Natural Computing Series

Yossi Borenstein
Alberto Moraglio *Editors*

Theory and Principled Methods for the Design of Metaheuristics

Springer

Natural Computing Series

Yossi Borenstein • Alberto Moraglio
Editors

Theory and Principled Methods for the Design of Metaheuristics

Springer

Editors
Yossi Borenstein
VisualDNA
London
United Kingdom

Alberto Moraglio
University of Birmingham
School of Computer Science
Birmingham
United Kingdom

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
Vrije Universiteit Amsterdam
The Netherlands

ISSN 1619-7127 Natural Computing Series
ISBN 978-3-642-33205-0 ISBN 978-3-642-33206-7 (eBook)
DOI 10.1007/978-3-642-33206-7
Springer Heidelberg New York Dordrecht London

Library of Congress Control Number: 2013956610

For Aloe, Annette, and Shlomo—Y.B.

For Fumiyo, Giuseppina, and Giuseppe—A.M.

Foreword

I was very happy and excited when I was invited to write a foreword for the book "Theory and Principled Methods for the Design of Metaheuristics" by Alberto Moraglio and Yossi Borenstein. There are several reasons for this.

Firstly, the focus of the book—bridging the gap between theory and practice—is an exceptionally important and timely topic in the field of metaheuristics and is one that is very dear to me. Despite numerous attempts to change this, by and large theoreticians and practitioners live on different planets. Often theoretical studies on metaheuristics are hardly applicable to real-world problems: so why should practitioners take notice? Conversely, understanding theoretical results requires some discipline and effort, and I must say that practitioners, often unjustifiably, do not think that what they can learn from theory is worth the investment of their time, when in fact it is. My feeling is that this book will be an important reference in this area from which both theoreticians and practitioners can learn much.

Secondly, I was really impressed with the list of contributors to this edited book. This includes many of the most influential and respected researchers on metaheuristics. This book is a fantastic fusion of their collective knowledge. In terms of both contributions to the science and engineering of metaheuristics and inspiration for future researchers and practitioners, this book appears to have been a resounding success.

Finally, not too long ago, both Alberto and Yossi were doctoral students under my supervision. They were both exceptionally good students—independent thinkers with sharp minds and a strong motivation and will. It is both very satisfying and a source of inspiration for me to see them actively working with a number of world-class researchers on casting light on some of the most difficult yet crucially important issues in metaheuristics and much beyond, at a time when my own will is faltering.

I am sure I will learn a lot from studying this volume in detail, and I am sure many others, practitioners and theoreticians alike, can gain much from doing the same—Yossi and Alberto, thank you for having put together such a high-profile and inspiring volume.

Colchester, Essex, UK Riccardo Poli

Preface

Metaheuristics and evolutionary algorithms in particular are adaptable optimization frameworks that are routinely and successfully applied to hard real-world problems. Intuitively, they seem to capture some fundamental biological property which makes them inherently good general problem solvers. At the same time, the informal way in which metaheuristics are defined and tailored to each problem can lead to various misconceptions. More than anything else, their successful application is often the outcome of a long trial-and-error process to identify a good problem-specific design followed by extensive parameter tuning. Ideally, theoretical studies should rectify this situation by explaining how, when, and why metaheuristics work and providing guidelines for their successful design and optimal parameter choice. However, the challenge is huge: mathematical analysis requires time and effort even for very simple scenarios, whereas in practice problems as well as algorithms are quite complex and subject to rapid change. The different motivations—practical applications vs. mathematical analysis—lead to distinct subcommunities with different research cultures which rarely communicate with one another.

The dialog between theory and practice is very important to us. We organized a workshop at Parallel Problem Solving from Nature (PPSN) on this subject and later edited a special issue of the Evolutionary Computation Journal (ECJ). Our study is theoretical, but we always made an effort to identify and promote potential practical applications. In this book, rather than focusing on our own work, we collected several theoretical and principled methods that when strung together we believe indicate a viable route towards bridging theory and practice. The book outlines the contribution of current theoretical work to practical problems, points to various theoretical approaches that have the potential to have a real impact on practice in the future, and, at the same time, provides principled methods that can be applied now to make empirical work more rigorous. It is divided into four themes: the first three are related to theory and the last one to practice.

The Story Told in the Book

The wildest dream of those working with metaheuristics is to have a single universal search algorithm that could be applied unchanged to any problem and that would always deliver the optimal solution, efficiently. Claims along this line have been made in the past about genetic algorithms. It is clear, nowadays, that such an algorithm does not exist. One of the roles of theory is to defy claims of the sort. In the section "Theory for Drawing the Line," we give two examples of theoretical results that show what is *not* possible.

Results such as the no-free-lunch theorems show that it is necessary to make a compromise between the class of problems that a search algorithm is applied to and its overall expected performance. The most common counterargument to no-free-lunch theorems is that problems of practical interest are a very small subclass of all possible problems, and commonly used metaheuristics do well exactly on this class. This argument, however, begs the question of what really accounts for an interesting problem. In the section "Relevant Scope of Problems," we give three examples for possible ways of defining formally general classes of real-world problems.

The requirement to match problem class and search algorithm can be also looked at the other way around. Given a not too large rigorously defined class of problems, in principle it could be possible to design a search algorithm that is provably good for this class. In the section "Top-Down Principled Design of Search Algorithms," we give three examples of works that pursue this line of investigation.

The outline of a theory given above (i.e., formally defining an interesting general class of problems and then, accordingly, developing an optimal search algorithm for this class) has the potential to be the ultimate tool for practitioners. In principle, once the practitioner identifies a problem as a specific case of a more general class, he/she will have a choice of different optimal search algorithms designed for that class with guaranteed expected optimization time. Unfortunately, it is very challenging to find a balance between a class of problems which is broad enough to be practically interesting and yet is focused enough to admit an efficient search algorithm. Therefore, it is difficult to estimate when such a vision will become a practicable reality.

For the time being, it is therefore necessary to embrace an experimental approach to the application of metaheuristics to specific problems. Nonetheless, existing theory can be a guide for good practice. The section "Principled Practice" is about reasoned and systematic approaches to setting up experiments, metaheuristic adaptation to the problem at hand, and parameter settings. We give three examples of such works.

Overview of the Chapters

Theory for Drawing the Line

Knowing what is not possible avoids tempting but hopeless lines of research. The first two contributions present theoretical results that were developed as a response to empirical attempts to chase chimeras.

In the first chapter, “No Free Lunch Theorems: Limitations and Perspectives of Metaheuristics,” Christian Igel reviews the no-free-lunch theorems for search and optimization, and their implications for the design of metaheuristics are discussed. The no-free-lunch theorems show that it is not possible to develop a black-box search algorithm that is universally better than any other on every problem. Search algorithms must be tailored to the problem class at hand using prior knowledge to deliver good performance.

Fabien Teytaud and Olivier Teytaud consider in the second chapter, “Convergence Rates of Evolutionary Algorithms and Parallel Evolutionary Algorithms,” a large family of search algorithms that uses comparisons rather than absolute fitness values in the selection process. The focus on comparisons—even without considering specific classes of problems—is sufficient to demonstrate advantages in terms of robustness and, at the same time, drawbacks in terms of diminished performance. Practical implications of these results for evolutionary algorithms on parallel machines are discussed.

Relevant Scope of Problems

In order to design a “better than random search” algorithm, it is necessary to restrict the scope of the problems one considers. If the scope is too large, the gain in performance may be not practically relevant. If the scope is too narrow, the search algorithm may not be of any general interest (other than to the very specific problem at hand). If the scope excludes real-world problems, it will not be interesting, even if it encompasses a fairly large number of problems and it works substantially better than other algorithms on this class of problems. The scope needs to be defined rigorously; this will make it possible to: avoid improper claims, be a starting point for devising search algorithms matching it, and serve as a starting point to prove general results (on the performance of search algorithms) on this class of problems. The following three contributions describe research attempting to identify interesting classes of problems.

In Chap. 3, “Rugged and Elementary Landscapes,” Konstantin Klemm and Peter F. Stadler provide an introduction to the structural features of discrete fitness landscapes from both the geometric and the algebraic perspectives. In particular, the chapter focuses on elementary landscapes, which are a class of fitness landscapes that encompass several important real-world problems.

In Chap. 4, “Single-Funnel and Multi-funnel Landscapes and Subthreshold-Seeking Behavior,” Darrell Whitley and Jonathan Rowe introduce the classes of single-funnel and multi-funnel landscapes. These classes of problems are quite large; however, they capture the characteristics of many typical real-world problems. They show that a simple subthreshold-seeker algorithm performs provably better than random search on these classes.

Chapter 3 introduces an important class of problems but without devising a better than random search algorithm for that class. Chapter 4 provides such an algorithm for a very large class of problems; however, the size of this class limits the potential performance of the algorithm. In Chap. 5 “Black-Box Complexity for Bounding the Performance of Randomized Search Heuristics,” Thomas Jansen considers more specific classes of problems and provides optimal randomized search heuristics for those problems. This chapter highlights the importance of focusing on specific classes of problems. It also exemplifies, using the notion of black-box complexity, how one can theoretically prove optimality for black-box algorithms (and hence make any attempts to design better algorithms redundant).

Top-Down Principled Design of Search Algorithms

The features of the class of problems considered can be used to derive in a principled way search algorithms that, exploiting these properties, reach the best possible (average) result on the considered class. For example, for a strictly unimodal fitness landscape, we might want to use a steepest-descent local search algorithm. This way, we take advantage of the special feature of this problem—that any local optimum reached from any starting point is the global optimum. The first two contributions illustrate how to derive a search algorithm that is optimally matched in a certain sense with a probabilistic class of functions. Rather than considering an explicit class of problems, the third contribution shows how the well-known Covariance Matrix Adaptation Evolution Strategy (CMA-ES) was derived by exploiting a desirable property of such class.

In Chap. 6, “Designing an Optimal Search Algorithm with Respect to Prior Information,” Olivier Teytaud and Emmanuel Vazquez consider three approaches to derive an optimal search algorithm for a class of functions: experimentation (i.e., parameter tuning), a mathematical approach based on reinforcement learning, and a simplified version of the latter with more reasonable computational cost based on Gaussian processes.

In Chap. 7, “The Bayesian Search Game,” Marc Toussaint draws links between no-free-lunch theorems that, interpreted inversely, lay the foundation of how to design search heuristics that exploit prior knowledge about the function, partially observable Markov decision processes and their approach to the problem of sequentially and optimally choosing search points, and the use of Gaussian processes as a representation of belief, i.e., knowledge about the problem.

In Chap. 8, “Principled Design of Continuous Stochastic Search: From Theory to Practice,” Nikolaus Hansen and Anne Auger derive the well-known *covariance* *matrix adaptation evolution strategy*, which has been shown to work very well in continuous optimization occurring in practice. They show how this algorithm was developed based only on a few fundamental principles—namely, maximal entropy, unbiasedness, maintaining invariance, and, under these constraints, exploiting all available information and solving simple functions reasonably fast.

Principled Practice

The literature is rich with an ever-increasing number of new metaheuristics that have demonstrated, one way or another, their potential usefulness. Albeit, metaheuristics are far from being plug-and-play friendly: Given a problem, one has initially to choose which metaheuristic to use. Then, as metaheuristics are not ready-made search algorithms, it is necessary to spend a considerable amount of time on adapting the operators for the particular problem domain and tuning the parameters. The final three contributions suggest how to address these more practical issues in a reasoned and systematic way.

In Chap. 9, "Parsimony Pressure Made Easy: Solving the Problem of Bloat in GP," Riccardo Poli and Nicholas Freitag McPhee use Price's theorem to characterize mathematically the size evolution of programs and to derive theoretical results that show how to practically and optimally use the parsimony pressure method to achieve complete control over the growth of the programs in a population.

In Chap. 10, "Experimental Analysis of Optimization Algorithms: Tuning and Beyond," Thomas Bartz-Beielstein and Mike Preuss present a tutorial on methodological approaches for experimental research in evolutionary computation and metaheuristic optimization.

In the last contributed chapter entitled "Formal Search Algorithms + Problem Characterizations = Executable Search Strategies," Patrick D. Surry and Nicholas J. Radcliffe present a principled way to derive search operators for nonstandard solution representations which also takes into account the structure of the problem at hand.

London, UK — Yossi Borenstein
July 2012 — Alberto Moraglio

Acknowledgements

We would like to express our gratitude to the following referees who gave their time and energy to review the chapters of the book ensuring their quality:

Dimo Brockhoff
Ying-Ping Chen
Marco Chiarandini
Cecilia Di Chio
Tobias Friedrich
Mario Giacobini
Mario Graff
Li Hui
Christian Igel
Thomas Jansen
Colin Johnson
Bryant A. Julstrom
Yong-Hyuk Kim
Dario Landa-Silva
Per Kristian Lehre
Guanzhou Lu
Yusuke Nojima
Pietro Oliveto
Mike Preuss
Ramon Sagarna
Jonathan Shapiro
Terence Soule
Peter F. Stadler
Olivier Teytaud
Marc Toussaint
Carsten Witt
Yourim Yoon

Contents

1 **No Free Lunch Theorems: Limitations and Perspectives of Metaheuristics** 1
Christian Igel

2 **Convergence Rates of Evolutionary Algorithms and Parallel Evolutionary Algorithms** 25
Fabien Teytaud and Olivier Teytaud

3 **Rugged and Elementary Landscapes** 41
Konstantin Klemm and Peter F. Stadler

4 **Single-Funnel and Multi-funnel Landscapes and Subthreshold-Seeking Behavior** 63
Darrell Whitley and Jonathan Rowe

5 **Black-Box Complexity for Bounding the Performance of Randomized Search Heuristics** 85
Thomas Jansen

6 **Designing an Optimal Search Algorithm with Respect to Prior Information** 111
Olivier Teytaud and Emmanuel Vazquez

7 **The Bayesian Search Game** 129
Marc Toussaint

8 **Principled Design of Continuous Stochastic Search: From Theory to Practice** 145
Nikolaus Hansen and Anne Auger

9 **Parsimony Pressure Made Easy: Solving the Problem of Bloat in GP** 181
Riccardo Poli and Nicholas Freitag McPhee

10 Experimental Analysis of Optimization Algorithms: Tuning and Beyond 205
Thomas Bartz-Beielstein and Mike Preuss

11 Formal Search Algorithms + Problem Characterisations = Executable Search Strategies 247
Patrick D. Surry and Nicholas J. Radcliffe

List of Contributors

Anne Auger INRIA Saclay – Île-de-France, Orsay, France

Thomas Bartz-Beielstein Faculty of Computer Science and Engineering Science, Institute of Computer Science, Cologne University of Applied Sciences, Cologne, Germany

Nikolaus Hansen INRIA Saclay – Île-de-France, Orsay, France

Christian Igel Department of Computer Science, University of Copenhagen, Copenhagen, Denmark

Thomas Jansen Department of Computer Science, Aberystwyth University, Aberystwyth, UK

Konstantin Klemm Bioinformatics Group, Department of Computer Science, Interdisciplinary Center for Bioinformatics, University of Leipzig, Leipzig, Germany

Nicholas Freitag McPhee Division of Science and Mathematics, University of Minnesota, Morris, MN, USA

Riccardo Poli School of Computer Science and Electronic Engineering, University of Essex, Colchester, Essex, UK

Mike Preuss Algorithm Engineering, Department of Computer Science, TU Dortmund, Dortmund, Germany

Nicholas J. Radcliffe Stochastic Solutions Limited, Edinburgh, UK
Department of Mathematics, University of Edinburgh, Edinburgh, UK

Jonathan Rowe Department of Computer Science, University of Birmingham, Birmingham, UK

Peter F. Stadler Bioinformatics Group, Department of Computer Science, Interdisciplinary Center for Bioinformatics, University of Leipzig, Leipzig, Germany
Max Planck Institute for Mathematics in the Sciences, Leipzig, Germany

Fraunhofer Institut für Zelltherapie und Immunologie, Leipzig, Germany
Department of Theoretical Chemistry, University of Vienna, Wien, Austria
Santa Fe Institute, Santa Fe, NM, USA

Patrick D. Surry Portrait Software, Boston, MA, USA

Fabien Teytaud TAO, Inria Saclay IDF, LRI, University Paris-Sud, Paris, France

Olivier Teytaud TAO, Inria Saclay IDF, LRI, University Paris-Sud, Paris, France

Marc Toussaint Machine Learning & Robotics Lab, Free University of Berlin, Berlin, Germany

Emmanuel Vazquez Department of Computer Science and Information Engineering, National University of Tainan, Tainan, Taiwan
SUPELEC, Gif-sur-Yvette, France

Darrell Whitley Department of Computer Science, Colorado State University, Fort Collins, CO, USA

Chapter 1
No Free Lunch Theorems: Limitations and Perspectives of Metaheuristics

Christian Igel

Thus not only our reason fails us in the discovery of the ultimate connexion of causes and effects, but even after experience has informed us of their constant conjunction, it is impossible for us to satisfy ourselves by our reason, why we should extend that experience beyond those particular instances, which have fallen under our observation. We suppose, but are never able to prove, that there must be a resemblance betwixt those objects, of which we have had experience, and those which lie beyond the reach of our discovery. (David Hume, 1739 [10, 14])

Abstract The No Free Lunch (NFL) theorems for search and optimization are reviewed and their implications for the design of metaheuristics are discussed. The theorems state that any two search or optimization algorithms are equivalent when their performance is averaged across all possible problems and even over subsets of problems fulfilling certain constraints. The NFL results show that if there is no assumption regarding the relation between visited and unseen search points, efficient search and optimization is impossible. There is no well-performing universal metaheuristic, but the heuristics must be tailored to the problem class at hand using prior knowledge. In practice, it is not likely that the preconditions of the NFL theorems are fulfilled for a problem class and thus differences between algorithms exist. Therefore, tailored algorithms can exploit structure underlying the optimization problem. Given full knowledge about the problem class, it is, in theory, possible to construct an optimal algorithm.

