and xj depend on the number of explanatory variables in the dataset. Hence, the genotype will need to allow the generation of a series of integer and real values (Fig. 3.4).

**3.3.2 Genotype Encodings**

In early research on GAs, the canonical GA used binary-valued encodings for genotypes (0101 ...1). Although this seems very limiting at first glance, binary-valued encodings can be easily used to produce real (or integer) valued

3.3 Choosing a Representation 27

xi < > Value1 AND /

OR

xi < > Value1

Integer Integer Real Integer Integer Integer Real

**Fig. 3.4.** Mixed integer/real genotype, consisting of seven elements, which could encode a classification rule

outputs, thereby allowing the application of a binary-valued GA to a wide range of problems.

The simplest decoding method is to convert the binary string to an integer value, which can in turn be converted into a real value if required. A binary genotype of length n can encode any integer from 0 to 2n − 1 (Table 3.5). More generally, it is possible to encode any integer in the range 0,...,L − 1, even when there is no n such that L = 2n.

If a real-valued output is required, the integer value obtained by decoding the binary string can be divided by 2n − 1 to obtain a real number in the interval [0,1]. A real number in any interval1 [a, b] can be obtained by taking the result of the last calculation and rescaling it using the formula a+x(b−a). Taking an example, a binary string which is eight bits long can encode any integer between 0 and 255. If we consider the binary string (00000111), reading from right to left, this can be decoded into the integer value 7 (calculated as: 20 × 1+21 × 1+22 × 1+23 × 0+24 × 0+25 × 0+26 × 0+27 × 0). If instead of an integer value in the range 0 to 255, a real value in the range [0,5] were required, the integer value could be converted into a real value as follows: Although 0 + 255 7

the × above (5 − 0) decoding = 0.027451.

scheme for a binary string is easy to un- derstand, it can suffer from Hamming cliffs, as sometimes a large change in the genotype is required to produce a small change in the resulting integer value. Looking at the change in the binary value required to move from an integer value of 3 to 4 in Table 3.5, it can be seen that the underlying geno- type needs to change in all three bit positions. Hamming cliffs can potentially create barriers that the GA could find difficulty in passing.

An alternative encoding system that has been used in some GA systems is that of Gray coding. In Gray coding a single integer change only requires a one bit change in the binary genotype. This means that adjacent solutions in the integer search space will be adjacent in the (binary) encoding space as well, requiring fewer mutations to discover. The Gray coding rule starts with a string of all 0s for the integer value 0. To create each subsequent integer in sequence the rule successively flips the right-most bit that produces a new string.

1Recall that the shorthand [a, b] denotes the interval of all real numbers x ∈ R such that a ≤ x ≤ b, while [a, b) denotes the interval of all x ∈ R such that a ≤ x<b.

28 3 Genetic Algorithm

**Table 3.5.** Integer conversion for standard and Gray coding

**Canonical Integer Value Binary Code Gray Code**

0 000 000 1 001 001 2 010 011 3 011 010 4 100 110 5 101 111 6 110 101 7 111 100

For many problems, a real-valued genotype encoding is the most natural representation and most current optimisation applications of the GA use real- valued encodings (for example, 2.13,...,−14.56).

**3.3.3 Representation Choice and the Generation of Diversity**

The choice of representation is crucial as it determines the nature of the search space traversed by the GA. The choice also impacts on the appropriate design of diversity generation processes such as crossover and mutation.

Ideally, small (big) changes in the genotype should result in small (larger) changes in the phenotype and its associated fitness. This feature is known as locality. For example, if there is a good pairing of representation and di- versity operators, minor mutations on the genotype will produce relatively small changes in the phenotype and its fitness, whereas a crossover between two very different parents will lead to a larger change in the phenotype. This means that the operators of mutation and crossover will perform distinctly different search processes.

Going back to the previous example of Gray coding, it can be observed from the integer to Gray mapping in Table 3.5, that a small change in the phenotype corresponds to a small change in the genotype. However, the reverse is not true. A single bit-flip in the genotype can lead to a large change in the phenotype (integer value). Hence, even a Gray encoding has poor locality properties and will not necessarily produce better results than the canonical binary coding system.

Raidl and Gottlieb [525] emphasise three key characteristics for the design of quality evolutionary algorithms (EAs):

i. locality, ii. heritability, and iii. heuristic bias.

Locality refers to the case where small steps in the search space result in small steps in the phenotypic space. Strong locality increases the efficiency of

3.5 Measuring Fitness 29

evolutionary search by making it easier to explore the neighbourhood of good solutions, whereas weak locality means that evolutionary search will behave more like random search.

Heritability refers to the capability of crossover operators to produce chil- dren that utilise the information contained in their parents in a meaningful way. In general, good heritability will ensure that each property of a child should be inherited from one of its parents, and that traits shared by both parents should be inherited by their child. Crossover operators with weak heritability are more akin to macro-mutation operators.

Heuristic bias occurs when certain phenotypes are more likely to be cre- ated by the EA than others when sampling genotypes without any selection pressure. In an unbiased case, each item in the phenotypic space has the same chance of occurring if a genotype is randomly generated.

Obviously, inducing heuristic bias can be good if it tends to lead to better solutions (in contrast, random search will tend to have lower heuristic bias), but inducing bias tends to reduce genotypic diversity, which can hinder the search for a global optimum. Heuristic bias arises from the choice of represen- tation and the choice of variation operators.

**3.4 Initialising the Population**

If good starting points for the search process are known a priori, the efficiency of the GA can be improved by using this information to seed the initial pop- ulation. More commonly, good starting points are not known and the initial population is created randomly. For binary-valued genotypes, a random num- ber between 0 and 1 can be generated for each element of the genotype, with random numbers ≥ 0.5 resulting in the placing of a 1 in the corresponding locus of the genotype. If a real-valued representation is used, and boundary values for each locus of the genotype can be determined, each element of the genotype can be selected randomly from the bounded interval.

**3.5 Measuring Fitness**

The importance of the choice of fitness measure when designing a GA cannot be overstressed as this metric drives the evolutionary process. The first step in creating a suitable fitness measure is to identify an appropriate objective function for the problem of interest. This objective function often needs to be transformed into a suitable fitness measure via a transformation (for example, to ensure that the resulting fitness value is always nonnegative); hence:

F(x) = g(f(x)) (3.2)

30 3 Genetic Algorithm

where f is the objective function, g transforms the value of the objective function into a nonnegative number, and F is the fitness measure. A sim- ple example of a transformation is the linear rescaling of the raw objective function value:

g(x) = af(x) + b (3.3)

where a is chosen in order to ensure that the maximum fitness value is a scaled multiple of the average fitness and b is chosen in order to ensure that the resulting fitness values are nonnegative. Using rescaled fitnesses rather than raw objective function values can also help control the selection pressure in the algorithm (Sect. 3.6.1).

In addition to absolute measures of fitness as just described, it is also possible to define fitness in relative rather than absolute terms, thus avoiding having to calculate explicit fitness values for each population member. For example, if our aim is to evolve a chess player we could evaluate the population by allowing individuals to play tournaments against each other where the winner of the tournament is deemed the fittest.

**Estimating Fitness**

Evaluating the fitness of individual members of the population is usually the most computationally expensive and time-consuming step in a GA. In some cases it is not practical to obtain an exact fitness value for every individual in each iteration of the algorithm.

A simple initial step is to avoid retesting the same individuals, so before testing the fitness of a newly created individual, a check could be made to determine if the same genotype has been tested in a prior generation. If it has been, the known fitness value can simply be assigned to the current individual. More generally, in cases where fitness function evaluation is very expensive, we may wish to use less costly approximations of the fitness function in order to quickly locate good search regions.