C. Igel (✉)
Department of Computer Science, University of Copenhagen, Universitetsparken 5, 2100 Copenhagen, Denmark
e-mail: igel@diku.dk

Table 1.1 All possible functions $\{0,1\}^2 \rightarrow \{0,1\}$, which will be used as examples throughout this chapter

(x_1,x_2)	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
$(0,0)$	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
$(0,1)$	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
$(1,0)$	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
$(1,1)$	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

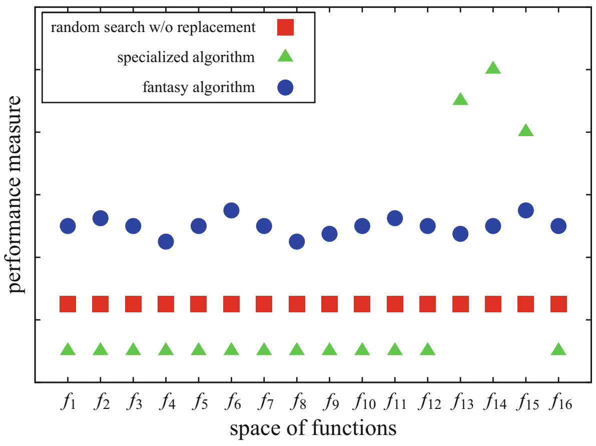


Fig. 1.1 Distribution of algorithm performance over a discrete space of problems. For random search without replacement the average performance on a function is shown. A performance profile such as depicted for the "fantasy algorithm" is not possible if the space has the property of being closed under permutation (e.g., as in the case of the problems shown in Table 1.1)

1.1 Introduction

Metaheuristics such as evolutionary algorithms, simulated annealing, swarm algorithms, and tabu search are general in the sense that they can be applied to any objective (target of fitness) function $f : \mathcal{X} \rightarrow \mathcal{Y}$, where $\mathcal{X}$ denotes the search space and $\mathcal{Y}$ a set of totally ordered cost-values. The goal of designing metaheuristics is to come up with search or optimization methods that are superior to others when applied to instances of a certain class of problems. In this chapter, we ask under which conditions can one heuristic be better than another at all and we derive answers based on the No Free Lunch (NFL) theorems.

Example 1.1. Let us consider all objective functions mapping from some finite domain $\mathcal{X}$ to some finite set of values $\mathcal{Y}$, for example, those given by all functions $\{0,1\}^2 \rightarrow \{0,1\}$ listed in Table 1.1. Figure 1.1 depicts the performance of three algorithms over this set of functions. Performance could, for example, be measured in terms of the number of objective function evaluations needed to find the optimum or by the quality of the best solution found in m steps (or by any other measure consistent with the definition given in the next section). The baseline is given by the average performance of a random search algorithm that picks search points uniformly at random without replacement (i.e., every point is visited only once). A highly specialized algorithm – which has been developed for many years based on extensive domain knowledge – may have a performance profile as indicated by the triangles. It performs extremely well on a very few problems, but very badly on all the others. Now, would it not be great to come up with an algorithm that performs well, in particular, better than random search, across *all* problems as indicated by the circles in Fig. 1.1? Could this be achieved by designing a metaheuristic using principled methods? Unfortunately, the answer to these questions is *no*.

The NFL theorem for optimization – and we do not want to distinguish between search and optimization in this chapter – roughly speaking, states that all non-repeating search algorithms have the same mean performance when averaged uniformly over *all* possible objective functions $f : \mathcal{X} \to \mathcal{Y}$ [6, 7, 21, 24, 35, 36]. In practice, of course, algorithms need not perform well on all possible functions, but only on a subset that arises from the application at hand.

In this chapter, we discuss extensions and implications of this fundamental result. In the next section, we introduce the basic notation and formally state the original NFL theorem as well as some of its refinements, in particular NFL results for restricted problem classes [20, 28, 29]. Section 1.3 discusses the optimization scenario underlying these theorems. Section 1.4 studies how many problem classes fulfill the prerequisites for a NFL result and if these problem classes are likely to be observed in real-world applications. It ends with a discussion of the Almost NFL theorem [6]. In the more research-oriented Sect. 1.5 the link between optimization problems and Markov decision processes is established, and it is shown how to construct optimal algorithms in this framework. Finally, the main results are summarized and further general conclusions are drawn.

1.2 The NFL Theorem for Search

In the following, we first fix the notation before we state the basic NFL theorem and its extensions.

1.2.1 Basic Definitions

Let us assume a finite search space $\mathcal{X}$ and a finite set of cost-values $\mathcal{Y}$. Let $\mathcal{F}$ be the set of all objective functions $f : \mathcal{X} \to \mathcal{Y}$ to be optimized (also called target, fitness, energy, or cost functions). NFL theorems make statements about non-repeating search algorithms (referred to as algorithms) that explore a new point in the search

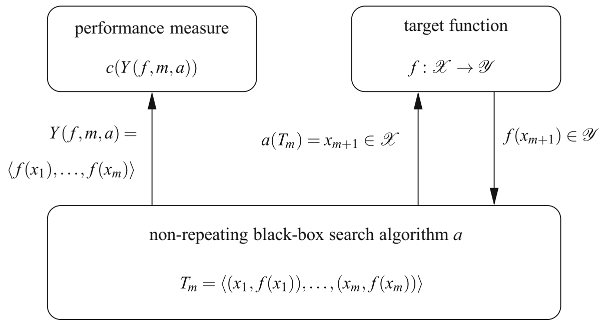

performance measure
$c(Y(f,m,a))$
target function
$f: \mathcal{X} \to \mathcal{Y}$
$Y(f,m,a) =$
$\langle f(x_1),\dots,f(x_m)\rangle$
$a(T_m) = x_{m+1} \in \mathcal{X}$
$f(x_{m+1}) \in \mathcal{Y}$
non-repeating black-box search algorithm a
$T_m = \langle (x_1, f(x_1)),\dots,(x_m, f(x_m))\rangle$

Fig. 1.2 Scheme of the optimization scenario considered in NFL theorems. A non-repeating black-box search algorithm a chooses a new exploration point in the search space depending on the sequence T_m of the already visited points with their corresponding cost-values. The target function f returns the cost-value of a candidate solution as the only information. The performance of a is determined using the performance measure c, which is a function of the sequence $Y(f,m,a)$ containing the cost-values of the visited points

space depending on the history of previously visited points and their cost-values. Non-repeating means that no search point is evaluated more than once. Let the sequence $T_m = \langle (x_1, f(x_1)), (x_2, f(x_2)), \dots, (x_m, f(x_m))\rangle$ represent m pairs of different search points $x_i \in \mathcal{X}, \forall i, j : x_i \neq x_j$ and their cost-values $f(x_i) \in \mathcal{Y}$. An algorithm a appends a pair $(x_{m+1}, f(x_{m+1}))$ to this sequence by mapping T_m to a new point x_{m+1} with $x_{m+1} \neq x_i$ for $i = 1, \dots, m$.

We assume that the performance of an algorithm a after $m \leq |\mathcal{X}|$ iterations with respect to a function f depends only on the sequence

$$Y(f,m,a) = \langle f(x_1), f(x_2), \dots, f(x_m)\rangle$$

of cost-values the algorithm has produced. Let the function c denote a performance measure mapping sequences of cost-values to the real numbers. Figure 1.2 depicts the optimization scenario assumed in NFL theorems.

Example 1.2. In the case of function minimization a performance measure that returns the minimum cost-value in the sequence could be a reasonable choice. Alternatively, the performance measure could return the number of objective function evaluations that were needed to find a search point with a cost-value below a certain threshold.

In general, one must distinguish between an algorithm and its search behavior. For finite $\mathcal{X}$ and $\mathcal{Y}$ and thus finite $\mathcal{F}$, there are only finitely many different non-repeating, deterministic search behaviors. For a given function f, there exist only $|\mathcal{X}|!/(|\mathcal{X}| - m)!$ different search behaviors corresponding to the possible orders in which the search points can be visited. In practice, two algorithms that always exhibit the same search behavior (i.e., produce the same sequence T_m given the same function f and number of steps m) may differ, for example, in their space and run-time requirements (see Sect. 1.3.1).

Example 1.3. Given some function $f : \{0,1\}^2 \to \{0,1\}$, there are 12 possible search behaviors for two-step search resulting in the following sequences:
$\langle((0,0), f((0,0))), ((0,1), f((0,1)))\rangle$, $\langle((0,0), f((0,0))), ((1,0), f((1,0)))\rangle$,
$\langle((0,0), f((0,0))), ((1,1), f((1,1)))\rangle$, $\langle((0,1), f((0,1))), ((0,0), f((0,0)))\rangle$,
$\langle((0,1), f((0,1))), ((1,0), f((1,0)))\rangle$, $\langle((0,1), f((0,1))), ((1,1), f((1,1)))\rangle$,
$\langle((1,0), f((1,0))), ((0,0), f((0,0)))\rangle$, $\langle((1,0), f((1,0))), ((0,1), f((0,1)))\rangle$,
$\langle((1,0), f((1,0))), ((1,1), f((1,1)))\rangle$, $\langle((1,1), f((1,1))), ((0,0), f((0,0)))\rangle$,
$\langle((1,1), f((1,1))), ((0,1), f((0,1)))\rangle$, $\langle((1,1), f((1,1))), ((1,0), f((1,0)))\rangle$.

Often we are not interested in statements that assume that each function in $\mathcal{F}$ is equally likely to be the objective function. Instead, we want to make statements that refer to *problem classes*. A reasonable general working definition of a problem class is:

Definition 1.1 (Problem Class). Given a finite search space $\mathcal{X}$ and a finite set of cost-values $\mathcal{Y}$, a *problem class* can be defined by a probability distribution $p_{\mathcal{F}}$ over the space $\mathcal{F}$ of all possible objective functions $f : \mathcal{X} \to \mathcal{Y}$, where $p_{\mathcal{F}}(f)$ is the probability that $f \in \mathcal{F}$ is the objective function faced by an algorithm.

In the special case that all functions that have a non-zero probability of being the target function are equally likely, we can identify a problem class by the subset $F \subseteq \mathcal{F}$ with $f \in F \Rightarrow p_{\mathcal{F}}(f) = 1/|F| > 0$.

1.2.2 The NFL Theorem

The original and most popular version of the NFL theorem for optimization was formally stated and proven by Wolpert and Macready in 1995 [35, 36] It can be expressed as:

Theorem 1.1 (NFL Theorem [36]). *For any two algorithms a and b, any $k \in \mathbb{R}$, any $m \in \{1,\dots,|\mathcal{X}|\}$, and any performance measure c*

$$\sum_{f\in\mathcal{F}} \delta(k, c(Y(f,m,a))) = \sum_{f\in\mathcal{F}} \delta(k, c(Y(f,m,b))) \ . \tag{1.1}$$

Herein, δ denotes the Kronecker function ($\delta(i,j) = 1$ if $i = j$, $\delta(i,j) = 0$ otherwise). Proofs can be found in [21, 27, 35, 36]. Equation (1.1) implies

$$\sum_{f\in\mathcal{F}} c(Y(f,m,a)) = \sum_{f\in\mathcal{F}} c(Y(f,m,b)) \tag{1.2}$$

for any two algorithms a and b, any $m \in \{1,\dots,|\mathcal{X}|\}$, and any performance measure c.

This proves that statements such as "averaged over all functions, my search algorithm is the best" are misconceptions. Radcliffe and Surry were among the first who appreciated and discussed this theorem [24]. Without rigorous proof, the basic NFL result was stated already earlier by Rawlins in 1991 [25].

When looking at the proof of the NFL theorem, it implies the following statements (see [33]):

Corollary 1.1. *For any $m \in \{1, \ldots, |\mathcal{X}|\}$ and any performance measure c we have:*

1. *For any two algorithms a and b, for each $f_a \in \mathcal{F}$ it holds*

$$c(Y(f_a, m, a)) = k \Rightarrow \exists f_b \in \mathcal{F} : c(Y(f_b, m, b)) = k \ . \tag{1.3}$$

2. *For any two algorithms a and b and any subset of functions $F \subset \mathcal{F}$ with complement $F^c = \mathcal{F} \setminus F$ it holds*

$$\sum_{f \in F} c(Y(f, m, a)) > \sum_{f \in F} c(Y(f, m, b)) \Rightarrow$$

$$\sum_{f \in F^c} c(Y(f, m, a)) < \sum_{f \in F^c} c(Y(f, m, b)) \ . \tag{1.4}$$

The first statement says that for any function $f_a \in \mathcal{F}$, there is a function $f_b \in \mathcal{F}$ on which algorithm b has the same performance as algorithm a on f_a. That is, if my algorithm outperforms your algorithm on some function, then there is also an objective function on which your algorithm outperforms mine. And if my algorithm outperforms yours on some set of benchmark problems, then your algorithm is better than mine averaged over the remaining problems.

1.2.3 The Sharpened NFL Theorems

Theorem 1.1 assumes that all possible objective functions in $\mathcal{F}$ are equally likely. Given that it is fruitless to design a metaheuristic for all possible objective functions, the question arises under which constraints can the average performance of one algorithm be better than another when the average is taken only over a subset $F \subset \mathcal{F}$. This is a relevant scenario if the goal is to develop metaheuristics for certain (e.g., "real-world") problem classes.

The NFL theorem has been extended to subsets of functions with the property of being closed under permutation (c.u.p.) Let $\pi : \mathcal{X} \to \mathcal{X}$ be a permutation of $\mathcal{X}$. The set of all permutations of $\mathcal{X}$ is denoted by $\Pi(\mathcal{X})$. A set $F \subseteq \mathcal{F}$ is said to be c.u.p. if for any $\pi \in \Pi(\mathcal{X})$ and any function $f \in F$ the function $f \circ \pi$ is also in F. In [27, 28] the following result is proven:

Theorem 1.2 (Sharpened NFL Theorem [27,28]). *If and only if F is c.u.p., then for any two algorithms a and b, any $k \in \mathbb{R}$, any $m \in \{1, \ldots, |\mathcal{X}|\}$, and any performance measure c*

$$\sum_{f \in F} \delta(k, c(Y(f, m, a))) = \sum_{f \in F} \delta(k, c(Y(f, m, b))) \ . \tag{1.5}$$

This is an important extension of Theorem 1.1, because it gives necessary and sufficient conditions for NFL results for subsets of functions.

Example 1.4. Consider again the mappings $\{0,1\}^2 \to \{0,1\}$, denoted by f_0, $f_1, \ldots, f_{15}$ as shown in Table 1.1. Then $\{f_1, f_2, f_4, f_8\}$ and $\{f_0, f_1, f_2, f_4, f_8\}$ are examples of sets that are c.u.p. The set $\{f_1, f_2, f_3, f_4, f_8\}$ is not c.u.p., because some functions are "missing". These missing functions include f_5, which results from f_3 by switching the elements $(0,1)$ and $(1,0)$.

Example 1.5. Theorem 1.1 tells us that on average all algorithms need the same time to find a desirable, say optimal, solution – but how long does it take? The average number of evaluations needed to find an optimum (the mean hitting time) depends on the cardinality of the search space $|\mathcal{X}|$ and the number n of search points that are mapped to a desirable solution. As the set of *all* functions where n search points represent desirable solutions is c.u.p., it is sufficient to compute the average time to find one of these points for an arbitrary algorithm, which is given by $(|\mathcal{X}| + 1)/(n + 1)$ [17] (a proof for $n = 1$ is given in Chap. 5).

In Theorems 1.1 and 1.2 it is implicitly assumed that each function in $\mathcal{F}$ and F, respectively, has the same probability to be the target function, because the summations in Eqs. (1.1) and (1.5) average uniformly over the functions. However, it is more realistic to assume that different functions can have different probabilities to be the target function. In this more general case, a problem class is described by a probability distribution assigning each function f its probability $p_{\mathcal{F}}(f)$ to be the objective function.

To derive results for this general scenario it is helpful to introduce the concept of $\mathcal{Y}$-histograms. A *$\mathcal{Y}$-histogram* (*histogram* for short) is a mapping $h : \mathcal{Y} \to \mathbb{N}_0$ such that $\sum_{y \in \mathcal{Y}} h(y) = |\mathcal{X}|$. The set of all histograms is denoted by $\mathcal{H}$. Any function $f : \mathcal{X} \to \mathcal{Y}$ implies a histogram $h_f(y) = |f^{-1}(y)|$ that counts the number of elements in $\mathcal{X}$ that are mapped to the same value $y \in \mathcal{Y}$ by f. Herein, $f^{-1}(y)$ returns the preimage $\{x | f(x) = y\}$ of y under f. Further, two functions f and g are called *h-equivalent* if and only if they have the same histogram. The corresponding h-equivalence class $B_h \subseteq \mathcal{F}$ containing all functions with histogram h is termed a *basis class*. It holds:

Lemma 1.1 ([18]). *Any subset $F \subseteq \mathcal{F}$ that is c.u.p. is uniquely defined by a union of pairwise disjoint basis classes. B_h is equal to the permutation orbit of any function f with histogram h, i.e., $\forall f \in F : B_{h_f} = \bigcup_{\pi \in \Pi(\mathcal{X})} \{f \circ \pi\}$.*

Example 1.6. Consider the functions in Table 1.1. The $\mathcal{Y}$-histogram of f_1 contains the value zero three times and the value one time, i.e., we have $h_{f_1}(0) = 3$ and $h_{f_1}(1) = 1$. The mappings f_1, f_2, f_4, f_8 have the same $\mathcal{Y}$-histogram and are therefore in the same basis class $B_{h_{f_1}} = \{f_1, f_2, f_4, f_8\}$. The set $\{f_1, f_2, f_4, f_8, f_{15}\}$ is c.u.p. and corresponds to $B_{h_{f_1}} \cup B_{h_{f_{15}}}$.

The following ramification of the Sharpened NFL theorem (derived independently in [19, 29], and [9] generalizing the results in [8]) gives a necessary and sufficient condition for a NFL result in the general case of arbitrary distributions $p_{\mathcal{F}}$ over $\mathcal{F}$:

Theorem 1.3 (Non-uniform Sharpened NFL Theorem [9,19,20,29]). *If and only if for all histograms h*

$$f, g \in B_h \Rightarrow p_{\mathcal{F}}(f) = p_{\mathcal{F}}(g) \ , \tag{1.6}$$

then for any two algorithms a and b, any value $k \in \mathbb{R}$, any $m \in \{1, \dots, |\mathcal{X}|\}$, and any performance measure c

$$\sum_{f\in\mathcal{F}} p_{\mathcal{F}}(f)\,\delta(k, c(Y(f,m,a))) = \sum_{f\in\mathcal{F}} p_{\mathcal{F}}(f)\,\delta(k, c(Y(f,m,b))) \ . \tag{1.7}$$

This observation is the "sharpest" NFL so far. It gives necessary and sufficient conditions for NFL results for general problem classes.

1.3 The Preconditions in the NFL Theorem and the Sharpened NFL Theorems

The NFL Theorems 1.1–1.3 consider a certain optimization scenario. In the following, we discuss its basic preconditions.

1.3.1 Independence of Algorithmic Complexity

It is important to note that the NFL theorems make no assumptions about the space and time complexity of computing the next search point. For example, it makes no difference if the algorithm simply enumerates all elements of $\mathcal{X}$ or comes up with a decision after running some complex internal simulation.

For many practical applications, it is indeed reasonable to assume that the time needed to evaluate the fitness dominates the computational costs of computing the next step and that memory requirements are not an issue. Still, this need not always be true.

Example 1.7. The update of the strategy parameters in CMA-ES scales quadratically with the search space dimension ([13, 30], see Chap. 8). For extremely high-dimensional fitness functions that can be evaluated quickly this may have a significant impact on the run time (however, in most practical applications I dealt with, the evaluation of the objective function was by far the dominating term).

This topic is discussed in more detail in Sect. 5.2.1, which demonstrates the differences between *algorithmic complexity* and *black-box complexity* of search algorithms.

1.3.2 Non-repeating Algorithms

Metaheuristics often do not have the non-repeating property. If a randomized algorithm searches locally in discrete (or discretized) domains, the chance of resampling a search point can be high even if the search space is huge. In evolutionary algorithms, the variation operators that are used to generate a new solution based on an existing one are often symmetric in the sense that the probability to generate some solution x' from x is equal to generating x from x'. Thus, there is always a chance to jump back to an already visited point.

We can always turn a repeating search algorithm into a non-repeating one by adding a look-up table for already visited points. The algorithm then internally uses this look-up table and only evaluates the objective function (i.e., "makes a step") for previously unseen points. Of course, this increases the memory requirements of the algorithm.

Example 1.8. Studies in which evolutionary algorithms searching a space of graphs are turned into non-repeating algorithms by coupling them with a search-point database are described in [16, 23].

1.3.3 Deterministic and Randomized Algorithms

In general, NFL results hold for deterministic as well as randomized algorithms. A randomized search algorithm a can be described by a probability distribution p_a over deterministic search behaviors [6]: Every search behavior generated by a single application of a randomized algorithm has a certain probability. The same behavior could have been generated by some deterministic algorithm. One can view the application of a randomized algorithm a as picking – according to a fixed, algorithm-dependent distribution p_a – a deterministic search behavior at random and applying it to the problem (we view the deterministic algorithms as a subset of the randomized algorithms having degenerated probability distributions). An alternative way to see this is to think of drawing all realizations of random variables required by a randomized search method at once prior to the search process and to use these events as inputs to a deterministic algorithm (see Chap. 5 for a detailed discussion of this issue).

Let the set A contain all deterministic search behaviors operating on $\mathcal{F}$. The performance of a randomized search algorithm a corresponds to the expectation over the possible search behaviors A w.r.t. p_a [22]:

$$\mathbb{E}\{c(Y(f,m,a))\} = \sum_{a' \in A} p_a(a') c(Y(f,m,a')) \tag{1.8}$$

For deterministic algorithms, the previous theorems not only state that the average performance of two algorithms is the same across all functions, but also *any* statistic – in particular the variance – of the performance values is the same. We can derive a similar result for randomized algorithms. Let us assume that the conditions of Theorem 1.3 are met. We consider two randomized algorithms described by p_a and p_b and extend the NFL theorems using simple transformations:

$$\begin{aligned} &\sum_{f \in \mathcal{F}} p_{\mathcal{F}}(f) \sum_{a' \in A} p_a(a') \delta(k, c(Y(f,m,a'))) \qquad (1.9) \\ &= \sum_{a' \in A} p_a(a') \underbrace{\sum_{f \in \mathcal{F}} p_{\mathcal{F}}(f)\, \delta(k, c(Y(f,m,a')))}_{\substack{\text{independent of } a' \\ \text{because of Theorem 1.3}}} \\ &\overset{\substack{\text{for any} \\ \text{algorithm } z}}{=} \sum_{f \in \mathcal{F}} p_{\mathcal{F}}(f)\, \delta(k, c(Y(f,m,z))) \sum_{a' \in A} p_a(a') \\ &= \sum_{f \in \mathcal{F}} p_{\mathcal{F}}(f)\, \delta(k, c(Y(f,m,z))) \sum_{b' \in A} p_b(b') \\ &\overset{\substack{\text{reversing the} \\ \text{prev. arguments}}}{=} \sum_{f \in \mathcal{F}} p_{\mathcal{F}}(f) \sum_{b' \in A} p_b(b') \delta(k, c(Y(f,m,b'))) \end{aligned}$$

Equation (1.2) holds for randomized algorithms if we simply replace $c(\cdot)$ by $\mathbb{E}\{c(\cdot)\}$.[1]

[1]In general, this cannot be done in the theorems because $\sum_{f \in F} \delta(k, \sum_{a' \in A} p_a(a') c(Y(f,m,a'))) \neq \sum_{f \in F} \sum_{a' \in A} p_a(a') \delta(k, c(Y(f,m,a')))$.

1.3.4 Finiteness of Domain and Codomain

If optimization problems solved on today's digital computers are considered, the restriction to finite domains and codomains is no restriction at all. Still, NFL results for continuous search spaces are very interesting from the theoretical point of view. The reader is referred to [26] and [1] for different views on this topic as well as to Chap. 7, which explains why assumptions on the Lebesgue measurability of $p_{\mathcal{F}}$ play a role when studying NFL in continuous search spaces.

1.3.5 Restriction to a Single Objective

The basic NFL theorems considers single-objective optimization. However, it also holds for multi-objective (vector) optimization as proven in [5].

1.3.6 Fixed Objective Functions

The framework considered in this chapter does not include repeated game scenarios in which the "objective function" for some agent can vary based on the behavior of other agents as is the case in coevolutionary processes. For results and discussions of NFL in such game-like scenarios the reader is referred to the work by Wolpert and Macready presented in [37].

1.3.7 Averaging Over All Search Behaviors and All Performance Criteria

The NFL theorems rely on averages over all possible search behaviors and all possible performance criteria. However, in practice we are most often concerned with the comparison of a subset of algorithms A using a fixed performance measure c and some fixed number of steps m. This scenario is considered in *Focused NFL theorems* [32]. Given a set of algorithms A and a performance measure c and some fixed number of steps m, Whitley and Rowe define a *focus set* as a set of functions on which the algorithms in A have the same mean performance measured by c (after m steps). The *orbit* of an objective function f is the smallest of these focus sets containing f. It is important to note that focus sets need not be c.u.p.

Example 1.9. Assume we want to compare two deterministic algorithms a and b run for m steps using performance measure c. Further, assume two objective functions f_1 and f_2 with $c(Y(f_1, m, a)) = c(Y(f_2, m, b))$ and $c(Y(f_2, m, a)) = c(Y(f_1, m, b))$. Then $\{f_1, f_2\}$ is clearly a focus set for $A = \{a, b\}$, c, and m – regardless of whether $\{f_1, f_2\}$ is c.u.p. or not.

For details about Focused NFL theorems the reader is referred to [32] and Chap. 4.

1.4 Restricted Function Classes and NFL

In this section we take a look at restricted function classes. First, we compute the probability that a randomly chosen function class meets the conditions of the NFL theorems. Then it is argued that there are certain restrictions that are likely to constrain problem classes corresponding to "real-world" problems. It is shown that these constraints are not compatible with the conditions of the NFL theorems. This is an encouraging result for the design of metaheuristics. However, it is difficult to evaluate heuristics empirically, because good performance on some functions does not necessarily imply good performance on others – even if these functions seem to be closely related. This is underlined by the Almost NFL theorem discussed at the end of this section.

1.4.1 How Likely Are the Conditions for NFL?

The question arises whether the preconditions of the NFL theorems are ever fulfilled in practice. How likely is it that a randomly chosen subset is c.u.p.? There exist $2^{\left(|\mathcal{Y}|^{|\mathcal{X}|}\right)} - 1$ non-empty subsets of $\mathcal{F}$, and it holds:

Theorem 1.4 ([18]). *The number of non-empty subsets of $\mathcal{Y}^{\mathcal{X}}$ that are c.u.p. is given by*

$$2^{\binom{|\mathcal{X}|+|\mathcal{Y}|-1}{|\mathcal{X}|}} - 1 \tag{1.10}$$

and therefore the fraction of non-empty subsets c.u.p. is given by

$$\left(2^{\binom{|\mathcal{X}|+|\mathcal{Y}|-1}{|\mathcal{X}|}} - 1\right) \Big/ \left(2^{\left(|\mathcal{Y}|^{|\mathcal{X}|}\right)} - 1\right) \ . \tag{1.11}$$

Figure 1.3 shows a plot of the fraction of non-empty subsets c.u.p. versus the cardinality of $\mathcal{X}$ for different values of $|\mathcal{Y}|$. The fraction decreases for increasing $|\mathcal{X}|$ as well as for increasing $|\mathcal{Y}|$. More precisely, using bounds for binomial coefficients one can show that Eq. (1.11) converges to zero doubly exponentially fast with increasing $|\mathcal{X}|$ (for $|\mathcal{Y}| > e|\mathcal{X}|/(|\mathcal{X}| - e)$, where e is Euler's number). For small $|\mathcal{X}|$ and $|\mathcal{Y}|$ the fraction almost vanishes.

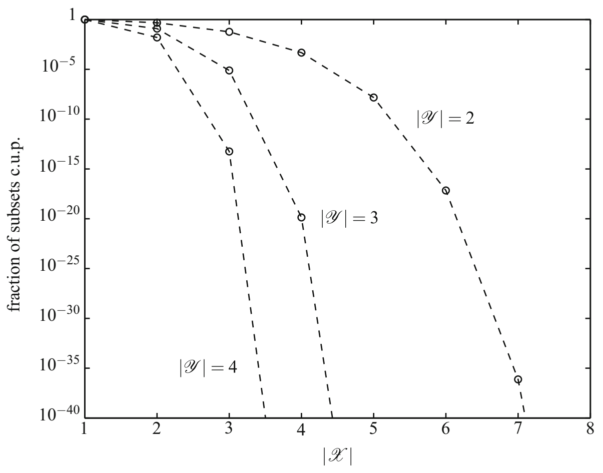


Fig. 1.3 Fraction of subsets of objective functions that are closed under permutation (c.u.p.) of all possible subsets depending on the number of search points and the number of possible objective function values. The ordinate gives the fraction of subsets c.u.p. on logarithmic scale given the cardinality $|\mathcal{X}|$ of the search space. The different curves correspond to different cardinalities of the space of objective function values $\mathcal{Y}$

Thus, the statement "I'm only interested in a subset F of all possible functions and the precondition of the Sharpened NFL theorem is not fulfilled" is true with a probability close to one if F is chosen uniformly at random and $\mathcal{Y}$ and $\mathcal{X}$ have reasonable cardinalities.

The probability that a randomly chosen distribution over the set of objective functions fulfills the preconditions of Theorem 1.3 has measure zero. This means that in this general and realistic scenario the conditions for a NFL result almost surely do not hold.

1.4.2 Structured Search Spaces and NFL

Although the fraction of subsets c.u.p. is close to zero for small search and cost-value spaces, the absolute number of subsets c.u.p. grows rapidly with increasing $|\mathcal{X}|$ and $|\mathcal{Y}|$. What if these classes of functions are the relevant ones in the sense that they correspond to the problems we are dealing with in practice?

It can be argued that presumptions can be made for most of the functions relevant in real-world optimization: First, the search space has some structure. Second, the set of objective functions fulfills some constraints defined based on this structure. More formally, there exists a non-trivial neighborhood relation on $\mathcal{X}$ based on which constraints on the set of functions under consideration are formulated. Such constraints include upper bounds on the ruggedness or on the maximum number of local minima. Concepts such as ruggedness or local optimality require a notion of neighborhood.

A neighborhood relation on $\mathcal{X}$ is a symmetric function $n : \mathcal{X} \times \mathcal{X} \to \{0, 1\}$. Two elements $x_i, x_j \in \mathcal{X}$ are called neighbors if $n(x_i, x_j) = 1$. A neighborhood relation is called non-trivial if $\exists x_i, x_j \in \mathcal{X} : x_i \neq x_j \wedge n(x_i, x_j) = 1$ and $\exists x_k, x_l \in \mathcal{X} : x_k \neq x_l \wedge n(x_k, x_l) = 0$. There are only two trivial neighborhood relations, either every two points are neighbored or no points are neighbored. Non-trivial neighborhood relations are not preserved under permutations, it holds:

Theorem 1.5 ([18]). *A non-trivial neighborhood relation on $\mathcal{X}$ is not invariant under permutations of $\mathcal{X}$, i.e.,*

$$\exists x_i, x_j \in \mathcal{X}, \pi \in \Pi(\mathcal{X}) : n(x_i, x_j) \neq n(\pi(x_i), \pi(x_j)) \ . \tag{1.12}$$

This result is quite general. Assume that the search space $\mathcal{X}$ can be decomposed as $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_l$, $l > 1$, and let a non-trivial neighborhood $n_i : \mathcal{X}_i \times \mathcal{X}_i \to \{0, 1\}$ exist on one component $\mathcal{X}_i$. This neighborhood induces a non-trivial neighborhood on $\mathcal{X}$, where two points are neighbored if their ith components are neighbored with respect to n_i regardless of the other components. Thus, the constraints discussed in the examples below need only refer to a single component. Note that the neighborhood relation need not be the canonical one (e.g., the Hamming-distance for Boolean search spaces). For example, if integers are encoded by bit-strings, then the bit-strings can be defined as neighbored iff the corresponding integers are. The following examples illustrate further implications of Theorem 1.5 [18].

Example 1.10. Consider a non-empty subset $F \subset \mathcal{F}$ where the co-domains of the functions have more than one element and a non-trivial neighborhood relation exists. If for each $f \in F$ it is not allowed that a global maximum is neighbored to a global minimum (i.e., we have a constraint "steepness"), then F is not c.u.p. (because for every $f \in F$ there exists a permutation that maps a global minimum and a global maximum of f to neighboring points). An example of such a function class is OneMax* as described in Sect. 5.5.

Example 1.11. Imposing an upper bound on the complexity of possible objective functions can lead to function classes that are not c.u.p. Let us assume without loss of generality minimization tasks. Then the number of suboptimal local minima is often regarded as a measure of complexity of an objective function [31] (see Sect. 1.4.3 for other notions of complexity). Given a neighborhood relation on $\mathcal{X}$, a local minimum can be defined as a point whose neighbors all have worse fitness. As a concrete example, consider all mappings $\{0, 1\}^\ell \to \{0, 1\}$ not having maximum complexity in the sense that they have less than the maximum number of 2^{n-1} local minima w.r.t. the ordinary hypercube topology on $\{0, 1\}^\ell$. For example, this set does not contain mappings such as the parity function, which is one iff the number of ones in the input bit-string is even. This set is not c.u.p. The general statement that imposing an upper bound on the number of local minima leads to function classes not c.u.p. can be formally proven using the following line of arguments. Let $l(f)$

be the number of local minima of a function $f \in \mathcal{F}$, and let $B_{h_f} \subseteq \mathcal{F}$ be the set of functions in $\mathcal{F}$ with the same $\mathcal{Y}$-histogram as f (i.e., functions where the number of points in $\mathcal{X}$ that are mapped to each $\mathcal{Y}$-value is the same as for f). Given a function f we define $l^{\max}(f) = \max_{f' \in B_{h_f}} l(f')$ as the maximal number of local minima that functions in $\mathcal{F}$ with the same $\mathcal{Y}$-histogram as f can possibly have. For a non-empty subset $F \subset \mathcal{F}$ we define $l^{\max}(F) = \max_{f \in F} l^{\max}(f)$. Let $g(F) \in \mathcal{F}$ be a function with $l(g(F)) = l^{\max}(F)$ local minima and the same $\mathcal{Y}$-histogram as a function in F. Now, if the number of local minima of every function $f \in F$ is constrained to be smaller than $l^{\max}(F)$ (i.e., $\max_{f \in F} l(f) < l^{\max}(F)$), then F is not c.u.p. – because $\exists f \in F$ with the same $\mathcal{Y}$-histogram as g and thus $\exists \pi \in \Pi(\mathcal{X}) : f \circ \pi = g$.

1.4.3 The Almost NFL Theorem

The results presented above are encouraging, because they suggest that for a restricted problem class the NFL results will most likely not apply. However, this does not promise a "free lunch". For a problem class violating the conditions for a NFL result, the average performance of two algorithms may differ. But this difference may be negligible.