One method of doing this is problem approximation, where we replace the original problem statement (fitness function) with a simpler one which approx- imates the problem of interest, the assumption being that a good solution to the simpler problem would be a good starting point in trying to solve the real problem of interest. An example of this would be the use of crash simulation systems where designs that perform well in computer simulations could then be subject to (expensive) real-world physical testing.

A second approach is to try to reduce the number of fitness function eval- uations by estimating an individual’s fitness based on the fitness of other ‘similar’ individuals. Examples of this include fitness inheritance, where the fitness of a child is inherited from its parent(s), or fitness imitation, where all the individual solutions in a cluster (those close together as defined by some distance metric) are given the same fitness (that of a representative solution of the cluster). Hence, an approximate fitness evaluation is used for much of

3.6 Generating Diversity 31

the EA run, with the population, or a subset of the better solutions in the population, being exposed to the real (expensive) fitness function periodically during the run. This process entails a trade-off, with the gains from the reduc- tion in the number of fitness evaluations being traded off against the risk that the search process will be biased through the use of fitness approximations.

A practical problem that can arise in applying GAs to real-world problems is that the fitness measures obtained can sometimes be noisy (for example, due to measurement errors). In this case, we may wish to resample fitness over a number of training runs, using an average fitness value in the selection and replacement process.

**3.6 Generating Diversity**

The process of generating new child solutions aims to exploit information from better solutions in the current population, while maintaining explorative ca- pability in order to uncover even better regions of the search space. Too much exploitation of already-discovered good solutions runs the risk of convergence of the population of genotypes to a local optimum, while too much exploration drives the search process towards random search.

A key issue in designing a good GA is the management of the explo- ration vs. exploitation balance. The algorithm must utilise, or exploit, already- discovered fit solution encodings, while not neglecting to continue to explore new regions of the search space which may contain even better solution en- codings. Choices for the selection strategy, the design of mutation and re- combination operators, and the replacement strategy, determine the balance between exploration and exploitation. Selection and crossover tend to pro- mote exploitation of already-discovered information, whereas mutation tends to promote exploration.

**3.6.1 Selection Strategy**

The design of the ‘selection for mating’ strategy determines the selection pres- sure (the degree of bias towards the selection of higher-fitness members of the population) of the algorithm. If the selection pressure is too low, information from good parents will only spread slowly through the population, leading to an inefficient search process. If the selection pressure is too high, the popula- tion is likely to get stuck in a local optimum, as a high selection pressure will tend to quickly reduce the degree of genotypic diversity in the population. Better-quality selection strategies therefore, encourage exploitation of high- fitness individuals in the population, without losing diversity in the population too quickly.

Although a wide variety of selection strategies have been designed for the GA, two common approaches are fitness proportionate selection and ordinal selection.

32 3 Genetic Algorithm

**Fitness Proportionate Selection**

The original method of selection for reproduction in the GA is fitness- proportionate selection (FPS) and under this method the probability that a specific member of the current population is selected for mating is directly related to its fitness relative to other members of the population. The selection process is therefore biased in favour of ‘good’ (i.e., fit) members of the current population. Given a list of each of the n individuals in the population and their associated fitnesses fi, a simple way to implement FPS is to generate a random number r ∈

[0,∑nj=1 fj), then select the individual i such that:

∑i−1j=1

fj ≤ r <

∑ifj. (3.4) j=1

As 20. 29.4. a Therefore, This numerical value ∑example, falls 4j=1 fin j suppose n = 4, f1 = f2 = 15, f3 = 10 and f4 = = 60. Assume a random draw from [0,60) produces the range [15,30) and hence results in the selection of individual 2. The FPS selection process can be thought of as spinning a roulette wheel, where the fitter individuals are allocated more space on the wheel (Fig. 3.5).

Selection pointer

4 123 **Fig. 3.5.** Fitness-proportionate selection with the area on the roulette wheel cor- responding to the fitness of each member of the population. Here individual 2 is selected

Although this method of selection is intuitive, it can produce poor results in practice as it embeds a high selection pressure in the early stage of the GA. Under FPS, the expected number of offspring for each encoding in the population is given by the corresponding solution PPavgobs

, and where Pavg Pobs is the is the average observed performance performance of all (fitness) solutions of

in the current population.

3.6 Generating Diversity 33

Commonly, in the early stage of the search process there is a high variance in the fitness of solution encodings, with a small number of encodings being notably fitter than the others. When FPS is used, these encodings and their descendants can quickly overrun the entire population (better-quality solution encodings may be chosen for replication several times in a single generation), and lead to the premature convergence of the population and stagnation of the algorithm in a local optimum.

Conversely, FPS can result in low selection pressure later in the GA run, as the population and the associated fitness values of individuals converge. When fitness values of individuals are very similar, each individual has an almost uniform chance of selection, and hence, slightly better solutions find it difficult to strongly influence future populations.

**Ordinal Selection**

One approach to overcome the problems of fitness-proportionate selection is to use rank-based selection. In rank-based selection, individuals are ranked from best to worst based on their raw fitness and this rank information is used to calculate a rescaled fitness for each individual. The rescaled fitness values rather than the original fitness values are used in the selection process. An example of a linear ranking process is provided by [25]:

frank = 2 − P + 2(P − 1)(rank (n − − 1) 1)

. (3.5)

In (3.5) rank is the ranking of an individual member of the population (the least fit individual has a rank of 1, and the most fit has a rank of n), there are n members of the population, and P is a scaling factor ∈ [1.0,2.0] which determines the selection pressure.

To illustrate the operation of the linear ranking process, assume that n = 5 and let P = 2. Table 3.6 lists five members of a sample population in order of their raw fitness (the least fit individual is ranked number 1), and also lists their rescaled fitnesses and their selection probabilities. These fitness values are then used in the roulette wheel selection process described above and higher ranking individuals are clearly more likely to be selected.

**Table 3.6.** Rank ordering and selection fitness

**Ranking** 1 2 3 4 5

Rescaled fitness 0 0.5 1 1.5 2 Selection probability 0 0.10 0.20 0.30 0.40

An advantage of rank-based selection is that it lessens the risk of biasing the search process as a result of too-intensive selection of the better solutions in the early generations of the GA. Another advantage of rank-based selection

34 3 Genetic Algorithm

is that it only requires relative (as distinct from absolute) measures of fitness. This could be an advantage if fitness measures are noisy. scheme, Another the top rank-based 1

th of the method n individuals of selection is truncation selection. In this in the population each get n copies in the mating pool. For example, if there are 100 members of the current population and the truncation rate is set at the population are each copied twice to create 1

2, then the the 50 fittest members of

mating pool. A commonly used, and computationally efficient, rank-selection method is tournament selection (Fig. 3.6). Under tournament selection, k members are chosen randomly without replacement from the population. The fittest of these is chosen as the tournament winner and is ‘selected’ to act as a parent. Assuming a population of size N, the value of k can be varied from 2, . . . , N. Lower values of k provide lower selection pressure, while higher values provide higher selection pressure. For example, if k = N, the fittest individual in the current population is always the tournament winner.

f4=12 f5=4 f1=9

f6=7 *Contestants* f5=4

f7=2 f1=9

f2=5

f6=7

f3=1

*Population*

f6=7

f1=9 *Winner*

**Fig. 3.6.** Tournament selection where k = 4. Four individuals are randomly chosen from the population with the fittest of these individuals winning the tournament and being selected for reproduction

n

3.6 Generating Diversity 35

As selection works on phenotypes (and their related fitness) it is ‘represen- tation independent’. This is not the case for the diversity generating operators of mutation and crossover.