The performance on two functions, although they are similar according to some reasonable similarity measure, may differ significantly. In particular, for any function f on which algorithm a performs well, there may be many functions on which it performs badly and that are similar to f in a reasonable sense. This becomes clear from the Almost NFL (ANFL) theorem derived by Droste et al.[6], which proves such statements for a particular scenario:

Theorem 1.6 (Almost NFL Theorem [6]). *Let $\mathcal{F}$ be the set of objective functions $f : \{0,1\}^n \to \{0,1,\ldots,N-1\}$ for fixed positive integers n and N. For any algorithm a operating on $\mathcal{F}$ and any function $f \in \mathcal{F}$ there exist at least $N^{2^{n/3}-1}$ functions in $\mathcal{F}$ that agree with f on all but at most $2^{n/3}$ inputs for which a does not find their optimum within $2^{n/3}$ steps with probability bounded above by $2^{-n/3}$.*

Exponentially many of these functions have the additional property that their evaluation time, circuit size representation, and Kolmogorov complexity is only by an additive term of $O(n)$ larger than the corresponding complexity of f.

The last sentence formalizes that extremely poor behavior can be expected on functions which are similar to our reference (e.g., benchmark) function f. These functions do not only coincide on $2^{n/3}$ inputs with f; they also have a similar complexity. Complexity can either be measured in the number of steps needed to evaluate the objective function at a certain point (evaluation time), the length of the shortest program implementing f (Kolmogorov complexity), or by the size of a circuit representation of f (circuit size representation).

1.5 Search and Markov Decision Processes

If a problem class does not fulfill the assumptions for NFL, then there may be differences in performance between algorithms on this class. The question arises: What is then the best method given a problem class $p_{\mathcal{F}}$, a performance measure, and a number of steps m? In the following, it is shown how to determine an optimal algorithm using dynamic programming (DP) [15]. For a similar approach see [1] and Chap. 6. However, it is not claimed that this way is efficient.

The problem of finding an optimal search algorithm can be transformed to a *finite horizon optimal control problem* that can be solved by DP. In the context of mathematical optimization for optimal control, DP is concerned with some discrete-time dynamic system, in which at time t the state s_t of the system changes according to given transition probabilities that depend on some action a_{t+1} (I adopt a standard formalism and use a for actions instead of algorithms in this section). Each transition results in some immediate reward r_{t+1} (or, alternatively, immediate cost) [3]. The dynamic system can be described by a finite Markov decision process:

Definition 1.2 (Finite MDP). A finite Markov decision process $[\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}]$ (finite MDP) is given by:

1. A finite set $\mathcal{S}$ of states;
2. A finite set of actions $\mathcal{A}$, where $\mathcal{A}(s)$ denotes the set of possible actions in state $s \in \mathcal{S}$;
3. Transition probabilities $\mathcal{P}^a_{ss'} = \Pr\{s_{t+1} = s' \mid s_t = s, a_t = a\}$ describing how likely it is to go from $s \in \mathcal{S}$ to $s' \in \mathcal{S}$ when taking action $a \in \mathcal{A}$, and
4. Expected reward values $\mathcal{R}^a_{ss'} = \mathrm{E}\{r_{t+1} \mid s_{t+1} = s', s_t = s, a_t = a\}$ expressing the expected reward when going from $s \in \mathcal{S}$ to $s' \in \mathcal{S}$ after taking action $a \in \mathcal{A}$.

The goal of a *finite horizon problem*[2] is to find a policy $\varpi : \mathcal{S} \to \mathcal{A}$ that maximizes the accumulated m-step reward

$$R_m = \sum_{t'=1}^{m} r_{t'} \ . \tag{1.13}$$

Given $[\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}]$ the optimal policy can be computed using established standard techniques [2, 3].

Now we turn an optimization problem (as illustrated Fig. 1.2) into a finite MDP. First, we need to fix some additional notation. The length of a sequence $T_m = \langle (x_1, f(x_1)), \dots, (x_m, f(x_m)) \rangle$ is given by $\mathrm{length}(T_m) = m$. Adding an element to a sequence is indicated by $\langle x_1, x_2, \dots, x_n \rangle \oplus x' = \langle x_1, x_2, \dots, x_n, x' \rangle$. Given

[2]It is also possible to map the optimization to an *infinite horizon problem* with appropriate *absorbing states*.

T_M, we denote the corresponding sequence and set of search points by $X(T_m) = \langle x_1, x_2, \dots, x_m \rangle$ and $\mathcal{X}(T_m) = \{x_1, x_2, \dots, x_m\}$, respectively, and the sequence of cost values by $Y(T_m) = \langle f(x_1), f(x_2), \dots, f(x_m) \rangle$.

We define the set $F(T_m)$ of all functions that are *compatible with* T_m *in* $F \in \mathcal{F}$ by

$$F(T_m) = \{g \mid g \in F \wedge \forall x \in \mathcal{X}(T_m) : g(x) = f(x)\} \tag{1.14}$$

The *search MDP* (S-MDP) given an optimization scenario can now be formalized as follows:

Definition 1.3 (S-MDP). Given a problems class $p_{\mathcal{F}}$, a performance measure c, and a number of steps $m \in \{1, \dots, |\mathcal{X}|\}$, the search MDP (S-MDP) is defined by

1. $\mathcal{S}$ = set of all possible traces T_n with $0 \leq n \leq m$;
2. $\mathcal{A}(s) = \mathcal{X} \setminus \mathcal{X}(s)$;
3. $\mathcal{P}^a_{ss'} = \begin{cases} 0 & X(s') \neq X(s) \oplus a \\ \sum\limits_{g \in F(s')} p_{\mathcal{F}}(g) \Big/ \sum\limits_{h \in F(s)} p_{\mathcal{F}}(h) & \text{otherwise} \end{cases}$;
4. $\mathcal{R}^a_{ss'} = \begin{cases} 0 & \text{if length}(s) \neq m-1 \\ c(Y(s')) & \text{otherwise} \end{cases}$.

This MDP has been constructed such that the following theorem holds, which establishes the link between optimal search algorithm and optimal policy:

Theorem 1.7. *For any performance measure c, any problem class $p_{\mathcal{F}}$ over $\mathcal{F}$, and any $m \in \{1, \dots, |\mathcal{X}|\}$, an algorithm b that maximizes*

$$\sum_{f \in F} p_{\mathcal{F}}(f)\, c(Y(f, m, b))$$

is given by the optimal policy for the corresponding S-MDP (with $t = 0$ and $s_0 = \langle\rangle$).

Proof. By definition, every policy operating on a S-MDP is an algorithm. The states of the dynamic system correspond to the sequences of previously evaluated search points and an action corresponds to exploring a new search point.

First, we verify that the definition of $\mathcal{P}^a_{ss'}$ leads to proper probability distributions, that is, that $\forall s \in \mathcal{S} : \forall a \in \mathcal{A}(s) : \sum_{s' \in \mathcal{S}} \mathcal{P}^a_{ss'} = 1$. We rewrite the definition of $\mathcal{P}^a_{ss'}$ as

$$\mathcal{P}^a_{ss'} = \delta(X(s'), X(s) \oplus a) \sum_{g \in F(s')} p_{\mathcal{F}}(g) \Big/ \sum_{h \in F(s)} p_{\mathcal{F}}(h) \ . \tag{1.15}$$

For every $f \in F(s)$ and $a \in \mathcal{A}(s)$ there is exactly one $s' \in \mathcal{S}$ with $X(s') = X(s) \oplus a$ and $f \in F(s')$. Thus we have

$$\sum_{h \in F(s)} p_{\mathcal{F}}(h) = \sum_{\substack{s' \in \mathcal{S} \\ \wedge X(s') = X(s) \oplus a}} \sum_{g \in F(s')} p_{\mathcal{F}}(g) = \sum_{s' \in \mathcal{S}} \delta(X(s'), X(s) \oplus a) \sum_{g \in F(s')} p_{\mathcal{F}}(g) \tag{1.16}$$

and the normalization is correct.

Next, we show that maximizing the accumulated reward $\mathbb{E}\{R_0 \,|\, s_0 = \langle\rangle\,, \varpi\}$ maximizes the performance measure $\sum_{f \in F} p_{\mathcal{F}}(f)\, c(Y(f, m, \varpi))$. The accumulated immediate reward reduces to

$$R_0 = \sum_{t'=1}^{m} r_{t'} = r_m \quad . \tag{1.17}$$

Given a function $f \in \mathcal{F}$ and a policy ϖ we have $r_m = c(Y(f, m, \varpi))$ and therefore

$$\mathbb{E}\{R_0 \,|\, s_0 = \langle\rangle\,, f, \varpi\} = \mathbb{E}\{r_m \,|\, s_0 = \langle\rangle\,, f, \varpi\} = c(Y(f, m, \varpi)) \tag{1.18}$$

as the process is deterministic for fixed f. Thus

$$\mathbb{E}\{R_0 \,|\, s_0 = \langle\rangle\,, \varpi\} = \sum_{f \in \mathcal{F}} p_{\mathcal{F}}(f)\, c(Y(f, m, \varpi)) \quad , \tag{1.19}$$

which completes the proof. □

Solving the S-MDP leads to an optimal algorithm, which "plans" the next search points in order to maximize the reward (e.g., to find the best solution in m steps). Although solving a MDP can be done rather efficiently[3] in terms of scaling with $|\mathcal{S}|$, $|\mathcal{A}|$, and m, *solving the S-MDP is usually intractable* because $|\mathcal{S}|$ obviously scales badly with the dimensionality of the optimization problem. In Chap. 6, Teytaud and Vazquez discuss this issue and optimal search algorithms in general in more detail.

1.6 What Can We Learn from the NFL Results for the Design of Metaheuristics?

There is no universal best search or optimization algorithm. A "careful consideration of the NFL theorems forces us to ask what set of problems we want to solve and how to solve them" [33]. The NFL theorem "highlights the need for exploiting

[3]In a S-MDP no state is ever revisited. Hence, for any policy, the transition probability graph is acyclic and thus *value iteration* finds the optimal policy after at most m steps, where each step needs $O(|\mathcal{A}||\mathcal{S}|)^2$ computations (see [3, Sect. 2.2.2] or [2]).

problem-specific knowledge to achieve better than random performance" [37]. Its essence is nicely captured in the discussion of the NFL results for learning [34] by Bousquet et al. [4]. If "there is no a priori restriction on the possible phenomena that are expected [...] there is no better algorithm (any algorithm would be beaten by another one on some phenomena)" [4]. If their words are adapted for the search and optimization scenario, the main message of the NFL theorems may be summarized as follows:

> If there is no restriction on how the past (already visited points) can be related to the future (not yet explored search points), efficient search and optimization are impossible.

Between the two extremes of knowing nothing about a problem domain and knowing a domain so well that an efficient, highly specialized algorithm can be derived in reasonable time, there is enough room for the application of well designed metaheuristics. Often we have only little, quite general knowledge about the properties of a problem class. In such a case, using quite general metaheuristics exploiting these properties may be a good choice. In practice, sometimes fine tuning an algorithm to a problem class is not efficient in terms of development time and costs. Then a more broadly tuned metaheuristic may be preferable.

Finally, let me mention some further conclusions that can be drawn from the results reviewed in this chapter.

1.6.1 The Preconditions of the NFL Theorem Are Not Met in Practice

I argue that if we consider a restricted class of problems, it is reasonable to assume that the necessary conditions in the NFL theorems are not fulfilled. First, if we would select a problem class at random, the probability that the conditions hold is extremely low. Second, if the objective functions in a problem class obey some constraints that are defined with respect to some neighborhood relation on the search space, then the necessary prerequisites are likely to be violated. Thus, we can hope to come up with metaheuristics showing above average performance for real-world problem classes.

However, one has to keep in mind that the strict NFL results refer to averages over all possible search behaviors and performance criteria. If we compare a selection of algorithms using a fixed criterion, there can exist sets of functions on which the algorithms have the same mean performance even if the sets do not meet the conditions of the general NFL theorems. This is investigated in the context of Focused NFL theorems, see [32] and Chap. 4.

1.6.2 Generalization from Benchmark Problems Is Dangerous

The Almost NFL theorem and the second statement of Corollary 1.1 show that drawing general conclusions about the ranking of algorithms – even if referring

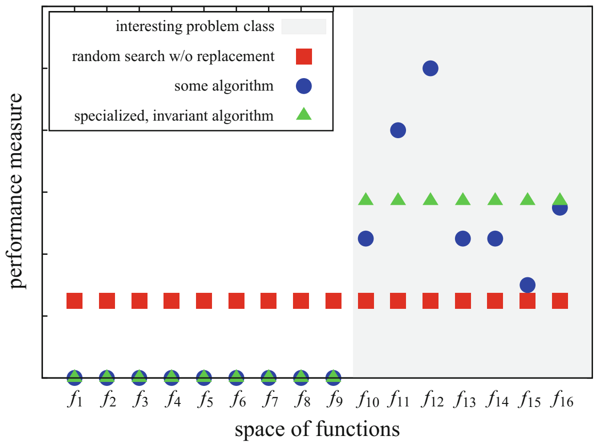


Fig. 1.4 Distribution of mean performance over a discrete space of problems for some randomized algorithms. The algorithms indicated by *triangles* and *circles* both have a similar mean performance on the interesting problems (i.e., some problem class we are developing a metaheuristic for), but a different variance. Both are biased towards the interesting problems. Uniform random search and the algorithm indicated by *triangles* are invariant under the choice of the problem instance from the class of interesting problems

to a restricted problem class – based on evaluating them on single benchmark functions is dangerous (see the discussion in [33]). For every set of functions on which algorithm a outperforms algorithm b there is a set of function on which the opposite is true. Thus, one must ensure that the results on the considered benchmark functions can be generalized to the whole class of problems the algorithms are designed for. If algorithm b outperforms a on some test functions that are not likely to be observed in practice (in my opinion this includes functions frequently used in benchmark suites, e.g., some "deceptive" problems), this can even be regarded as an argument in favor of a.

There are ways to derive valid statements about algorithm performance on a problem class without testing the algorithm on every instance of the class. First, one can derive formal proofs about the performance of the algorithms. Of course, this can be extremely difficult. Second, it is possible to generalize from empirical results on single benchmark functions to a class of problems in a strict sense if the algorithms have certain invariance properties, see [11, 12] and Chap. 8 for a discussion.

Example 1.12. Evolutionary algorithms in which the selection is based on ranking, such as the CMA-ES [13] presented in Chap. 8, are invariant under order-preserving transformations of the fitness function. For example, if the fitness values given by a function $f(x)$ are all positive, the performance of a purely rank-based algorithm on f generalizes to $f(x)^2, \log(f(x)), \ldots$. The CMA-ES has several additional invariance properties, in particular it is invariant under rotation of the search space.

As there is no well-performing universal search algorithm, the metaheuristics we are developing must be biased towards certain problem classes. There is some trade-off between the specialization to a problem class required for efficient optimization and invariance properties. This becomes obvious by the invariance properties of uniform random-search, which is fully unbiased. Figure 1.4 illustrates specialization and invariance properties, especially showing the performance of an algorithm that is biased toward an interesting problem class and at the same time invariant under the choice of the problem instance from this class. Understanding, formulating, and achieving the right bias and the right invariance properties is perhaps the most important aspect when designing metaheuristics.

1.7 Further Reading

The book chapter on "Complexity Theory and the No Free Lunch Theorem" by Whitley and Watson is recommended for an alternative review of NFL for search and optimization [33]. The proof of the basic NFL theorem can found in [36]; the proof of the Sharpened NFL theorem in [27]. Theorem 1.3 is proven in [20] and the results in Sects. 1.4.1 and 1.4.2 in [18]. The Almost NFL theorem is derived in [6], and Focused NFL is discussed in [32].

Acknowledgements Parts of this text are based on former joint work with Marc Toussaint. The author acknowledges support from the German Federal Ministry of Education and Research within the National Network Computational Neuroscience – Bernstein Fokus: "Learning behavioral models: From human experiment to technical assistance", grant FKZ 01GQ0951.