**3.6.2 Mutation and Crossover**

The mutation operator plays a vital role in the GA as it ensures that the search process never stops. In each iteration of the algorithm, mutation can potentially uncover useful novelty. In contrast, crossover, if applied as a sole method of generating diversity, ceases to generate novelty once all members of the population converge to the same genotype.

The rate of mutation has important implications for the usefulness of se- lection and crossover. If a very high rate of mutation is applied, the selection and crossover operators can be overpowered and the GA will effectively re- semble a random search process. Conversely, if a high selection pressure is used, a higher mutation rate will be required in order to prevent premature convergence of the population. In setting an appropriate rate of mutation, the aim is to select a rate which helps generate useful novelty but which does not rapidly destroy good solutions before they can be exploited through selection and crossover. In contrast to mutation, crossover allows for the inheritance of groups of ‘good genes’ or building blocks by the offspring of parents, thereby encouraging more intensive search around already discovered good solutions. There is a close link between the choice of genotype representation and the design of effective mutation and crossover operators. Initially, mutation and crossover mechanisms for binary encodings are discussed, followed by the consideration of what modifications should be made to these for real-valued encodings.

**Binary Genotypes**

The original form of crossover for binary-valued genotypes was single point crossover (Fig. 3.7). A value pcross is set at the start of the GA (say at 0.7) and for each pair of selected parents, a random number is generated from the uniform distribution U(0,1). If this value is < 0.7, crossover is applied to generate two new children; otherwise crossover is bypassed and the two children are clones of their parents. Crossover rates are typically selected from the range pcross ∈ (0.6,0.9) but, if desired, the rate of crossover can be varied during the GA run.

One problem of single point crossover, is that related components of a solution encoding (schema) which are widely separated on the string tend to be disrupted when this form of crossover is applied. One way of reducing this problem is to implement two point crossover (Fig. 3.8), where the two cut positions on the parent strings are chosen randomly and the segments between the two positions are exchanged.

36 3 Genetic Algorithm

**Parent 1**

0 1 0 1 1 1 0 0

**Parent 2**

0 0 1 1 0 1 0 1

**Child 1**

0 1 0 1 0 1 0 1

**Child 2**

0 0 1 1 1 1 0 0

**Fig. 3.7.** Single point crossover where the cut-point is randomly selected after the third locus on the parent genotypes. The head and tail of the two parents are mixed to produce two child genotypes

**Parent 1**

0 1 0 1 1 1 0 0

**Parent 2**

0 0 1 1 0 1 0 1

**Child 1**

0 1 1 1 0 1 0 0

**Child 2**

0 0 0 1 1 1 0 1

**Fig. 3.8.** Two-point crossover

Another popular form of crossover is uniform crossover. In uniform crossover, a random selection of gene value is made from each parent when filling each corresponding locus on the child’s genotype. The process can be repeated a second time to create a second child, or the second child could be created using the values not selected when producing the first child (Fig. 3.9). To im- plement the latter approach, a random number r is drawn from the uniform distribution U(0,1) for each locus. If r < 0.5, child 1 inherits from parent 1; else, it inherits from parent 2, with child 2 being comprised of the bit values not selected for child 1.

For binary genotypes, a mutation operation can be defined as a bit-flip, whereby a ‘0’ can be mutated to a ‘1’ or a ‘1’ to a ‘0’. Figure 3.10 illustrates

**Parent 1**

**Parent 2**

**Child 1**

**Child 2**

**Fig. 3.9.** Uniform crossover where a random choice is made as to which parent donates a bit to child 1. Child 2 is then constructed using the bits not selected for inclusion in child 1

an implementation of the mutation process, where pmut = 0.1. Five random numbers (corresponding to a genotype length of five bits) are generated from U(0,1) and if any of these are < 0.1 then the value of that bit is ‘flipped’. This mutation process is repeated for all child solutions generated by the crossover process.

Typical mutation rates for a binary-valued GA are commonly of the order pmut = requirement 1

L where L is the length of the binary string. Of course, there is no that the mutation rate must remain constant during the GA run (Sect. 3.7).

0.45 0.69 0.23 0.09 0.86

1 0 0 0 0

1 0 0 1 0

**Fig. 3.10.** Illustration of mutation, with the bit at the fourth locus being ‘flipped’

3.6 Generating Diversity 37

0.71 0.22 0.34 0.67 0.93

1 0 1 0 1

1 1 0 0 0

1 0 1 0 0

1 1 0 0 1

38 3 Genetic Algorithm

**Real-Valued Genotypes**

The crossover operator can be modified for real-valued genotypes so that (for example) elements from the string of each parent are averaged in order to produce the corresponding value in their child(ren) (Fig. 3.11). Figure 3.12 illustrates this geometrically in two dimensions.

More generally, the real values in each locus of the child may be calculated as P1+α(P2−P1), where P1 and P2 are the real values for that locus in each of the two parents, and α is a scaling factor randomly drawn from some interval (say [−1.5,+1.5]). This crossover operator defines a hypercube based on the location of the parents (Fig. 3.13).

**Parent 1**

3 -2 8

**Parent 2**

1 -4 9

**Child**

(3+1)/2 (-2 -4)/2 (8+9)/2

**Fig. 3.11.** Simple real-valued crossover with two parents producing a single child

(1,1)

(2,2)

(1.5,1.5)

**Fig. 3.12.** Simple real-valued intermediate crossover with two parents (1,1) and (2,2) producing a single child at (1.5,1.5)

Many alternative mutation and crossover schemes for real-valued encod- ings exist. For example, a simple strategy for modifying mutation for real- valued encodings is to implement a stochastic mutation operator, where an element of a real-valued string can be mutated by adding a small (positive or negative) real value to it. Each element of the string xi could be mutated by adding a random number drawn from the normal distribution N(0,αi), where

3.6 Generating Diversity 39

(1,1) (2.5, -0.5)

(2.5, 2.5)

(2,2)

(-0.5, -0.5)

(-0.5, 2.5)

**Fig. 3.13.** Hypercube defined by crossover operator where parents are (1,1) and (2,2), with α ∈ [−1.5, 1.5]

the standard deviation αi is defined by the user. This mutation scheme will produce relatively small mutations most of the time, with occasional larger mutation steps.

**3.6.3 Replacement Strategy**

In deciding which parents and children survive into the next generation a wide variety of replacement strategies can be applied, including:

i. direct replacement (children replace their parents), ii. random replacement (the new population is selected randomly from the

existing population members and their children), iii. replacement of the worst (all parents and children are ranked by fitness

and the poorest are eliminated), and iv. tournament replacement (the loser of the tournament is selected for re-

placement).

In the canonical GA, a generational replacement strategy is usually adopted. The number of children produced in each generation is the same as the cur- rent population size and during replacement the entire current population is replaced by the newly created population of child encodings.

The ratio of the number of children produced to the size of the current population is known as the generation gap. Hence, the generation gap is typ- ically 1.0. It is also possible to create more offspring than members of the current population (generation gap > 1), and then select the best n (where n is the population size) of these offspring for survival into the next generation. Many variants on the replacement process exist. As already seen, the num- ber of children produced need not equal the current population size, and the automatic replacement of parents by children is not mandatory. A popular

40 3 Genetic Algorithm

strategy is steady state replacement, where only a small number of children (sometimes only one) are created during each generation, with only a small number of the current population, usually the least fit, being replaced during each iteration of the GA. For example, the worst x members of the current population could be replaced by the best x children. Adopting a steady state replacement strategy ensures that successive populations overlap to a signifi- cant degree (parents and their children can coexist), requires less memory, and allows the GA to exploit good solutions immediately after they are uncovered. Generally, fitness-based selection is implemented for either parent selection or replacement selection, not both. If fitness-based selection is implemented for both, very strong selection pressure will be created, leading to very rapid convergence of the population and poor search of the solution space.

Another common replacement strategy is elitism, whereby the best mem- ber (or several best members) of the current population always survive into the next population. This strategy ensures that a good individual is not lost between successive generations.

Some GA applications use crowding operators to supplement their replace- ment strategy. In order to encourage diversity in the population of solution encodings, a new child solution is only allowed to enter the population by re- placing the current member of the population which is most similar to itself. The objective is to avoid having too many similar individuals (crowds) in the new population.

**3.7 Choosing Parameter Values**

When applying the GA to real-world problems the user has to choose values for several parameters including the rate of mutation, the rate of crossover, and the size of the population, in addition to selecting the form of selection, the replacement strategy, etc. Even if attention is restricted to the choice of good crossover and mutation rates, the modeller faces a nontrivial problem in selecting these. A common approach in tuning the parameters for a GA application is to undertake a series of trial and error experiments before mak- ing parameter choices for the final GA runs (computer simulations). However, this approach is problematic as it can be time-consuming and good choices for these parameters are unlikely to remain constant over the entire GA run. Rather than selecting static parameter values, an alternative approach is to dynamically adapt the parameters during the run. There are three broad methods of dynamically adapting parameter settings (Fig. 3.14) [175]. Deterministic methods of parameter control vary parameter settings during the GA run, without using any feedback from the search process. An example of a deterministic rule for adapting the mutation rate is:

α(t) = α(t0)(1 − 0.8t/T) (3.6)

3.8 Summary 41

Dynamic Parameter Control

Deterministic Feedback Adaptive Evolve the Parameters

**Fig. 3.14.** Taxonomy of adaptive parameter control

where t (0 ≤ t ≤ T) denotes the current generation, α(t0) is a fixed value and α(t) is the adaptive mutation rate. This rule will reduce the value of the mutation rate during the run, biasing the GA towards increasing exploitation of current solutions as the run progresses.

Under a feedback adaptive process, the parameter values are altered based on feedback from the algorithm. If the composition of the population has converged (perhaps measured using the entropy of the population of binary strings) to a threshold level, the mutation rate could be increased by a pre- specified rate.

Another possibility is to have the GA evolve good choices for its param- eters. Under this idea, the GA is double-tasked: both to self-calibrate and to find a good solution to the problem at hand. A comprehensive description of dynamic parameter adaption is provided in [175].

**3.8 Summary**

This chapter presented an introduction to the best-known evolutionary algo- rithm, the genetic algorithm. The canonical GA is based on a very simpli- fied abstraction of evolutionary processes, usually employing fixed-size pop- ulations, unisex individuals, stochastic mating, and ignoring the child-adult development process [144].

GAs, since their introduction, have been shown to be powerful problem solvers and have been successfully applied to solve a large number of real- world optimisation problems. The methodology has particular utility when traditional techniques fail, either because the objective function is ‘hard’ (for example, noncontinuous), or because the landscape is highly multimodal. The parallel nature of a GA search process makes it less vulnerable to local optima than traditional hill climbing optimisation methods. However, it is important to note that the GA is not a ‘black box’ optimiser. While a canonical GA may obtain good results when applied to a real-world problem, obtaining the best results usually requires a careful design of the algorithm and the careful use of any domain knowledge available.

Despite the good properties of GAs, they, like all optimisation techniques, are subject to limitations. There is no guarantee that an optimal solution

42 3 Genetic Algorithm

will be found in finite time and progress towards better solutions may be intermittent rather than gradual. Consequently, the time required to find a high-quality solution to a problem is not determinable ex ante. The GA, and indeed all evolutionary optimising methodologies, rely on feedback in the form of fitness evaluations. For some problems, measuring fitness can be difficult (perhaps fitness can only be assessed subjectively by a human) or expensive in terms of cost or computation time. In these cases, GA may not be the most suitable choice of optimising technique.

In this chapter, we have outlined the primary components and principles upon which the GA is based. The next chapter describes a number of exten- sions of the GA model.

**4Extending the Genetic Algorithm**

The previous chapter provided an overview of the main concepts behind the GA. Since the introduction and popularisation of the GA, a substantial body of research has been undertaken in order to extend the canonical model and to increase the utility of the GA for hard, real-world problems. While it is beyond the scope of any single book to cover all of this work, in this chapter we intro- duce the reader to a selection of concepts drawn from this research. Many of the ideas introduced in this chapter have general application across the mul- tiple families of natural computing algorithms and are not therefore limited to GAs. The chapter concludes with an introduction to Estimation of Dis- tribution Algorithms (EDAs). EDAs are an alternative way of modelling the learning which is embedded in a population of genotypes in an evolutionary algorithm and have attracted notable research interest in the GA community in recent years.

**4.1 Dynamic Environments**

Many of the most challenging problems facing researchers and decision-makers are those with a dynamic nature. That is, the environment in which the so- lution exists, and consequently the optimal solution itself, changes over time. Examples of dynamic problems include trading in financial markets, time series analysis of gene expression data, and routing in telecommunication net- works. Biological organisms inhabit dynamic environments and mechanisms have arisen to promote the ‘survivability’ of biological creatures in these en- vironments. These mechanisms are useful sources of inspiration in helping us to design computer algorithms to attack real-world problems in dynamic environments.

© Springer-Verlag Berlin Heidelberg 2015 A. Brabazon et al., *Natural Computing Algorithms*, Natural Computing Series, DOI 10.1007/978-3-662-43631-8\_4

43

44 4 Extending the Genetic Algorithm

**4.1.1 Strategies for Dynamic Environments**

In designing an evolutionary algorithm for application in a dynamic environ- ment, the nature of the environmental changes will determine the appropriate strategy. For example, if change occurs at a slow pace, adapting the rate of mutation in a GA may be sufficient to allow the population to adjust to a slowly changing location for the global optimum. If the environment alters in a cyclic fashion, a memory of good past solutions may be useful. On the other hand, if the environment is subject to sudden discontinuous change then more aggressive adaptation strategies will be required. Hence, we can adopt a variety of strategies, including [302]:

• restart of the learning process,

• generation of more genotypic diversity if environmental change is detected,

• maintenance of genotypic diversity during the GA run,

• use of a memory mechanism to retain good past solutions (assumes cycling solutions), and

• use of multiple populations.

In an extreme case, it may be necessary to restart the learning process as past learning embedded in the population is no longer useful. More generally, if past learning still provides some guide to finding good solutions in the current environment, the focus switches to how best to adapt the current population in order to track the optimal solution as it changes. The following subsections discuss various aspects of diversity generation and maintenance. The use of multiple populations is discussed in Sect. 4.2.

**4.1.2 Diversity**

Maintaining diversity in the population of genotypes is important in all EC applications. Even in static environments, a population needs diversity in order to promote a good exploration of the search space. The role of diversity is even more important when faced with a dynamic environment. In the absence of any countermeasures, the canonical GA will tend to lose genotypic diversity during its run as selection and crossover will tend to push the population to a small set of genotypic states; hence the canonical algorithm needs some modification when it is applied in a dynamic environment.

Depending on the expected rate of environmental change, the modeller may decide to maintain a high degree of populational diversity at all times (useful if the environment has high rate of change), or generate it ‘on demand’ when a change in the environment is detected. At first glance it may appear that the better option is to maintain populational diversity at all times. How- ever, maintaining diversity has a cost, either in terms of having a larger pop- ulation, or in terms of less intensive exploitation of already discovered good regions. If the environment only changes occasionally, generation of diversity when environmental change is detected may be the better option.