References

1. A. Auger, O. Teytaud, Continuous lunches are free plus the design of optimal optimization algorithms. Algorithmica **57**(1), 121–146 (2010)
2. D.P. Bertsekas, Dynamic programming and optimal control. Athena Sci. **2** (2007)
3. D.P. Bertsekas, J.N. Tsitsiklis, Neuro-dynamic programming. Athena Sci. (1996)
4. O. Bousquet, S. Boucheron, G. Lugosi, Introduction to statistical learning theory, in *Advanced Lectures in Machine Learning*, ed. by O. Bousquet, U. von Luxburg, G. Rätsch. LNAI, vol. 3176 (Springer, Berlin Heidelberg, 2004), pp. 169–207
5. D.W. Corne, J.D. Knowles, No free lunch and free leftovers theorems for multiobjective optimisation problems, in *Evolutionary Multi-Criterion Optimization (EMO 2003)*, ed. by C.M. Fonseca, P.J. Fleming, E. Zitzler, L. Thiele, K. Deb. LNCS, vol. 2632 (Springer, Berlin Heidelberg, 2003), pp. 327–341
6. S. Droste, T. Jansen, I. Wegener, Optimization with randomized search heuristics – the (A)NFL theorem, realistic scenarios, and difficult functions. Theor. Comput. Sci. **287**(1), 131–144 (2002)
7. T.M. English, Evaluation of evolutionary and genetic optimizers: no free lunch, in *Proceedings of the Fifth Annual Conference on Evolutionary Programming (EP V)*, San Diego, ed. by L.J. Fogel, P.J. Angeline, T. Bäck (MIT, 1996), pp. 163–169

8. T.M. English, Optimization is easy and learning is hard in the typical function, in *Proceedings of the 2000 Congress on Evolutionary Computation (CEC 2000)*, San Diego, ed. by A. Zalzala, C. Fonseca, J.H. Kim, A. Smith (IEEE, 2000), pp. 924–931
9. T. English, On the structure of sequential search: beyond "No free lunch", in *Evolutionary Computation in Combinatorial Optimization (EvoCOP 2004)*, ed. by J. Gottlieb, G.R. Raidl. LNCS, vol. 3004 (Springer, Berlin Heidelberg, 2004), pp. 95–103
10. C. Giraud-Carrier, F. Provost, Toward a justification of meta-learning: is the no free lunch theorem a show-stopper? in *Proceedings of the ICML-2005 Workshop on Meta-learning*, Bonn, 2005
11. N. Hansen: Invariance, self-adaptation and correlated mutations and evolution strategies, in *Parallel Problem Solving from Nature (PPSN VI)*, ed. by M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J.J.M. Guervós, H.P. Schwefel. LNCS, vol. 1917 (Springer, Berlin Heidelberg, 2000), pp. 355–364
12. N. Hansen, Adaptive encoding: how to render search coordinate system invariant, in *Parallel Problem Solving from Nature (PPSN X)*, ed. by G. Rudolph, T. Jansen, S.M. Lucas, C. Poloni, N. Beume. LNCS, vol. 5199 (Springer, Berlin Heidelberg, 2008), pp. 205–214
13. N. Hansen, A. Ostermeier, Completely derandomized self-adaptation in evolution strategies. Evol. Comput. **9**(2), 159–195 (2001)
14. D. Hume, *A Treatise of Human Nature*, Chap. Sect. vi. Of the Inference from the impression to the idea. Web edition published by eBooks@Adelaide, 2006 (1739)
15. C. Igel, Recent results on no-free-lunch for optimization, in *Theory of Evolutionary Algorithms*, no. 04081, ed. by H. Beyer, T. Jansen, C. Reeves, M.D. Vose, in *Dagstuhl Seminar Proceedings, Abstract Collection*, Schloss Dagstuhl (Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), 2004)
16. C. Igel, P. Stagge, Graph isomorphisms effect structure optimization of neural networks, in *International Joint Conference on Neural Networks (IJCNN 2002)*, Honolulu (IEEE, 2002), pp. 142–147
17. C. Igel, M. Toussaint, Neutrality and self-adaptation. Nat. Comput. **2**(2), 117–132 (2003)
18. C. Igel, M. Toussaint, On classes of functions for which no free lunch results hold. Inf. Process. Lett. **86**(6), 317–321 (2003)
19. C. Igel, M. Toussaint, Recent results on no-free-lunch theorems for optimization. arXiv preprint cs.NE/0303032 (2003), http://arxiv.org/abs/cs.NE/0303032
20. C. Igel, M. Toussaint, A no-free-lunch theorem for non-uniform distributions of target functions. J. Math. Model. Algorithms **3**(4), 313–322 (2004)
21. M. Köppen, D.H. Wolpert, W.G. Macready, Remarks on a recent paper on the "no free lunch" theorems. IEEE Trans. Evol. Comput. **5**(3), 295–296 (1995)
22. R. Motwani, P. Raghavan, *Randomized Algorithms* (Cambridge University Press, Cambridge, 1995)
23. J. Niehaus, C. Igel, W. Banzhaf, Reducing the number of fitness evaluations in graph genetic programming using a canonical graph indexed database. Evol. Comput. **15**(2), 199–221 (2007)
24. N.J. Radcliffe, P.D. Surry, Fundamental limitations on search algorithms: evolutionary computing in perspective, in *Computer Science Today: Recent Trends and Development*, ed. by J. van Leeuwen. LNCS, vol. 1000 (Springer, Berlin Heidelberg, 1995), pp. 275–291
25. G.J.E. Rawlins, Introduction, in *Foundations of Genetic Algorithms (FOGA)*, ed. by G.J.E. Rawlins (Morgan Kaufmann, San Mateo, 1991), pp. 1–10
26. J.E. Rowe, M.D. Vose, A.H. Wright, Reinterpreting no free lunch. Evol. Comput. **17**(1), 117–129 (2009)
27. C. Schumacher, Fundamental limitations of search. Ph.D. Thesis, University of Tennessee, 2000
28. C. Schumacher, M.D. Vose, L.D. Whitley, The no free lunch and description length, in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)*, San Francisco, ed. by L. Spector, E. Goodman, A. Wu, W. Langdon, H.M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. Garzon, E. Burke (Morgan Kaufmann, 2001), pp. 565–570

29. M.J. Streeter, Two broad classes of functions for which a no free lunch result does not hold, in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2003)*, Chicago, ed. by E. Cantú-Paz, J.A. Foster, K. Deb, D. Davis, R. Roy, U.M. O'Reilly, H.G. Beyer, R. Standish, G. Kendall, S. Wilson, M. Harman, J. Wegener, D. Dasgupta, M.A. Potter, A.C. Schultz, K. Dowsland, N. Jonoska, J. Miller. LNCS, vol. 2724 (Springer, Berlin Heidelberg, 2003), pp. 1418–1430
30. T. Suttorp, N. Hansen, C. Igel, Efficient covariance matrix update for variable metric evolution strategies. Mach. Learn. **75**(2), 167–197 (2009). doi:10.1007/s10994-009-5102-1
31. D. Whitley, A free lunch proof for Gray versus binary encodings, in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999)*, Orlando, ed. by W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela, R.E. Smith, vol. 1 (Morgan Kaufmann, 1999), pp. 726–733
32. D. Whitley, J. Rowe, Focused no free lunch theorems, in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2008)*, Atlanta, ed. by M. Keijzer, et al. (ACM, 2008), pp. 811–818
33. D. Whitley, J.P. Watson, Complexity theory and the no free lunch theorem, in *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, ed. by E.K. Burke, G. Kendall, Chap. 11 (Springer, New York, 2005), pp. 317–339
34. D.H. Wolpert, The lack of a priori distinctions between learning algorithms. Neural Comput. **8**(7), 1341–1390 (1996)
35. D.H. Wolpert, W.G. Macready, No free lunch theorems for search. Technical Report SFI-TR-05-010, Santa Fe Institute, Santa Fe (1995)
36. D.H. Wolpert, W.G. Macready, No free lunch theorems for optimization. IEEE Trans. Evol. Comput. **1**(1), 67–82 (1997)
37. D.H. Wolpert, W.G. Macready, Coevolutionary free lunches. IEEE Trans. Evol. Comput. **9**(6), 721–735 (2005)

Chapter 2
Convergence Rates of Evolutionary Algorithms and Parallel Evolutionary Algorithms

Fabien Teytaud and Olivier Teytaud

Abstract This chapter discusses the advantages (robustness) and drawbacks (slowness) of algorithms searching the optimum by comparisons between fitness values only. The results are mathematical proofs, but practical implications in terms of speed-up for algorithms applied on parallel machines are presented, as well as practical hints for tuning parallel optimization algorithms and on the feasibility of some specific forms of optimization. Throughout the chapter, $[[a,b]] = \{a, a+1, \ldots, b\}$.

2.1 Introduction: Comparison-Based Algorithms and Their Robustness

There are several important families of optimization algorithms in the literature: Newton-like algorithms, using the Hessian (i.e., the second-order derivative); gradient descent, using the gradient (i.e., the first-order derivative); and algorithms using the objective function values (also termed the fitness values) only. There are particular cases to be emphasized.

First, Quasi-Newton methods (in particular BFGS [6, 8, 10, 18]) are, formally, in the family of algorithms using the gradient: They build, internally, an approximation of the Hessian, but they never request the Hessian.

Second, some algorithms use less than the fitness values: These algorithms are *comparison-based* algorithms. They include direct search methods [7] and most evolutionary algorithms [4, 15]. This choice of using only a small part of the available information is justified by the following:

F. Teytaud (✉) · O. Teytaud
TAO, Inria Saclay IDF, LRI, University Paris-Sud, UMR CNRS 8623, Paris, France
e-mail: fteytaud@lri.fr; Olivier.Teytaud@lri.fr

- Sometimes it is just the only available information. For example, when optimizing a strategy for two-player game, typically, an iteration of the evolution strategy performs a tournament between the individuals. This tournament runs as long as confidence intervals are not significant (or it reaches a time limit and concludes to a draw), and this only provides a ranking (possibly with ties).
- Whenever more information is available, the robustness is better with comparisons only. This was explained in [2, 3, 27] and was formally established in [9]. Essentially, Hansen and Ostermeier [9] show that when considering the worst case among compositions of a given family of fitness functions with increasing mappings (i.e., when considering that the fitness function f might be replaced by $g \circ f$ for some increasing $g : \mathbb{R} \to \mathbb{R}$), then optimality is necessarily reached by a comparison-based algorithm. Even methods based on surrogate models are sometimes now comparison-based for improved robustness [13]. In spite of the loss of information, using comparisons only has the advantage that it makes the optimization run resistant against some forms of noise.

Algorithms using comparisons are now widely established, and this chapter is devoted essentially to these algorithms (however, the tools used in the proofs can be applied in other cases). In particular, we will consider the following families of algorithms (more formally presented in [21]):

- Selection-based non-elitist (μ, λ) evolution strategies (SB-(μ, λ)-ES). These algorithms, at each generation, generate λ points (also termed individuals) in the search space, and the fitness function must inform the algorithm of which μ of these λ points are the μ best individuals (we must resolve ties here). These μ points are termed the *selected set*.
- Selection-based elitist $(\mu + \lambda)$ evolution strategies (SB-$(\mu + \lambda)$-ES). These algorithms, at each generation, generate λ points (also termed individuals) in the domain of the optimization, and the fitness function must inform the algorithm which μ of the union of (i) these λ points and (ii) the μ points selected at the previous generation, are the μ best individuals (we must resolve ties here).
- Full ranking versions of the algorithms above, i.e., FR-(μ, λ)-ES and FR-$(\mu+\lambda)$-ES; in these cases, the optimization is informed of which μ points are the best, and also of the complete ranking of these μ points. Please note that the definition "full ranking" is not often used in the literature; nonetheless, it is necessary here as many well-known algorithm benefit from a full ranking.

For given fixed values of λ and μ, full ranking algorithms use more information than selection-based elitist algorithms, which in turn use more information than selection-based non-elitist strategies; as a consequence, they are expected to be faster in terms of local convergence. We will see to which extent they can be faster, and we will in particular consider the behavior as $\lambda \to \infty$. This is particularly relevant now with the arrival of multi-core machines, clusters, and grids/clouds.

It is intuitively quite natural that comparison-based algorithms are slower, as they have less information guiding the search. This is the price to pay for increased robustness, shown in [9], and is well-known and widely asserted in evolutionary algorithms. In this chapter, we discuss the advantages (robustness) and drawbacks (slowness) of comparison-based algorithms. In particular, we will:

- Introduce, in Sect. 2.2, the important notion of branching factor [22]. This notion is central to the understanding of comparison-based algorithms, in parallelization, and it is also important beyond the scope of this chapter;
- Show the complexity bounds derived from this notion (Sect. 2.3);
- Show the computational cost associated with some real-world algorithms (Sect. 2.4), which is often significantly different from the theoretical optimum;
- Give the implications of these complexity bounds (Sect. 2.5).

Throughout the chapter $\mathbb{E}$ denotes the expectation operator. We use the abbreviation $(1), (2), \ldots, (\lambda)$ to denote a reordering of fitness values, i.e., $(1) = i_1, (2) = i_2, (3) = i_3, \ldots, (\lambda) = i_\lambda$ with $y_{i_1} \leq y_{i_2} \leq \cdots \leq y_{i_\lambda}$.

2.2 The Branching Factor

Let us first present the branching factor in a simple example: The $(1+1)$-ES. If there is no equality, then there are two cases: The newly sampled point is better than the current iteration, or it is worse. For defining a $(1+1)$-ES, one should specify how the internal state of the algorithm is updated in each of the two cases. We say that the branching factor is two, and we have two update formulas. We now define formally this concept in the general case.

We present below a simplified version of [21]; please refer to [21] for formal details and detailed proofs. We consider a (μ, λ)-ES (the same reasoning holds for $(\mu + \lambda)$-ES; see [21] for details and more generality). (μ, λ)-ES are as in Algorithm 2.1.

Of course, Algorithm 2.1 does not mean that the algorithm must absolutely be written under this form in order to be in the scope of the bounds in this paper; it must only be *equivalent* to an algorithm written under this form. In particular, most evolutionary algorithms (at least (μ, λ)-ES and $(\mu + \lambda)$-ES) can be rewritten in this form.

The constant K (i.e., the number of branches in the SWITCH) is the branching factor. Informally speaking, the branching factor is the number of different cases that the algorithm can consider, depending on the fitness values. For example, in a $(1, \lambda)$-ES, there are λ cases:

- The first individual can be the best;
- The second individual can be the best;
- ...
- The λth individual can be the best.

For a selection-based non-elitist (μ, λ) evolution strategy, there is one possible case for each possible subset of μ individuals among the λ generated individuals. More generally, the minimum number K such that the algorithm is still equivalent to the initial version should be considered. Therefore, we can consider a rewriting of

Algorithm 2.1. One iteration of a (μ, λ)-ES (simplified by the assumption that there are no ties). We here assume that all the y_j are distinct. Each S_i is a fixed permutation. $updateFormula1$ can be an arbitrary function, which modifies the internal state as a function of the x_i in case $y_1 < y_2 < y_3 < \cdots < y_\lambda$. There is such an update formula for each possible ranking of the $y_1, \ldots, y_\lambda$

```
One iteration of a $(\mu, \lambda)$-ES, for internal state of the algorithm $s$ and fitness function $f$
Compute individuals $x_1, \ldots, x_\lambda$ as a function of the internal STATE.
Compute their fitness values $y_1, \ldots, y_\lambda$ with $y_i = f(x_i)$.
Consider $S$ the permutation of $[[1, \lambda]]$ uniquely determined by: $y_{S(i)} < y_{S(i+1)}$
switch $S$ do
    case $S_1$
        $s = updateFormula1(s, x_1, \ldots, x_\lambda)$.break;
    endsw
    case $S_2$
        $s = updateFormula2(s, x_1, \ldots, x_\lambda)$.break;
    endsw
    case $S_3$
        $s = updateFormula3(s, x_1, \ldots, x_\lambda)$.break;
    endsw
    ... case $S_K$
        $s = updateFormulaK(s, x_1, \ldots, x_\lambda)$.break;
    endsw
    otherwise
        No other case should ever be raised.
    endsw
endsw
```

the algorithm as presented in Algorithm 2.2, where several cases are grouped into only one update formula in order to reduce the branching factor. For example, in the case of Selection-Based algorithms, K is at most $\binom{\lambda}{\mu}$: There are only $K = \binom{\lambda}{\mu}$ different update formula, one for each possible selected set. In the case of FR algorithm, there are at most $K = \binom{\lambda}{\mu} \mu!$ different update formulas.

We have seen how to upper bound the branching factor by rewriting the algorithm such that equivalent "cases" (permutations) are grouped together. This uses the fact that the formula is the same for several cases, for example, all rankings of the λ points which lead to the same selected set.

2.2.1 Using VC-Dimension

This advanced section assumes that the reader has some understanding of the concept of VC-dimension [26]. Intuitively, one can consider the VC-dimension of a set of functions as a measure of the combinatorial complexity of this set of functions. We also use the notion of level sets; the level set of f for r is the set of x in the domain of f such that $f(x) = r$.

Algorithm 2.2. A rewriting of Algorithm 2.1, by grouping cases leading to the same update formula. Please note that the number of cases leading to the same update formula does not need to be the same for all update formulas; the important number is only the total number of update formulas

```
One iteration of a SB-$(\mu, \lambda)$-ES, for internal state $s$ and fitness function $f$:
Compute individuals $x_1, \ldots, x_\lambda$ as a function of the internal state.
Compute their fitness values $y_1, \ldots, y_\lambda$ with $y_i = f(x_i)$.
Consider $S$ the permutation of $[[1, \lambda]]$ uniquely determined by:
$y_{S(i)} < y_{S(i+1)}$
switch $S$ do
    case $S_1$ ;
    case $S_2$ ;
    case $S_3$
        $s = updateFormula1(s, x_1, \ldots, x_\lambda)$.break;
    endsw
    case $S_4$ ;
    case $S_5$
        $s = updateFormula2(s, x_1, \ldots, x_\lambda)$.break;
    endsw
    ...
    case $S_{L-2}$ ;
    case $S_{L-1}$ ;
    case $S_L$
        $s = updateFormulaK(s, x_1, \ldots, x_\lambda)$.break;
    endsw
    otherwise
        No other case should ever be raised.
    endsw
endsw
```

However, a second tool can be used for removing some branches: Removing cases which are not possible because there is no fitness function which leads to this permutation S. At first view, all permutations are possible; however, some permutations are very unlikely. For example, it is very unlikely that the crosses with circles are selected in Fig. 2.1. The essential principle in [21], for improving bounds in [22] is to reduce the branching factor accordingly, thanks to VC-dimension assumptions to the VC-dimension of the set of fitness functions. Basically, the VC-dimension is finite in many cases of interest. Figure 2.1 gives an example of branches that can be cut under some assumptions on the fitness functions, and in Sect. 2.3 we give some quantitative classical bounds on VC-dimension. See [14] for a general introduction.