4.1 Dynamic Environments 45

**Diversity Generation if Change Is Detected**

Cobb’s hypermutation strategy [115] was one of the earliest approaches to varying the rate of diversity generation as changes in the fitness landscape are detected. In this approach, if a change in the fitness landscape is detected, the base mutation rate of the GA is multiplied by a hypermutation factor. The size of the factor determines its effect; so if it is very large, it is equivalent to randomly reinitialising the entire population.

A common approach in the detection of environmental change is to use a sentry strategy. In a sentry strategy the fitness of a number of fixed genotypes (a form of memory) is monitored throughout the run. If the environment changes, the fitness of some or all of the locations of these sentries will alter and this provides feedback which is used to set the rate of mutation of the GA. If a large change in fitness occurs, indicating that the environment has changed notably, the rate of mutation is increased.

The sentry strategy can be applied in a number of ways. The sentries can remain outside the adapting population of solutions or they can be available for selection and crossover. In the latter case, while the sentries can influence the creation of new child solutions, they remain ‘fixed’ in location and are not mutated or replaced. Morrison [420] provides a discussion of quality strategies for sentry location, finding that random location often provides good results. In addition to providing information on whether the environment is changing, a sentry strategy can also provide information on where it is changing, thereby providing feedback on whether the changes are local or global.

**Diversity Maintenance During Run**

Rather than waiting for environmental change to occur and then playing ‘catch-up’, a strategy of maintaining continual diversity in the population can be followed. A wide variety of methods can be used for this purpose, including:

• weakening selection pressure,

• continual monitoring of populational diversity,

• restricted mating/replacement,

• fitness-sharing/crowding, and

• random immigrants.

Strong selection pressure implies that the GA will intensively sample current high fitness individuals, leading, if unchecked, to a rapid convergence of the population to similar genotypic forms. This can make it difficult for the pop- ulation to adapt if environmental change occurs, particularly if the change occurs in a region of the landscape which is not currently being sampled by the population of solutions. Hence, the use of a lower selection pressure will help maintain diversity in the population of genotypes. Another related con- sideration when implementing a GA in a dynamic environment is what form

46 4 Extending the Genetic Algorithm

of selection and replacement strategy to implement. The steady-state GA can offer advantages over the canonical generational GA. It allows a quicker re- sponse to a shift in the environment, as high-quality, newly created children are immediately available for mating purposes.

The degree of diversity of a population can be continually monitored in real time as the GA runs. Populational diversity can be defined on many levels, including diversity of fitness values and diversity of phenotypic or genotypic structures. Multiple measures of diversity can be defined for each of these. For example, diversity in a collection of real-valued fitnesses could be measured using the standard deviation of those values. However measured, if population diversity falls below a trigger level, action can be taken to increase diversity by raising the level of mutation or by replacing a portion of the population by newly created random individuals. Under a restricted mating or restricted replacement strategy, individuals which are too similar are not allowed to mate, and in a restricted replacement strategy a newly created child is precluded from entering the population unless it is sufficiently different to existing members of the population. The object in both cases is to avoid convergence of the population to a small subset of genotypes.

A fitness-sharing mechanism [213] aims to reduce the chance that a mul- titude of similar individuals will be selected for reproduction, thereby reduc- ing the genetic diversity of subsequent generations. An example of a fitness- sharing mechanism is:

f (i) = f(i)

(4.1)

where f(i) represents the original raw fitness of individual i. If there are a number of individuals which are similar to i in the population, its fitness for use in the selection process is reduced. The shared (reduced) fitness of individual i is denoted as f (i), and this corresponds to i’s original raw fitness, derated or reduced by an amount which is determined by a sharing function. The sharing function s as in (4.2) provides a measure of the density of the population within a given neighbourhood of i. For any pair of individuals i, j in the population, the sharing function returns a value of ‘0’ if i and j are more than a specified distance t apart (Fig. 4.1), and a value of ‘1’ if they are identical.

s(d) =

{1 − ( d

t)α if d<t; 0 otherwise. (4.2)

where d is a measure of the actual distance between two solutions and α is a scaling constant. To provide intuition on the sharing formula, if two individu- als in the current population are virtually identical, the distance between them is close to zero. Consequently, the raw fitness of each individual is reduced by 50%, reducing each individual’s chance of being selected for reproduction.

n∑j=1s(d(i, j))

4.1 Dynamic Environments 47

**s(d(i,j))**

1.0

0.0 t

d(i,j)

**Fig. 4.1.** Sharing function where α = 1. As the distance between two solutions increases, the degree of fitness sharing between them decreases

In a random immigrants strategy [219] a portion of the population is replaced by new randomly created individuals in each generation. This ensures that there is a constant inflow of diverse individuals into the search process. Usually either the worst, or random, members of the current population are replaced by the immigrants.

**Memory**

If the environment is thought to switch between a number of different ‘states’ then a sensible adaptation strategy is to build up and maintain a memory of good past solutions. These can be injected into the population if an en- vironmental change is detected. An interesting biological example of this is provided by our acquired immune system, which has the capacity to remem- ber the molecular signature of past invading pathogens, thereby enabling it to respond quickly if the pathogen is subsequently reencountered (Sect. 16.1.3).

**Measurement of Performance**

An important open question in EAs (and other methods for optimisation) is how best to measure performance in a dynamic environment. Ideally, we want a solution that performs well under the expected environment but which will not fail completely if the environment changes slightly. For example, suppose we are developing a delivery schedule for a large truck fleet. The resulting schedule should be reasonably robust to truck breakdowns or unexpected traffic delays. Similarly, a control program for a machine should be robust to changes in environmental conditions such as temperature or humidity.

A simple way of assessing the brittleness of a proposed solution is to under- take sensitivity analysis on the solution by perturbing it slightly and observing the resulting effect on fitness. Solutions which produce large changes in fitness

48 4 Extending the Genetic Algorithm

when perturbed slightly could therefore be assigned a lower ‘adjusted fitness’, with this value being used to drive the selection process in the EA.

**Summary**

The application of GAs, and other natural computing algorithms to dynamic problems has become a major area of natural computing research in recent years and many open issues remain. As noted in the last subsection, the appropriate definition of performance measures for these problems is not a trivial issue. Another open issue is the appropriate definition of diversity.

**4.2 Structured Population GAs**

Most evolutionary algorithms including the GA are panmictic in that any two members of the population can potentially mate with each other. The pop- ulation consists of a single pool of individuals and the operators of selection and crossover act on the entire pool. An alternative approach is to imple- ment a structured population GA where the population is partitioned and mating is constrained to the individuals within each partition. Two popular versions of structured evolutionary algorithms exist, distributed EAs (dEAs) and cellular EAs (cEAs). Implementing a structured GA can lead to com- putational efficiencies as the evolutionary process can be parallelised across multiple computers or processors (a good overview of parallelisation tech- niques in evolutionary algorithms is provided in [11]) and it can also help maintain population diversity.

**Distributed EA**

The dEA is inspired by the concept of species which are simultaneously evolv- ing on geographically dispersed islands in an ocean. It is also known as the island model because of this. In dEA, several separate subpopulations are created and each commences its own evolutionary process. Periodically, fit in- dividuals are allowed to migrate between the subpopulations. The migrations promote the sharing of information from already-discovered good solution en- codings, while maintaining genotypic diversity between the subpopulations. Island versions of the GA are natural candidates for parallel implementation as the evolutionary process on each island can be assigned to an individual processor.

In implementing the island model, decisions must be made concerning how often migration events occur between subpopulations, how individuals are se- lected for migration, how many individuals are selected for each migration event, and what replacement process is applied to refill the subpopulation when members of it migrate to another subpopulation. A variety of migration

4.2 Structured Population GAs 49

Population 1

Population 2 Population 4

Population 3

Population 1

Population 2

Population 4

Population 3

**Fig. 4.2.** Two examples of an island topology. The top network has unrestricted migration between all islands and the bottom network has a ring migration topology, where individuals can only migrate to one adjacent island

strategies can be used. For example, migration of individuals from one popu- lation to another may be unrestricted, or it may be confined to a predefined neighbourhood for each population (Fig. 4.2).

Illustrating one implementation of an island model, suppose there are four subpopulations with an unrestricted migration structure. A migration pool can be created for each subpopulation consisting of individuals selected from the other three subpopulations (perhaps their most fit individual). For each subpopulation in turn, a random selection is then made from its migration pool, with this individual replacing the worst individual in the subpopulation it enters.

**Cellular EA**

In cEA each individual genotype is considered as occupying a cell in a lattice (or graph) structure (Fig. 4.3). The operations of selection and recombination are constrained to take place in a small neighbourhood around each individual. When a cell is being updated, two parents are selected from its surrounding

50 4 Extending the Genetic Algorithm

neighbourhood, and genetic operators are applied to the two parents to pro- duce an offspring, with this offspring replacing the current genotype stored in that cell.

Each cell has its own pool of potential mates and in turn is a member of several other neighbourhoods, as the neighbourhoods of adjoining cells over- lap. Therefore, in contrast to dEA, which typically has a few relatively large subpopulations, there are typically many small subpopulations in cEA.

**Fig. 4.3.** A grid structure where a neighbourhood is defined around an individual (here the shaded cell)

In implementations of cEAs, updates of the state of each cell can be syn- chronous, where all cells are updated simultaneously using the cell contents in the current lattice. As the new genotypes are created, they are copied across to the next generation’s lattice one at a time. Alternatively, the update process can be asynchronous, where the lattice is updated one cell at a time so that new genotypes can influence the update process as soon as they are created. One method of asynchronous update is to update all cells sequentially from left to right, and from line to line, starting from the top left corner (fixed line sweep). Another method of asynchronous update is to randomly select (with uniform probability and with replacement) which cell to update during each time step (uniform choice).

**4.3 Constrained Optimisation**

Many important problems consist of attempting to maximise or minimise an objective function subject to a series of constraints. The constraints serve to bound the feasible region of valid solutions, possibly to a very small subset of the entire (unbounded) search space (Fig. 4.4).

More formally, a constrained optimisation problem (assuming that the objective function is to be maximised) can be stated as follows: find the vector x = (x1,x2,...,xd)T,x ∈ Rd in order to:

Maximise f(x) (4.3)

Feasible region

**Fig. 4.4.** Feasible region for a maximisation problem bounded by the x and y axes and three other constraints

subject to

inequality constraints: gi(x) ≤ 0, i = 1,...,m (4.4) equality constraints: hi(x)=0, i = 1,...,r (4.5) boundary constraints: xmin i ≤ xi ≤ xmax i , i = 1,...,d. (4.6)

The boundary constraints can be used to enforce physical conditions such as mass ≥ 0, etc. There are several approaches that can be taken when using a GA for constrained optimisation. A simple approach is to apply the GA as normal and assign zero fitness to any genotypes which generate an illegal so- lution which breaches one or more constraints. This strategy can be poetically referred to as the death penalty [409]. A problem with this approach is that even with low-dimensional problems it can produce a highly inefficient search process. If the problem is highly constrained, many generated genotypes may be illegal (for example, none of the initially randomly generated solutions might be feasible); hence much of the GA’s effort is wasted. There is also a risk that there could be over-rapid convergence on the first feasible solution found. As the dimensionality of the problem increases, the above problems are worsened as ratio of invalid solutions outside the feasible area to valid solutions inside the feasible area will rapidly increase. Two key issues arise when applying the GA to a constrained optimisation problem:

i. it may be difficult to generate an initial population of feasible genotypes,

and ii. crossover and mutation may act to convert a legal solution into an illegal

one.

4.3 Constrained Optimisation 51

Constraints

52 4 Extending the Genetic Algorithm

Three approaches which can ameliorate these issues include the use of penalty functions, the use of repair operators, and the creation of tailored diversity generation operations.

**Penalty Functions**

Rather than assign a zero fitness to genotypes which generate infeasible solu- tions, the objective function from which the fitness value is calculated may be supplemented by appending a penalty function to it. The greater the number of constraints breached, or the further the solution is from the feasible region, the lower the fitness assigned to it.

Obj∗(s) = Obj(s) − pen(s) (4.7)

pen(s) =

∑mi=1

wibi (4.8)

where bi =

{1, if s breaches constrainti;

0, otherwise.

In (4.7) the initial value of the objective function (assuming a maximisation problem) for solution s is reduced by a penalty function pen(s). This penalty is determined by the number of the i constraints that are breached by s, where each of these constraints can have a different weight (wi) (4.8).

This approach can help the GA to guide the population back to the feasible region. A penalty approach is most likely to work if there are relatively few constraints, as the greater the number of constraints, the harder it is to design an appropriate penalty function. Choices of weight values are important as, if weights are set too low, solutions may violate multiple constraints, if they are set too high, only feasible solutions will effectively be considered. This could lead to a collapse of populational diversity early in the algorithm, resulting in convergence to a locally optimal solution.

In order to overcome the latter problem, an adaptive penalty system could be implemented where low penalty weights are initially applied to facilitate explorative search, with the penalty weights being increased later in the GA run in order to force feasibility.

**Repair Operators**

Another approach is to attempt to repair infeasible solutions by moving them back into the feasible region. The ‘repaired’ solution then replaces the ille- gal solution in the population. Unfortunately, the design of an efficient repair mechanism can become tricky, particularly as the number of constraints in- creases (a simple repair operation in order to satisfy one constraint may pro- duce a breach of another constraint). Generally, effective repair mechanisms are problem-specific.

4.4 Multiobjective Optimisation 53

**Tailored Diversity-Generation Operators**

A problem when canonical crossover and mutation operations are applied to a genotypic encoding is that they do not respect the context of the problem domain. Hence, a crossover operation on two parents drawn from the feasible region may produce a child which is infeasible. Similarly, the application of a canonical mutation operator to a feasible genotype may produce one which results in an infeasible solution. One solution to this is to design problem- specific versions of crossover and mutation. An example of this is provided by the NEAT system in Sect. 15.2, which shows how populations of feedforward multilayer perceptrons (neural networks) can be evolved.

**4.4 Multiobjective Optimisation**

Decision-makers are often faced with having to make trade-offs between multi- ple, conflicting, objectives. There are usually also constraints on the solutions, such that not all solutions will be considered feasible (Sect. 4.3). Examples of multiobjective problems abound in the real world, ranging from finance to engineering. In the former case, investors typically seek to maximise their return whilst minimising a risk measure (for example, the variability of their return). Hence, the decision faced by the investor is how to allocate her invest- ment funds across multiple assets, so as to attain the highest possible return for a given level of risk. As we would expect, there is no unique solution to this problem, as higher expected returns will typically come at the cost of higher risk. In the case of engineering design problems, there is often a re- quirement to trade off (for example) the durability of a component and its weight/performance.