2.3 Complexity Bounds

Above we defined the branching factor; we will now use it for providing complexity bounds. Consider, for simplicity, the deterministic case (the general case is treated in references below). We see, with the branching factor, that the number of possible

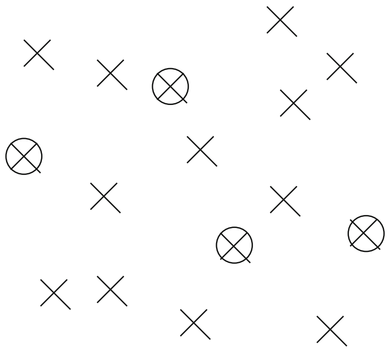


Fig. 2.1 Example of impossible selected set if the level sets are spheres: Branches corresponding to such selections can be discarded from the tree of possible behaviors of the algorithm. Therefore the branching factor is reduced. More generally, if the VC-dimension is V and if the number of individuals is λ, and if there are no ties, then the number of possibly selected sets is at most λ^V. The individuals are the crosses, with circles for the selected individuals. If we have a family of fitness functions which has some constraints, then we can remove the branches corresponding to fitness values which violate the constraints; quantifying the number of remaining branches in such a case is precisely why VC-dimension has been defined

distinct behaviors of the algorithm after a given number N of iterations is limited: There are K branches per iteration and N iterations, therefore the number of possible internal states of the algorithm is limited by K^N. On the other hand, if we want the algorithm to be able to find the optimum after N iterations, the number of possible outputs of the algorithm must be larger than the number of distinct optima – so if there are 2^d optima, we get $K^N \geq 2^d$. In this section we develop this idea more formally.

2.3.1 Convergence Ratio

We express bounds in terms of the convergence ratio. The convergence ratio is defined in [21] as

$$CR_\epsilon = \frac{\log N(\epsilon)}{dn_\epsilon}, \tag{2.1}$$

where

- n_ϵ is the number of iterations necessary for ensuring that with probability at least $\frac{1}{2}$, the algorithm has an estimate of the location of the optimum with precision ϵ for the Euclidean norm;
- d is the dimension of the search space;
- $N(\epsilon)$ is the maximum number k such that there exist k points in the domain with pairwise distance at least 2ϵ. Typically, $N(\epsilon)$ is equal to the cardinal of the search

space if it is finite and if ϵ is small enough (e.g., $N(\epsilon) = 2^d$ for a domain $\{0, 1\}^d$ if $\epsilon < 1$), and $N(\epsilon) = \Theta(\frac{1}{\epsilon^d})$ if the search space is an open subset of $\mathbb{R}^d$.

The constant $\frac{1}{2}$ is arbitrary, and very similar results are derived for a confidence $1-\delta$ with constant $\delta > 0$ (see [21] for more details). This definition implies that the complexity verifies

$$n_\epsilon = \frac{\log N(\epsilon)}{dCR_\epsilon}.$$

As discussed below, it is known that the convergence ratio for large values of λ can reach $CR_\epsilon = \Theta(\log(\lambda))$ for some algorithms; we will see that, for usual algorithms, this is not the case.

2.3.2 Link with the Convergence Rate

In the continuous case (which will be the main example throughout this chapter), this quantity is related to the convergence rate through the formula

$$-\log(\textit{convergence rate}) = \lim_{\epsilon \to 0} CR_\epsilon.$$

(see [21, 22] for more on this). The advantage of CR_ϵ is that it is inversely proportional to the computational cost for a given precision, and parallel speed-ups can be expressed by divisions between various CR_ϵ as follows: The speed-up of an algorithm x compared to an algorithm y is the ratio between the convergence ratio of x divided by the convergence ratio of y. This is particularly useful for estimating the benefit of adding processors; the curve $\lambda \mapsto CR$ at which $\lim_{\epsilon \to 0} CR(\epsilon)$ increases as a function of λ gives the parallel speed-up of the algorithm. Upper bounds on $CR(\epsilon)$, independently of the algorithm in a given class (e.g. selection-based algorithms, a wide and classical family), therefore provide universal upper bounds on the speed-up of the given class of parallel evolutionary algorithms.

2.3.3 Known Results

Combining combinatorial arguments on the branching factor and geometrical tricks (using VC-dimension), we get bounds provided in Table 2.1 on the convergence ratio [21, 22]. The VC-dimension of a set f of functions is, by definition, the maximum size of a set s such that all its subsets s' can be isolated from their complementary set $s \setminus s'$ by a function f, as follows:

$$\exists f; \max f(s') < \min f(s \setminus s')$$

Table 2.1 These formulas are upper bounds on the convergence ratio for all values of λ and μ, except the *last row*, which shows some lower bounds on the convergence ratio for $\lambda = 2d$ for the sphere function. These lower bounds from [21] show that a linear speed-up can be achieved w.r.t. λ, and that a constant convergence ratio (independent of the dimension) can be achieved for $\lambda = 2d$. The *first row* is the general case [22]; it might be better than other rows (for λ small). The *second row* is when the level sets of fitness functions have VC-dimension V in $\mathbb{R}^d$ [21]. The *third row* is just the application of the second row to the case of convex quadratic functions ($V = \Theta(d^2)$). The *fourth row* is the special case of the sphere function [21]. The tightness of the $\log(\lambda)$ can be shown by simple pattern search methods which reach such a speed-up

Framework	SB-(μ,λ)-ES	SB-$(\mu+\lambda)$-ES	FR-(μ,λ)-ES	FR-$(\mu+\lambda)$-ES
General case	$\frac{1}{d}\left(\lambda - \frac{1}{2}\log(2\pi\lambda)\right)$	$\frac{1}{d}\left(\log\binom{\lambda}{\mu}\right)$	$\left(\lambda - \frac{1}{2}\log(2\pi\lambda)\right) \times\frac{1}{d}\log(\mu!)$	$\left(\log\binom{\lambda}{\mu}\right) \times\frac{1}{d}\log(\mu!)$
General case, $\mu = 1$	$\frac{1}{d} \times\log(\lambda)$	$\frac{1}{d} \times\log(\lambda+1)$	$\frac{1}{d} \times\log(\lambda)$	$\frac{1}{d} \times\log(\lambda+1)$
VC-dim. V	$\frac{V}{d}\log(\lambda)$	$\frac{V}{d}\log(\lambda+\mu)$	$\frac{V}{d}(4\mu+\log(\lambda))$	$\frac{V}{d}(4\mu+\log(\lambda))$
Quadratic	$O(d\log\lambda)$	$O(d\log(\lambda+\mu))$	$O(d(\mu+\log\lambda))$	$O(d(\mu+\log\lambda))$
Sphere	$(1+\frac{1}{d})\log(\lambda)$	$\log(\mu+\lambda)(1+\frac{1}{d})$	$2\log(\lambda)$	$O(\mu+\log(\lambda))$
Sphere, $\lambda = 2d$	–	–	$\Omega(1)$	$\Omega(1)$

i.e., the VC-dimension V is

$$V = \max\{v \geq 0; \exists s_1,\dots,s_v \text{ such that } \forall s' \subset s,\ \max f(s') < \min f(s \setminus s')\}.$$

The VC-dimension of a set of quadratic functions in dimension d is $O(d^2)$, whereas the VC-dimension of a set of sphere functions or a set of linear functions is $O(d)$.

Basically, these results show that

- We can have a convergence ratio increasing as $\log(\lambda)$ as $\lambda \to \infty$; this shows the asymptotic behavior on parallel machines;
- We can have a linear improvement on the convergence ratio (as a function of λ) for values of λ that are not too high, i.e., essentially $\lambda \leq d$ where d is the dimension;
- The dependency on μ is not completely known.

2.4 The Limited Speed-Up of Many Real-World Algorithms

There are a lot of different methods for the *updateFormula* function. The internal state to be updated contains usually, at least, (i) the step-size σ (which is, roughly speaking, the scale of mutations) and (ii) the mean of a Gaussian distribution, to be used for generating new points. The most difficult issue is usually the update of σ; σ is the standard deviation of the Gaussian distribution used in Algorithm 2.3.

Algorithm 2.3. A typical evolutionary algorithm in the continuous domain

```
$\sigma = \sigma_0, x_0 \in \mathbb{R}^d$.
$n = 0$  // iteration number
**while** halting criterion not reached **do**
    $x_{n,1}, x_{n,2}, \ldots, x_{n,\lambda} = Gaussian(x_n, \sigma_n)$
    $\forall i \in [[1, \lambda]],\ y_{n,i} = f(x_{n,i})$
    $(x_{n+1}, \sigma_{n+1}) = updateFormula(x_{n,1}, x_{n,2}, \ldots, x_{n,\lambda}, y_{n,1}, y_{n,2}, \ldots, y_{n,\lambda})$
    $n \leftarrow n + 1$
**end while**
```

Three of the most widely known methods for updating σ are the one-fifth rule [15], self-adaptation (SA) [15, 17] and cumulative step-size adaptation (CSA) [11]. We saw in Sect. 2.1 that the optimal speed-up, for λ sufficiently large, cannot be better than $\log(\lambda)$. It is also known (e.g., [21]) that this can be reached. We will see now that the methods above do not reach the optimal speed-up, at least under their usual specifications.

We will use $\eta^* = \sigma_{n+1}/\sigma_n$. η^* small means that σ decreases quickly, whereas η^* close to 1 means a slow decrease, and the key point is that a fast convergence to the optimum implies a fast convergence of σ (see [1] for more on this). More precisely, the log of the convergence rate is lower bounded by $\mathbb{E} \log \eta^*$. Therefore, if we can build an absolute lower bound $\mathbb{E} \log \eta^* = \log(\Omega(1))$, this will provide an absolute upper bound on the convergence ratio, or, if you prefer, a lower bound on the convergence rate. More formally, $\mathbb{E} \log \eta^* = \log(\Omega(1))$ implies $CR_\epsilon = O(1)$. This is not optimal, as we know that the convergence ratio should be $\Theta(\log(\lambda))$ for well-designed algorithms. Equivalently, the convergence rate should be $\exp(-\Theta(\log(\lambda))$.

We will now see that some well-known algorithms have this undesirable property "$\mathbb{E}(\log \eta^* | x_n, \sigma_n) \geq C$" for some $C > -\infty$ and independent of λ. This shows that improvements are possible here: The difference between the $\Theta(\log \lambda)$ (predicted by theory) and the $O(1)$ in algorithms below show that the algorithms cited below (which cover most of evolutionary algorithms in the continuous domains) are not optimal.

2.4.1 The One-Fifth Rule

The one-fifth rule is a very common rule for the update of σ. The idea is to increase σ if the probability of success p is greater than $\frac{1}{5}$, and to decrease it otherwise. The probability of success is the probability that an offspring is better than its parent (Formally, in case of minimization, there is success for the ith offspring if $f(x_{n,i}) < f(x_n)$).

A usual implementation is

- $p \leq \frac{1}{5} \Rightarrow \eta^* = K_1 \in (0, 1)$, corresponding to the decreasing case, so we want $\sigma_{n+1} < \sigma_n$.

- $p > \frac{1}{5} \Rightarrow \eta^* = K_2 > 1$, corresponding to the increasing case, so σ_{n+1} must be greater than σ_n.

It is easy to see that, in the first case, $\eta^* = K_1 > 0$, and in the second case $\eta^* > 1$. Therefore, there exists a constant C such that $\mathbb{E} \log \eta^* > C$.

The one-fifth rule can also be expressed as $\eta^* = K_3^{p-\frac{1}{5}}$. Here also, it is easy to see that $\eta^* \geq K_3^{-\frac{1}{5}} > 0$, therefore the same conclusion holds, namely $\mathbb{E} \log \eta^* \geq C > -\infty$, for some C independent of λ. The one-fifth rule does not have the optimal speed-up $\log(\lambda)$ with its usual parametrization. Increasing K_3 (as a function of λ) might solve this.

2.4.2 Self-adaptation (SA)

Self-adaptation (SA) is another well-known algorithm for choosing the step-size (Algorithm 2.4). Here, $\mathcal{N}_i$ are independent standard Gaussian random variable, η^* is an average between log-normal random variables. Unfortunately, if $\mu = \lfloor \lambda/4 \rfloor$, then even if the μ selected mutations correspond to the smallest values of the mutations σ_i, $\log(\sigma)$ is, on average, decreased by $-\log(\tau Q_{\frac{1}{4}} \mathcal{N})$, where $Q_{\frac{1}{4}} \mathcal{N}$ is the average of the first quartile of the standard Gaussian variable. One can show [25], as for the one-fifth rule, that $\mathbb{E} \log(\eta^*) \geq C > -\infty$. Therefore, SA does not have the optimal speed-up $\log(\lambda)$ with its usual parametrization. Increasing τ (as a function of λ) might solve this.

2.4.3 Cumulative Step-Size Adaptation (CSA)

A third method for updating the step size is the cumulative step-size adaptation (CSA). This method looks at the path followed by the algorithm, and compares its length to the expected length under random selection. CSA increases σ if the first path is greater than the second one, and decreases σ otherwise.

We formalize an iteration of CSA with weights $w_1, \ldots, w_\mu$ in dimension d as follows; we do not have to assume anything on the constant χ_d and the constant p_c except assumptions (2.7) and (2.8):

$$\sigma_{n+1} = \sigma_n \exp\left(\left(\frac{||p_c||}{\chi_d} - 1\right) \cdot \frac{c_\sigma}{d_\sigma}\right) \tag{2.2}$$

$$\sum_{i=1}^{\mu} w_i = 1 \tag{2.3}$$

Algorithm 2.4. Self-adaptation algorithm. τ usually depends on the dimension. As well as for the one-fifth rule and cumulative step-size adaptation, the possible speed-up is $\Theta(1)$ independently of λ

```
Initialize $\sigma^{avg} \in \mathbb{R}$, $x_0 \in \mathbb{R}^d$
$n = 0$   // iteration number
while We have time do
  for $i = 1..\lambda$ do
    $\sigma_i = \sigma^{avg} e^{\tau \mathcal{N}_i(0,1)}$
    $z_i = \sigma_i \mathcal{N}_i(0, Id)$   // mutation for $i^{th}$ individual
    $x_{n,i} = x_n + z_i$
    $f_i = f(x_{n,i})$
  end for
  Sort the individuals by increasing fitness; $f_{(1)} < f_{(2)} < \cdots < f_{(\lambda)}$.
  $z^{avg} = \frac{1}{\mu} \sum_{i=1}^{\mu} z_{(i)}$
  $\sigma^{avg} = \frac{1}{\mu} \sum_{i=1}^{\mu} \sigma_{(i)}$
  $x_{n+1} = x_n + z^{avg}$
  $n \leftarrow n + 1$
end while
```

$$\mu_{\text{eff}} = \frac{1}{\sum_{i=1}^{\mu}(w_i^2)} \tag{2.4}$$

$$d_\sigma = 1 + 2\max(0, \sqrt{\frac{\mu_{\text{eff}} - 1}{d + 1}} - 1) \tag{2.5}$$

$$c_\sigma = \frac{\mu_{\text{eff}} + 2}{d + \mu_{\text{eff}} + 3} \tag{2.6}$$

$$\chi_d > 0 \tag{2.7}$$

$$||p_c|| \geq 0. \tag{2.8}$$

($||.||$ does not have to be a norm, we just need Eq. (2.8).) These assumptions, to the best of our knowledge, hold in all current implementations of CSA. They do not completely specify the algorithm, but are sufficient for our purpose.

One can easily show that Eqs. (2.3)–(2.8) imply that $\forall \lambda, \mathbb{E}(\log \eta^* | x_n, \sigma_n) \geq -1$; for this algorithm also, we see that $\exists C, \forall \lambda, \mathbb{E} \log \eta^* \geq C > -\infty$. CSA does not have the optimal speed-up $\log(\lambda)$ with its usual parametrization. Increasing c_σ / d_σ might solve this.

2.5 Implications

The above results have implications on practice. Several of them concern parallel evolutionary algorithms; parallelism is crucial in evolutionary algorithms as they are population-based and therefore are easy to parallelize, at least when the computational cost of the fitness is large in front of communication costs.

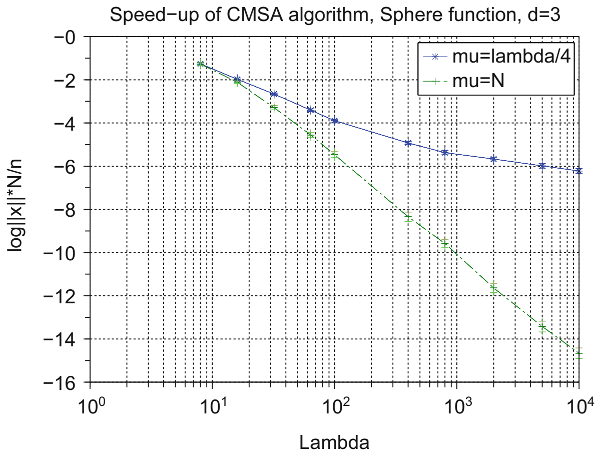


Fig. 2.2 Example of the limited speed-up of real-world algorithms on the sphere-function $x \mapsto ||x||^2$ from [20]. The Covariance Matrix Self-adaptation Evolution Strategy (CMSA-ES) is an algorithm using the self-adaptation rule combined with a full covariance matrix. This experiment is done in dimension 3, and we look at the distance to the optimum normalized by the dimension, divided by the number of generations of the algorithm (the lower the result, the better; this is a normalized convergence rate). With usual initialization, we have a selection ratio $\frac{\mu}{\lambda}$ equals to $\frac{1}{4}$. As we can note, using a smaller selection ratio (here the selection ratio equal to $\min(d, \lfloor\lambda/4\rfloor)/\lambda$) is a much better choice. With this improvement we can reach the theoretical logarithmic speed-up

Changing typical algorithms for large λ. The first consequence, around parallelism is implied by the combination of Sect. 2.3 (which shows complexity bounds) and Sect. 2.4 (which shows the speed-up of usual algorithms like cumulative step-size adaptation, the one-fifth rule, and self-adaptation). The results show that these three rules, with typical parameters, cannot reach the logarithmic speed-up $\Theta(\log(\lambda))$ for λ large, and have a bounded speed-up $\Theta(1)$. However, this might be easy to modify by adapting constants; for example, increasing the lognormal mutation strength as a function of λ, for the self-adaptation of σ, might solve this issue. Also, modifying $\frac{c_\sigma}{d_\sigma}$ as a function of λ, for CSA, might solve this issue. As an illustration, we show in Fig. 2.2 the great improvement provided by the reduction of μ (in order to avoid the weakness pointed out in Sect. 2.4.2) on the most recent SA variant. This is certainly an example of theory which has a direct impact on practice. As we can see in Fig. 2.2 more than 100 % speed-up on our graph, which increases as the number of processors increases, with only one line of code modified in SA.