The multiobjective problem can be generally formulated as follows. As- sume that there are n objectives f1,...,fn and d decision variables x1,...,xd with x = (x1,...,xd), and that the decision-maker is seeking to minimise the multiobjective function y = f(x)=(f1(x),...,fn(x)). The problem therefore is to find the set (region) R ⊆ Rd of vectors x = (x1,x2,...,xd)T, x ∈ Rd in order to:

Minimise y = f(x)=(f1(x1,...,xd),...,fn(x1,...,xd)) (4.9)

subject to

inequality constraints: gi(x) ≤ 0, i = 1,...,m (4.10) equality constraints: hi(x)=0, i = 1,...,r (4.11) boundary constraints: xmin i ≤ xi ≤ xmax i , i = 1,...,d. (4.12)

Any specific member x of R (corresponding to a set of decision variable val- ues) will produce a unique y (vector of objective values in objective space)

54 4 Extending the Genetic Algorithm

and the above formulation aims to determine the members of R which are nondominated by any other member of R. This set is termed a Pareto set. For example, in the investor’s problem the aim is to find the investment weights for each of the k possible assets that the investor can include in her port- folio, such that the resulting expected risk-return outcome for that portfolio is nondominated by the expected risk-return output from any other possible portfolio. A member of the Pareto set is nondominated when (in the above example) for a given level of risk no other portfolio with a higher rate of return exists, or for a given level of return no other portfolio with a lower level of risk exists. Once the Pareto set is uncovered, the final choice of investment portfolio is determined by the individual investor’s risk preference. Graphi- cally, the Pareto set corresponds to a Pareto frontier (usually referred to as the efficient frontier) in objective space (Fig. 4.5). Return *Pareto frontier*

Risk

**Fig. 4.5.** Pareto frontier. The frontier corresponds to the set of investment port- folios that are Pareto optimal

Generally, a solution is termed Pareto optimal if an improvement on any one component of the objective function implies a disimprovement on an- other component. Hence, all solutions on the Pareto front are nondominated. Therefore, unlike single criterion problems, a multiobjective problem has mul- tiple, usually an infinite number of, solutions, rendering the calculation of the entire Pareto set infeasible. Typically instead, the object is to find a good approximation of the Pareto set given limited computational resources.

**Approaches to Multiobjective Optimisation**

A simple approach to multiobjective problems is to attempt to convert them into a single objective problem, to which regular optimisation techniques (in- cluding the GA) can be applied. One approach is to assign weights to each objective and then compute a single value for the entire objective function

4.4 Multiobjective Optimisation 55

using a weighted linear combination of the individual components of the ob- jective function for any given solution vector.

Clearly this method is subjective in that it requires the decision-maker to supply the relevant weights. Depending on the problem, the solution obtained can be sensitive to even small changes in these weightings. For a given set of weightings a single solution is obtained; hence, this approach will not identify all nondominated solutions.

Another way of converting a multiobjective problem to a single objective one is to move all but one component of the objective function to the constraint set, again obtaining a single solution for each specification of the constraints.

**Multiobjective Optimisation with a GA**

In using the GA for multiobjective optimisation, we are trying to closely approximate the true (but unknown) Pareto frontier and this requires that we generate a diverse set of nondominated solutions. A large literature has emerged over the past 20 years on the application of EC methods for multi- objective optimisation.

While a canonical GA can be directly applied to solve a multiobjective problem, a practical issue that arises is that typically the population will converge to a single solution (or small set of solutions) and hence will not uncover a representation of the entire Pareto frontier. One way to overcome this is to restart the GA multiple times and keep a record of the solutions found in each run. While this approach may eventually approximate the Pareto frontier, it is likely to prove computationally expensive as multiple runs may uncover the same points.

Hajela and Lin [239] proposed an alternative approach known as the weight-based genetic algorithm (WBGA) which uses a weighted sum approach to convert the multiobjective optimisation problem into a single objective problem. In WBGA each member of the population uses a different weight vector, which is generated randomly at the start of the run (with all weights summing to 1, ∑ni=1 wi = 1) and is then fixed. This weight vector is then applied in determining the fitness of each solution. An advantage of this ap- proach is that it allows a standard GA to search for multiple solutions in a single run.

One of the earliest published multiobjective GA applications, called vec- tor evaluated GA (or VEGA), was proposed by Schaffer [554]. This algo- rithm employs a ‘switching objective approach’ which aims to approximate the Pareto optimal set by a set of nondominated solutions. In the algorithm, the population Pt is randomly divided into K equally sized subpopulations P1,P2,...,PK, assuming that there are K objectives. Then, each solution in subpopulation Pi is assigned a fitness value using the objective function zi corresponding to that subpopulation. Hence, each subpopulation is evalu- ated using a different objective — thereby removing the difficulty in trying to determine the value of a solution under multiple objectives simultaneously.

56 4 Extending the Genetic Algorithm

Solutions are then selected from these subpopulations, using fitness-based selection, for crossover and mutation. Crossover and mutation are performed on the new population in the same way as for a single objective GA (Algorithm 4.1).

The VEGA approach is relatively easy to implement but suffers from the draw- back that it tends to produce solutions which perform well on one objective but poorly on others.

While early approaches to the application of GAs for multiobjective opti- misation relied on conversion of the problem into a single objective problem, later algorithms have been developed which directly tackle the multiobjective nature of the problem. In Pareto ranking approaches the fitness of a solution depends on its ranking within the current population (in other words, how many individuals is a specific individual dominated by), rather than on its actual objective function value; hence solutions are iteratively improved by focussing on the nondominated solutions in the current population. One of the earliest applications of this idea was Goldberg [211]. See Algorithm 4.2. Note that, here, a lower rank corresponds to a better quality solution. The set Fi are the nondominated fronts, and F1 is the nondominated front of the population.

Related studies which further developed this method included Fonseca and Fleming’s [196] Multiobjective Genetic Algorithm (MOGA), Srinivas and

**Algorithm 4.1:** VEGA Algorithm drawn from [336]

Let the subpopulation size be NS = N/K; Generate a random initial population P0; Set t = 0; **repeat**

Assign fitness value f(xi) = zk(xi) to the ith solution in the sorted population; Based on the fitness values assigned, select NS solutions between the (1 + (k − 1)NS)th and (kNS)th solutions of the sorted population to create subpopulation Pk; **end until** complete for each objective k = 1,...,K; Combine all subpopulations P1,...,Pk and apply crossover and mutation on the combined population to create Pt+1 of size N; Let t = t + 1; **until** terminating condition; Return Pt;

Randomly sort population Pt; **repeat**

**for** i =1+(k − 1)NS,...,kNS **do**

Deb’s Nondominated Sorting Genetic Algorithm (NSGA) [590] and NSGA- II [136]. Zitzler and Thiele’s Strength Pareto Evolutionary Algorithm (SPEA and SPEA-2) [683, 684] uses a ranking procedure to assign better fitness values to nondominated solutions in sparser regions of the objective space, thereby encouraging the uncovering of a dispersed set of nondominated solutions.

An excellent concise introduction to evolutionary multiobjective optimisa- tion is provided by Zitzler, Laumanns and Bleuler [682] and Konak, Coit and Smith [336]. Detailed coverage of the field can be found in [116] and [135].

**4.5 Memetic Algorithms**

Learning in populations of individuals takes place at several levels. At one extreme, the effects of long-term evolutionary learning is encoded in the pop- ulation of genotypes which make up a species. On a shorter time frame, in- dividuals are also capable of lifetime learning based on their own and on observed experience. In populations of social individuals, learning can also be transmitted by means of a shared culture (e.g. via education systems, legal systems, etc.).

In his famous book, The Selfish Gene [134], Richard Dawkins coined the term ‘meme’ to refer to ‘the basic unit of cultural transmission, or imitation’. Dawkins suggested that these memes were selected and processed by individ- uals and could be improved by the person holding them. As memes could also be passed from person to person, Dawkins argued that they displayed the key characteristics of an evolutionary process, namely, inheritance, variation and selection, thereby leading to a process of cultural evolution, akin to biological evolution.