Choice of algorithm, given a number λ of processors. Let us consider the choice of an algorithm as a function of λ; this is the case in which λ is equal to the number of computing units available. New machines have an increasing number of cores, clusters or grids have thousands of cores, and since "jobs" submitted on grids must sometimes be grouped, the value of λ can be huge, beyond tens of thousands.

It is known [4] that evolution strategies do not all have the same speed-up. If λ is small in comparison with the dimension, $(\mu/\mu,\lambda)$-ES can reach a linear speed-up ,[1] whereas $(1,\lambda)$-ES have only logarithmic speed-up. If λ is small or of the same order as the dimension, this suggests that $(\mu/\mu,\lambda)$ strategies scale better than $(1,\lambda)$ algorithms.

For large λ, [21] (summarized in Table 2.1) has shown that the theoretical speed-up is $\Theta(\log(\lambda))$ for both algorithms (namely $(\mu/\mu,\lambda)$ and $(1,\lambda)$), at least for good parametrizations of these families of algorithms. However, as shown in Sect. 2.4, most usual $(\mu/\mu,\lambda)$ evolution strategies have limited speed-up $\Theta(1)$, and therefore their speed-up is much worse than $(1,\lambda)$-ES which reaches $\Theta(\log(\lambda))$! Should we deduce from this that $\mu = 1$ is better when λ is large? In fact, choosing μ linear as a function of λ (roughly one-fourth in many papers) is not a good idea for algorithms based on recombination by averaging. Maybe $\mu = \min(d, \lambda/4)$ could be a good idea; but this will not be sufficient (except maybe for SA). Our results are independent of μ in CSA, therefore changing μ in CSA is not sufficient for ensuring $\log(\lambda)$ speed-up in CSA, but preliminary investigations suggests that this formula for μ, combined with the modifications suggested above on $\frac{c_\sigma}{d_\sigma}$, might give good results.

This suggests that a lot of work remains around the case of λ larger than the dimension. In particular, we have shown [24] that Estimation of Multivariate Normal Algorithm (EMNA) [12] is, for large λ, much faster than most usual algorithms. In [23] we have shown that some algorithms could be improved by a factor of 8 by combining tricks dedicated to λ large.

Implications for multiobjective algorithms. An original application of the branching factor and of bounds derived with it is discussed in [19]. We have seen in Eq. (2.1) that the convergence ratio depends on the packing numbers (i.e., $N(\epsilon)$). This means that the number of fitness evaluations strongly depends on the packing number, i.e., for a precision ϵ, the number of disjoint balls of radius ϵ that can be put in the domain. Unfortunately in the multiobjective case, when the number of conflicting objectives is large, then the packing number is huge. Thanks to this principle, in [19] we have shown that, even if we restrict our attention to problems with Lipschitzian Pareto fronts, finding the optimal Pareto front with precision ϵ and confidence $1-\delta$ for the Hausdorff metric requires a number of fitness evaluations $\Omega(1/\epsilon^{d-1}) + \log_2(1-\delta)$:

- If the algorithm is based on Pareto-comparisons between individuals (i.e., we only know which points dominate other points);
- And if the number of objectives[2] is d.

[1]The optimal speed-up is asymptotically logarithmic, but as explained in this chapter non-asymptotically we can reach a linear speed-up (until $\lambda = \Theta(d)$).

[2]Importantly, the result is based on the fact that those objectives can all be conflicting – results are very different if we forbid too many conflicting objectives.

Consider a d-dimensional fitness function f (with values in $[0,1]^d$). Then, the Pareto set is a subset of $[0,1]^d$. The rate above is, for d large, close to the efficiency of a random search using a distribution of x such $f(x)$ is uniform in the Pareto set; namely the rate of this random search is $O(\epsilon^{-d})$. This means that all comparison-based algorithms are basically not much faster than random search. Comparison-based algorithms require, if they want to be significantly faster than random search for d large, either:

- Some feedback from the human user;
- Or some more information on the fitness (see, for example, informed operators [16]);
- Or moderately many conflicting objectives (see in particular [5]).

2.6 Conclusions

We have summarized theoretical complexity bounds for algorithms based on selection, on full ranking of selected individual, or on full ranking of the complete population; this covers most evolutionary algorithms. Many of these bounds are shown tight within constant factors. We have shown that many real-world algorithms are far from these complexity bounds, when λ is large. This suggests several modifications for real-world algorithms, easy to implement and which both provably (see Sect. 2.4 compared to bounds in Sect. 2.3) and experimentally (see Fig. 2.2) greatly improve the results for λ large.

References

1. A. Auger, Convergence results for (1,λ)-SA-ES using the theory of φ-irreducible Markov chains. Theor. Comput. Sci. **334**, 35–69 (2005)
2. T. Bäck, F. Hoffmeister, H.-P. Schwefel, Extended selection mechanisms in genetic algorithms, in *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Diego, ed. by R.K. Belew, L.B. Booker (Morgan Kaufmann, 1991)
3. J.E. Baker, Reducing bias and inefficiency in the selection algorithm, in *Proceedings of the Second International Conference on Genetic Algorithms and Their Application*, Cambridge (Lawrence Erlbaum, 1987), pp. 14–21
4. H.-G. Beyer, *The Theory of Evolution Strategies* (Springer, Heidelberg, 2001)
5. D. Brockhoff, E. Zitzler, Objective reduction in evolutionary multiobjective optimization: theory and applications. Evol. Comput. **17**(2), 135–166 (2009)
6. C.G. Broyden, The convergence of a class of double-rank minimization algorithms 2. New Algorithm J. Inst. Math. Appl. **6**, 222–231 (1970)
7. A. Conn, K. Scheinberg, L. Toint, Recent progress in unconstrained nonlinear optimization without derivatives. Math. Program. **79**, 397–414 (1997)
8. R. Fletcher, A new approach to variable-metric algorithms. Comput. J. **13**, 317–322 (1970)
9. S. Gelly, S. Ruette, O. Teytaud, Comparison-based algorithms are robust and randomized algorithms are anytime. Evol. Comput. Special Issue on Bridging Theory and Practice **15**(4), 26 (2007)

10. D. Goldfarb, A family of variable-metric algorithms derived by variational means. Math. Comput. **24**, 23–26 (1970)
11. N. Hansen, A. Ostermeier, Completely derandomized self-adaptation in evolution strategies. Evol. Comput. **9**(2), 159–195 (2001)
12. P. Larranaga, J.A. Lozano, *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation* (Kluwer, Boston, 2001)
13. I. Loshchilov, M. Schoenauer, M. Sebag, Comparison-based optimizers need comparison-based surrogates, in *Parallel Problem Solving from Nature (PPSN XI)*, ed. by R. Schaefer et al. LNCS, vol. 6238 (Springer, New York, 2010), pp. 364–373
14. J. Matoušek, *Lectures on Discrete Geometry*. Graduate Texts in Mathematics, vol 212 (Springer, New York, 2002)
15. I. Rechenberg, *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution* (Frommann-Holzboog Verlag, Stuttgart, 1973)
16. O. Rudenko, M. Schoenauer, Dominance based crossover operator for evolutionary multi-objective algorithms. *CoRR*, abs/cs/0505080 (2005)
17. H.-P. Schwefel, Adaptive Mechanismen in der biologischen Evolution und ihr Einfluss auf die Evolutionsgeschwindigkeit. Interner Bericht der Arbeitsgruppe Bionik und Evolutionstechnik am Institut für Mess- und Regelungstechnik Re 215/3, Technische Universität Berlin, Juli 1974.
18. D.F. Shanno, Conditioning of quasi-Newton methods for function minimization. Math. Comput. **24**, 647–656 (1970)
19. O. Teytaud, On the hardness of offline multiobjective optimization. Evol. Comput. **15**, 475–491 (2007)
20. F. Teytaud, A new selection ratio for large population sizes, in *Applications of Evolutionary Computation*. LNCS, vol. 6024 (Springer, Berlin, 2010), pp. 452–460
21. O. Teytaud, H. Fournier, Lower bounds for evolution strategies using VC-dimension, in *Proceedings of PPSN*, Dortmund (Springer, 2008), pp. 102–111
22. O. Teytaud, S. Gelly, General lower bounds for evolutionary computation, in *Proceedings of PPSN*, Reykjavik (Springer, 2006), pp. 21–31
23. F. Teytaud, O. Teytaud, Bias and variance in continuous EDA, in *Proceedings of EA09*, New York. LNCS (Springer, 2009)
24. F. Teytaud, O. Teytaud, On the parallel speed-up of estimation of multivariate normal algorithm and evolution strategies, in *Proceedings of EvoStar*, Tübingen, 2009, pp. 655–664
25. F. Teytaud, O. Teytaud, Log(lambda) modifications for optimal parallelism, in *Parallel Problem Solving from Nature* (Springer, New York, 2010)
26. V. Vapnik, A. Chervonenkis, On the uniform convergence of frequencies of occurence events to their probabilities. Sov. Math. Dokl. **9**, 915–918 (1968)
27. D. Whitley, The GENITOR algorithm and selection pressure: why rank-based allocation of reproductive trials is best, in *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, ed. by J.D. Schaffer (Morgan Kaufmann, 1989)

Chapter 3
Rugged and Elementary Landscapes

Konstantin Klemm and Peter F. Stadler

Abstract The landscape of an optimization problem combines the fitness (or cost) function f on the candidate set X with a notion of neighborhood on X, typically represented as a simple sparse graph. A landscape forms the substrate for local search heuristics including evolutionary algorithms. Understanding such optimization techniques thus requires insight into the connection between the graph structure and properties of the fitness function.

Local minima and their gradient basins form the basis for a decomposition of landscapes. The local minima are nodes of a labeled graph with edges providing information on the reachability between the minima and/or the adjacency of their basins. Barrier trees, inherent structure networks, and funnel digraphs are such decompositions producing "coarse-grained" pictures of a landscape.

A particularly fruitful approach is a spectral decomposition of the fitness function into eigenvectors of the graph Laplacian, akin to a Fourier transformation of a real

K. Klemm (✉)
Bioinformatics Group, Department of Computer Science, Interdisciplinary Center for Bioinformatics, University of Leipzig, D-04107 Leipzig, Germany
e-mail: klemm@bioinf.uni-leipzig.de

P.F. Stadler
Bioinformatics Group, Department of Computer Science, Interdisciplinary Center for Bioinformatics, University of Leipzig, D-04107 Leipzig, Germany

Max Planck Institute for Mathematics in the Sciences, Inselstraße 22, D-04103 Leipzig, Germany

Fraunhofer Institut für Zelltherapie und Immunologie – IZI Perlickstraße 1, D-04103 Leipzig, Germany

Department of Theoretical Chemistry, University of Vienna, Währingerstraße 17, A-1090 Wien, Austria

Santa Fe Institute, 1399 Hyde Park Rd., Santa Fe, NM 87501, USA
e-mail: peter.stadler@bioinf.uni-leipzig.de

function into the elementary waves on its domain. Many landscapes of practical and theoretical interest, including the Traveling Salesman Problem with transpositions and reversals, are elementary: Their spectral decomposition has a single non-zero coefficient. Other classes of landscapes, including k-satisfiability (K-SAT), are superpositions of the first few Laplacian eigenvectors. Furthermore, the ruggedness of a landscape, as measured by the correlation length of the fitness function, and its neutrality, the expected fraction of a candidate's neighbors having the same fitness, can be expressed by the spectrum. Ruggedness and neutrality are found to be independently variable measures of a landscape. Beyond single instances of landscapes, models with random parameters, such as spin glasses, are amenable to this algebraic approach.

This chapter provides an introduction into the structural features of discrete landscapes from both the geometric and the algebraic perspective.

3.1 Introduction

The concept of a *fitness landscape* originated in the 1930s in theoretical biology [75, 76] as a means of conceptualizing evolutionary adaptation: A fitness landscape is a kind of potential function on which a population moves uphill due to the combined effects of mutation and selection. Thus, natural selection acts like hill climbing on the topography implied by the fitness function.

From a mathematical point of view, a landscape consists of three ingredients: a set X of configurations that are to be searched, a topological structure $\mathcal{T}$ on X that describes how X can be traversed, and a fitness or cost function $f : X \to \mathbb{R}$ that evaluates the individual points $x \in X$. This generic structure is common to many formal models, from evolutionary biology to statistical physics and operations research [5, 26, 40, 41]. The topological structure $\mathcal{T}$ is often specified as a move set, that is, as a function $N : X \to \mathfrak{P}(X)$ specifying for each configuration $x \in X$ the subset $N(x) \subseteq X$ of configurations that are reachable from x. Usually, the move set is constructed in a symmetric way, satisfying $x \in N(y) \iff y \in N(x)$. The configuration space or *search space* $(X, \mathcal{T})$ becomes an undirected finite graph G in this case. Then for each $x \in X$, the degree of x is given by $\deg(x) = |N(x)|$. We say that G is d-regular for non-negative integer d if $\deg(x) = d$ for all $x \in X$.

Without losing generality we assume that optimization strives to find those $x \in X$ for which $f(x)$ is *small*. Thus, f is interpreted as energy (or fitness with negative sign), and we are particularly interested in the minima of f. A configuration $x \in X$ is a *local minimum* if $f(x) \le f(y)$ for all $y \in N(x)$. By M we denote the set of all local minima of the landscape. A local minimum $\hat{x} \in M$ is global if, for all $y \in X$, $f(\hat{x}) \le f(y)$.

The Traveling Salesman Problem (TSP) [22], see Fig. 3.1, may serve as an example. Given a set of n vertices (cities, locations) $\{1, \dots, n\}$ and a (not necessarily

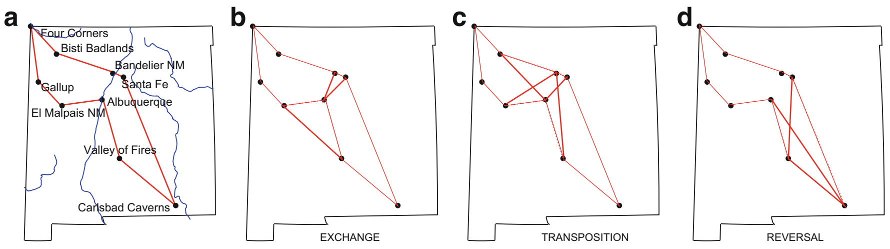


Fig. 3.1 A small instance (**a**) of the Traveling Salesman Problem with illustration of three different move types (**b**)–(**d**) described in the main text

symmetric) matrix of distances or travel costs d_{ij}, the task is to find the permutation (tour) π that minimizes the total travel cost

$$f(\pi) = \sum_{i=1}^{n} d_{\pi(i),\pi(i+1)} \tag{3.1}$$

where indices are interpreted modulo n. A landscape arises by imposing a topological structure on the set S_n of permutations, i.e., by specifying a move set that determines which tours are adjacent. Often the structure of the problem suggests one particular move set as natural, while others might look quite far-fetched. In principle, however, the choice of the move set is independent of the function f. For the TSP, for instance, three choices come to mind:

1. *Exchange (Displacement)* moves relocate a single city to another position in the tour

$$\begin{aligned}&(1,\ldots,i,i+1,i+2,\ldots j,j+1,\ldots,n)\\ &\quad\mapsto(1,\ldots,i,i+2,\ldots j,i+1,j+1,\ldots,n)\,.\end{aligned}$$

   See Fig. 3.1b.
2. *Reversals* cut the tour in two pieces and invert the order in which one half is transversed.

$$\begin{aligned}&(1,\ldots,i,i+1,i+2,\ldots,k-2,k-1,k,\ldots n)\\ &\quad\mapsto(1,\ldots,i,k-1,k-2,\ldots,i+2,i+1,k,\ldots n)\,.\end{aligned}$$

   See Fig. 3.1c.
3. *Transpositions* exchange the location of a single city

$$\begin{aligned}&(1,\ldots,i-1,i,i+1,\ldots,k-1,k,k+1\ldots n)\\ &\quad\mapsto(1,\ldots,i-1,k,i+1,\ldots,k-1,i,k+1,\ldots n)\,.\end{aligned}$$

   See Fig. 3.1d.

3.2 Ruggedness

The notion of *rugged landscapes* dates back to the 1980s, when Stuart Kauffman began to investigate these structures in some detail [32]. The intuition is simple. Once the landscape is fixed by virtue of the move set, optimization algorithms making local moves w.r.t. this move set will "see" a complex topography of mountain ranges, local optima, and saddles that influences their behavior. The more "rugged" this topography, the harder it becomes for an optimization algorithm based on local search to find the global optimum or even a good solution. Ruggedness can be quantified in many ways, however [29]:

1. The length distribution of down-hill walks, either gradient descent or so-called *adaptive walks*, is easily measured in computer simulations but is hard to deal with rigorously, see, e.g., [37].
2. Richard Palmer [45] proposed to call a landscape f rugged if the number of local optima scales exponentially with some measure of system size, e.g., the number of cities of a TSP. The distribution of local optima is in several cases accessible by the toolkit of statistical mechanics, see e.g. [50].
3. Barrier trees give a much more detailed view compared to local optima alone. Gradient walks are implicitly included in barrier tree computations. Section 3.3 of this chapter reviews barrier trees and other concise representations of landscapes.
4. Correlation measures capture landscape structure in terms of the time scales on which the cost varies under a random walk. These involve algebraic approaches with spectral decomposition, which are the topic of Sects. 3.4 and 3.5.