**Algorithm 4.2:** Pareto-ranking Algorithm from [336]

**repeat**

Assign rank r1(x, t) = i if x ∈ Fi; **end until** terminating condition; Return Fi;

Set i = 1 and TP = P; Identify nondominated solutions in TP and assign them to the set Fi; Set TP = TPFi; **repeat**

i = i + 1; Identify nondominated solutions in TP and assign them to the set Fi; Set TP = TPFi; **until** TP = ∅; **for** every solution x ∈ P at generation t **do**

4.5 Memetic Algorithms 57

58 4 Extending the Genetic Algorithm

The concept of a dual evolutionary/lifetime learning mechanism could of course be included in a computational algorithm and the term memetic algo- rithm was first used by Moscato in 1989 [421] to describe an algorithm which combined both genetic (population-based) and individual (or cultural) learn- ing. Early memetic algorithms (MAs) typically consisted of an evolutionary algorithm that included a stage of individual optimisation/learning as part of the search strategy [257, 345, 346, 422]. For example, a local search step could be added to a canonical GA. The local search process could be as simple as pe- riodically performing a hill-climb around a subset of the better solutions in the current population in order to improve these solutions further. Improvements from the local search would be encoded on that individual’s genotype and would therefore be potentially transmissible to other members of the pop- ulation in subsequent generations. Practical design issues in creating these algorithms include choices as to how often individual learning takes place, which members of the population engage in individual learning, how long the individual learning process lasts, and what individual learning method should be used.

These forms of MA are known by a variety of names, including hybrid genetic algorithms, genetic local search algorithms and Lamarckian genetic algorithms. A practical motivation for these hybrid evolutionary algorithms is that while evolutionary algorithms such as the GA tend to be useful in identifying a good (high-fitness) region of a search space, they can be less effective in efficiently searching within this region [452, 486].

Although the early versions of these algorithms were loosely inspired by memetic concepts, they did not explicitly embed the notion of a population of memes which is adapting over time. Over the two decades since the intro- duction of MAs a wide body of literature has developed on this topic with a variety of algorithms being developed which explicitly include memetic adap- tation. In the multimeme MA, memes are directly encoded on an individual’s genotype, and determine the nature of the local refinement process which that individual applies. An alternative approach is that a pool of candidate memes compete for survival based on the degree of their past success in producing improvements during the local refinement step with better memes having a higher chance of survival into future generations. Hence, in both of the above approaches, there may be multiple local refinement (individual learning) ap- proaches encoded in the memes. Interested readers are referred to [488]. More generally, a significant body of literature has developed on memetic computa- tion defined by [487] as “a paradigm that uses the notion of meme(s) as units of information encoded in computational representations for the purpose of problem solving”. A detailed introduction to this field is provided by [487].

4.6 Linkage Learning 59

**4.6 Linkage Learning**

In genetics, multiple genes can interact in producing an effect at the phe- notypic level. This is known as epigenesis. Unfortunately, when individual elements of a genotype interact in this manner, it becomes much harder to find good gene values, as the GA implicitly needs to tease out the linkages between the relevant genes and then coevolve good sets of values for them. A particular problem of the canonical GA is that the selection and crossover operators can easily break up promising solutions, where there are epistatic links between dispersed elements of the genotype (Fig. 4.6).

As an example, consider a binary string encoding which is n bits long, where the first and the last bit must both be ‘1’ if the string is to have high fitness. If basic single-point crossover is applied to two parents, one of which already has the correct (‘1’) value in these locations, it is quite possible that neither child will inherit the good genes from that parent. In other words, be- cause no attention has been paid to the linkage structure between the elements of the string, the crossover operator has acted in a destructive manner.

Recognition of this problem led to the development of literature on linkage learning, where the object is to design crossover operators which do not disrupt important emerging partial solutions (building blocks) but which still ensure an effective mixing of partial solutions.

**Messy GA**

One approach to this problem is to attempt to reorganise the representation of the solution encoding so that functionally related elements will be (re)located close together on the reordered genotype. This reduces the chance that impor- tant links between genes will be broken by the crossover operator (Fig. 4.7). GA variants which have employed reordering operators include messy GA (mGA) and the linkage learning genetic algorithm. While the idea is sensible, a general problem with reordering approaches is that they tend to scale poorly as the length of the genotype increases.

**Competent GA**

An important direction in GA research stems from the recognition of the limited scalability of the canonical GA when it is applied to problems of in- creasing difficulty. It has been recognised that the success of a GA is dependent upon facilitating the proper growth and mixing of building blocks, which is not achieved by problem-independent recombination operators [212, 621]. The algorithms emerging from this area of research are dubbed competent GAs. Competent GAs seek to perform a more intelligent search by respecting the functionally important linkages between the constituent components of a so- lution in order to prevent the disruption of potentially useful building blocks. More recently, the GP community has applied ideas from competent GA in designing GP algorithms [552, 572].

60 4 Extending the Genetic Algorithm

*Epistatic linkage*

x1 x2 x3 x4 x5 x6 x7 x8

0 1 0 1 1 1 0 0

0 0 1 1 0 1 0 1

0 1 0 1 0 1 0 1

0 0 1 1 1 1 0 0

**Parent 1**

**Parent 2**

**Child 1**

**Child 2**

**Fig. 4.6.** Three of the genes are epistatically linked (x1, x7 and x8). As the genes are widely separated on the genome, application of a single-point crossover will tend to disrupt sets of good choices for these genes

x1 x7 x8 x4 x5 x6 x2 x3

**Fig. 4.7.** The three epistatically linked genes (x1, x7 and x8) are reordered so they are grouped together at the beginning of the genome. By grouping them together, there is less chance that the linked genes will be broken up by a single-point crossover operator

4.7 Estimation of Distribution Algorithms 61

**4.7 Estimation of Distribution Algorithms**

Estimation of distribution algorithms (EDAs) are an alternative way of mod- elling the learning which is embedded in a population of genotypes in an evo- lutionary algorithm. EDAs are a rapidly growing subfield within evolutionary computing and have several names, including probabilistic model building al- gorithms (PBMAs) and iterated density estimation evolutionary algorithms (IDEAs) [361, 500, 504]. Recent years have seen the application of EDAs to a range of problem domains, including multiobjective optimisation [329, 622], and dynamic optimisation [668].

100100 100100

100100

001100

001100

110101

110101

Probabilistic model

Probabilistic model

100101

100101

100101

001100

001100

*Population (tx) Selection Population (tx+1)* **Fig. 4.8.** Illustration of EDA with sampling from a probabilistic model replacing the crossover and mutation operators of canonical GA

Rather than maintaining a population of solution encodings from one generation to the next, and manipulating this population using selection, crossover and mutation, global statistical information is extracted from pre- vious iterations of the algorithm. This information is used to construct a posterior probability distribution model of promising solutions, based on the extracted information. New solutions are then sampled from this probabil- ity distribution (Fig. 4.8). Hence, EDAs maintain the selection and variation concepts from EAs but generate variation in a different way. Particular ad- vantages of EDAs over genetic algorithms include their lack of multiple pa- rameters (such as crossover and mutation rates) that require tuning and the transparency of the underlying probabilistic model used to guide the search process [360].

Examples of EDAs include population-based incremental learning (PBIL) [26, 27], the compact genetic algorithm (cGA) [252], and the Bayesian Opti- misation Algorithm (BOA) [503].

The general EDA methodology can be operationalised in many ways. For example, the design of the model update step depends on the assumptions made concerning the nature of the problem being addressed. Three main groups of EDAs exist; algorithms that assume that all variables are inde- pendent (univariate EDA models), those that assume restricted interactions between the variables (for example, bivariate dependencies between variables),