3.3 Barriers

Optimization by local search or gradient walks is guaranteed to be successful in a landscape with a unique cost minimum. In all other cases, variations of local search are more suitable [14, 33] when they accept inferior solutions to some extent. Then the walk eventually overcomes a barrier to pass over a saddle and enters the basin of an adjacent – possibly lower – local minimum. In this section we formalize the notions of walks, barriers, saddles, basins, and their adjacency. They form the basis of coarse-grained representations of landscapes. These representations "live" on the set of local minima M which is typically much smaller than the set of configurations X.

We review four such representations. Inherent structure networks contain complete information about adjacency of the gradient walk basins of the local minima. Barrier trees describe for each pair of local minima the barrier, i.e., the increase in cost to be overcome in order to travel between the minima. The funnel digraph displays the paths taken by an idealized search dynamics that takes the exit at lowest cost from each basin. Valleys were recently introduced to formalize the

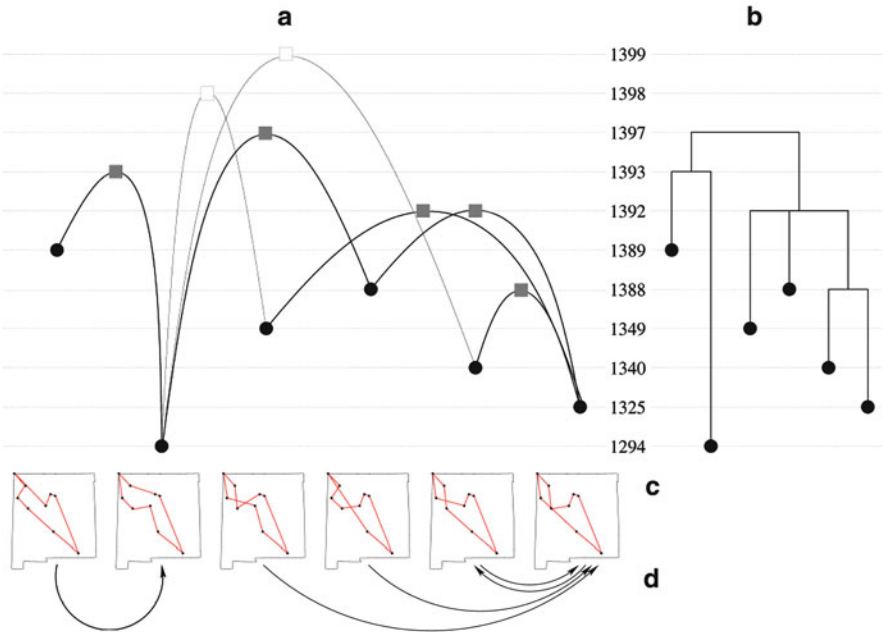


Fig. 3.2 Graphical representations of local minima and saddles for the TSP instance of Fig. 3.1 under transpositions. (**a**) Local minima (*circles*) and direct saddles between their basins (y-axis not drawn to scale). The inherent structure network is the complete graph because a direct saddle exists for each pair of minima. In the illustration, however, saddles at a cost above 1,400 have been omitted. (**b**) The barrier tree for the same landscape. An inner node (*vertical line*) of the barrier tree indicates the minimal cost to be overcome in order to travel between all minima in the subtrees of that node. In (**a**), the direct saddles relevant for the construction of the barrier tree are drawn as *filled rectangles*. (**c**) Local minima are those TSP tours without neighboring tours (configurations) of lower cost. Note that this depends on the move set: Under reversals rather than transpositions, the two rightmost configuration are adjacent. (**d**) In the funnel digraph, an arc $a \rightarrow b$ indicates that the lowest direct saddle from minimum a leads to minimum b

landscape structure implied by adaptive walks. Figure 3.2 serves as a comprehensive illustration throughout this section.

3.3.1 Walk and Accessibility

For configurations $x, y \in X$, a walk from x to y is a finite sequence of configurations $\mathbf{w} = (w_0, w_1, \ldots, w_l)$ with $w_0 = x$, $w_l = y$, and $w_i \in N(w_{i-1})$ for all $i \in \{1, 2, \ldots, l\}$. By $\mathbb{P}_{xy}$ we denote the set of all walks from x to y. We say that x and y are *mutually accessible at level* $\eta \in \mathbb{R}$, in symbols

$$x \xleftrightarrow{\;\eta\;} y\,, \tag{3.2}$$

if there is a walk $\mathbf{w} \in \mathbb{P}_{xy}$ such that $f(z) \leq \eta$ for all $z \in \mathbf{w}$. The *saddle height* or *fitness barrier* between two configurations $x, y \in X$ is the smallest level at which x and y are mutually accessible,

$$f[x, y] = \min\{\eta \in \mathbb{R} | x \overset{\eta}{\longleftrightarrow} y\} \,. \tag{3.3}$$

The saddle height fulfills the *ultrametric* (strong triangle) inequality. Namely, for all configurations $x, y, z \in X$

$$f[x, z] \leq \max\{f[x, y], f[y, z]\} \,. \tag{3.4}$$

because the set of walks from x to z passing through y is a subset of $\mathbf{P}_{xz}$.

3.3.2 Barrier Tree

Equipped with these concepts, let us return to the consideration of local minima. When restricting arguments to elements of M, the saddle height $f : M \times M \to \mathbb{R}$ still fulfills the ultrametric inequality and thereby induces a hierarchical structure on M [47]. To see this, we use the maximum saddle height

$$m = \max\{f[\hat{x}, \hat{y}] | \hat{x}, \hat{y} \in M\} \tag{3.5}$$

of the whole landscape to define a relation on M by

$$\hat{x} \sim \hat{y} :\Leftrightarrow f[\hat{x}, \hat{y}] < m \,. \tag{3.6}$$

If all local minima are pairwise non-adjacent, $\sim$ is an equivalence relation on M. Its transitivity follows directly from the ultrametric inequality for the saddle height. Unless $|M| = 1$, there is at least one pair of minima that are not related. Therefore the relation $\sim$ generates a partitioning of M into at least two equivalence classes. The argument may then be applied again to each class containing more than one element. This recursion of the partitioning into equivalence classes generates the *barrier tree*. Its leaves are the singleton classes, i.e., the minima themselves. Each inner node stands for a (sub-)partitioning at a given saddle height.

Barrier trees serve to reveal geometric differences between landscapes, e.g., by measuring tree balance [63]. Figure 3.3 shows an example for the number partitioning problem [22]. Standard "flooding" algorithms construct the barrier tree by agglomeration rather than division because the minima and saddle heights are not known a priori. By scanning the configurations of the landscape in the order of increasing fitness, local minima are detected and joined by an inner tree node when connecting walks are observed [52, 73]. Barrier trees can be defined also for *degenerate* landscapes where the assumption of non-adjacent local minima is not fulfilled [17].

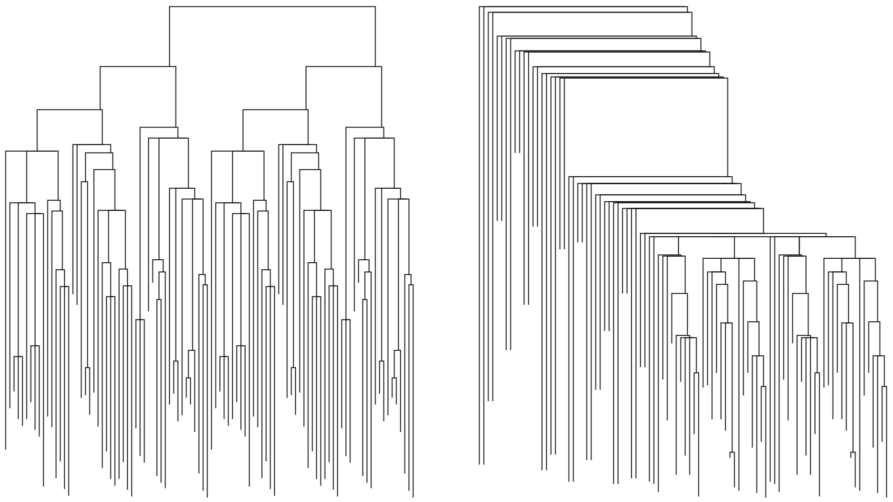


Fig. 3.3 Barrier trees of two fitness landscapes of the same size. *Left tree*: an instance of the number partitioning problem of size $N = 10$. *Right tree*: an instance of the truncated random energy model. The trees obtained for the two types of landscapes are distinguishable by measures of tree imbalance [63]

3.3.3 Basins and Inherent Structure Network

More detailed information about the geometry in terms of local minima is captured by basins and their adjacency relation. A walk $(w_0, w_1, \ldots, w_l)$ is a *gradient walk* (or *steepest descent*) if each visited configuration w_i is the one with lowest fitness in the closed neighborhood of its predecessor,

$$f(w_i) = \min\{f(x) | x \in N(w_{i-1}) \cup w_{i-1}\}, \ 1 \leq i \leq l \ . \tag{3.7}$$

From each starting configuration w_0, a sufficiently long gradient walk encounters a local minimum $w_l \in M$. If the encountered minimum $g(x)$ is unique given a starting configuration $x \in X$ of a gradient walk, this defines a mapping $g : X \rightarrow M$. In the case that neighbors with lowest fitness are not unique, ambiguity of gradient walks is resolved by an additional, e.g., lexicographic ordering on X. The *basin* or *gradient basin* of a local minimum $\hat{x}$ is the set $B(\hat{x}) = g^{-1}(\hat{x})$ of configurations from which a gradient walk leads to $\hat{x}$. Each basin is non-empty because it contains the local minimum itself. Since each configuration $x \in X$ is in the basin of exactly one local minimum $g(x)$, basins are a partitioning of the set X of configurations.

The *interface* between two local minima $\hat{x}, \hat{y} \in M$ is the set

$$I(\hat{x}, \hat{y}) = \{(x, y) | x \in B(\hat{x}), y \in B(\hat{y}), x \in N(y)\} \tag{3.8}$$

containing all pairs of adjacent configurations shared between the basins. The *direct saddle height* between $\hat{x}$ and $\hat{y}$ is

$$h[\hat{x}, \hat{y}] = \min\{\max\{f(x), f(y)\} | (x, y) \in I(\hat{x}, \hat{y})\} \ . \tag{3.9}$$

Direct saddle height is lower bounded by saddle height,

$$h[\hat{x}, \hat{y}] \geq f[\hat{x}, \hat{y}] \tag{3.10}$$

for all $\hat{x}, \hat{y} \in M$ with non-empty interface. A member of the interface $(x, y) \in I(\hat{x}, \hat{y})$ is a *direct saddle* (between $\hat{x}$ and $\hat{y}$) if its cost is the direct saddle height $\max\{f(x), f(y)\} = h(\hat{x}, \hat{y})$.

The *inherent structure network* (M, H) [13] is defined as a graph with node set M and edge set $H = \{\{\hat{x}, \hat{y}\} | I(\hat{x}, \hat{y}) \neq \emptyset\}$. An edge thus connects the local minima $\hat{x}$ and $\hat{y}$ if and only if there is a path from $\hat{x}$ to $\hat{y}$ that lies in the union of basins $B(\hat{x}) \cup B(\hat{y})$. The saddle heights $f[\hat{x}, \hat{y}]$ can be recovered from the inherent structure network by minimizing the maximum values of the direct saddle height $h[\hat{p}, \hat{q}]$ encountered along a path in (M, H) that connects $\hat{x}$ and $\hat{y}$. We remark, finally, that some studies use the term “direct saddle” to denote only the subset of direct saddles for which $h[\hat{x}, \hat{y}] = f[\hat{x}, \hat{y}]$, i.e., which cannot be circumvented by a longer but lower path in (M, H) [17,61]. Exact computation of inherent structure networks requires detection of all direct saddles and is thus restricted to small instances [66]. Efficient sampling methods exist for larger landscapes, e.g. by [38].

For the example in Fig. 3.2, the inherent structure network is the complete graph: All basins are mutually adjacent. Remarkable graph properties, however, have been revealed when studying the inherent structure networks of larger energy landscapes of chemical clusters [13], spin-glass models [7], and NK landscapes [66]. Compared to random graphs, the inherent structure networks have a large number of closed triangles (“clustering”), modular structure, and broad degree distributions, often with power law tails.

3.3.4 Funnel

Besides barrier tree and inherent structure network, another useful graph representation is the *funnel digraph* (M, A) [34]. Here, a directed arc runs from $\hat{x}$ to $\hat{y}$ if the basins of $\hat{x}$ and $\hat{y}$ have a non-empty interface and

$$h(\hat{x}, \hat{y}) = \min\{h(\hat{x}, \hat{z}) | \{\hat{x}, \hat{z}\} \in E\} \ . \tag{3.11}$$

Thus from each local minimum $\hat{x}$ an arc points to that neighboring local minimum $\hat{y}$ that is reached by the smallest direct saddle height. The node $\hat{x}$ has more than one outgoing arc if more than one neighboring basin is reached over the same minimal directed saddle height.

The definition is motivated by stochastic variants of local search such as Metropolis sampling [39] and simulated annealing [33]. Under the assumption that the search exits a basin via its lowest saddle with the largest probability, the most likely sequences of visits to basins are the directed paths of the funnel digraph. It serves to make precise the concept of a *funnel* [23, 43, 74]. A local minimum $\hat{x}$ is in the funnel $F(\hat{z}) \subseteq M$ of a global minimum $\hat{z}$ if there is a directed path from $\hat{x}$ to $\hat{z}$ on the funnel digraph. Thus the funnel contains all those local minima from which iterative exits over the lowest barrier eventually lead to the ground state [34].

In the funnel digraph for the small TSP instance in Fig. 3.2c,d, we see that the funnel consists of the basins of the global minimum itself (cost 1294) and of the highest local minimum (cost 1389) while the basins of the other three minima are outside the funnel. Numerical studies with the number partitioning problem [22] indicate that funnel size tends to zero for large random instances [34].

3.3.5 *Valleys*

Adaptive walks generalize gradient walks by accepting any fitness-improving steps. We say that $x \in X$ is *reachable* from $y \in X$ if there is an adaptive walk from y to x. A *valley* [61] is a maximal connected subgraph of X such that no point $y \notin W$ is reachable from any starting point $x \in W$. In contrast to basins, valleys do not form a hierarchy but rather can be regarded as a community structure of X, see, e.g., [20]. Instead, they overlap in the upper, high-energy, parts of the landscape, where adaptive walks are not yet committed to a unique local minimum. The parts of gradient basins below the saddle points linking them to other basins therefore are valleys. Conversely, entire gradient basins are always contained in valleys.

The exact mutual relationships among the barrier trees, basins, inherent structure networks, valleys, etc., are still not completely understood, in particular in the context of degenerate landscapes.

3.4 Elementary Landscapes

3.4.1 *Graph Laplacian*

In the previous section we have taken a geometric or topological approach to analyzing and representing a landscape. The alternative is to adopt an algebraic point of view, interpreting the landscape as a vector of fitness values. The neighborhood structure N on the set X of configurations, which index the fitness vector, also needs an algebraic interpretation. In this contribution we only deal with graphs whose edges are defined by the (symmetric) move set $N : X \to \mathfrak{P}(X)$. The most natural choice thus are the *adjacency matrix* $\mathbf{A}$ (with entries $\mathbf{A}_{xy} = 1$ if x and y are adjacent, i.e., $y \in N(x)$, and $\mathbf{A}_{xy} = 0$ otherwise) and the *incidence matrix* $\mathbf{H}$. The latter has entries $\mathbf{H}_{ex} = +1$ if x is the "head end" of the edge e, $\mathbf{H}_{ex} = -1$ if x is the "tail end" of e, and $\mathbf{H}_{ex} = 0$ otherwise. (The assignment of head and tail to each edge is arbitrary.) The basic idea of algebraic landscape theory is to explore the connections between the fitness function f and these matrix representations of the search space, and to use combinations the these algebraic object to derive numerical descriptors of landscape structure [49].

A convenient starting point is to consider the local variation of the fitness, e.g., in the form of fitness differences $f(x) - f(y)$ between adjacent vertices $y \in N(x)$. The sum of the local changes

$$(\mathbf{L}f)(x) := \sum_{y \in N(x)} (f(x) - f(y)) \tag{3.12}$$

defines the *Laplacian* $\mathbf{L}$ as a linear operator associated with the graph on (X, N) that can be applied to any function f on the nodes of the graph. When we imagine a spatially extended landscape, e.g., the underlying graph being a lattice, $(\mathbf{L}f)(x)$ can be interpreted as the local "curvature" of f at configuration x. In particular, the x-th entry of $\mathbf{L}f$ is negative if x is a local minimum and $(\mathbf{L}f)(x) > 0$ for a local maximum.

Since $\mathbf{L}$ is a linear operator, it may also be seen as a matrix. In fact, $\mathbf{L}$ is intimately related to both the adjacency and the incidence matrix: $\mathbf{L} = \mathbf{D} - \mathbf{A}$, where $\mathbf{D}$ is the diagonal matrix of vertex degrees $\deg(x) := \sum_{y \in X} \mathbf{A}_{xy}$, and $\mathbf{L} = \mathbf{H}^{+}\mathbf{H}$. The Laplacian is thus simply a representation of the topological structure of the search space.

3.4.2 Elementary Landscapes

Since $\mathbf{L}$ is a symmetric operator (on a finite-dimensional vector space), it can be diagonalized by finding its eigenfunctions. Let us perform the diagonalization for the simple and useful case that the graph is an n-dimensional hypercube with node set $X = \{-1, +1\}^n$ and $x \in N(y)$ if and only if x and y differ at exactly one coordinate. For each set of indices $I \subseteq \{1, \ldots, n\}$, the *Walsh functions* are defined as

$$w_I(x) = \prod_{i \in I} x_i \ . \tag{3.13}$$

Figure 3.4 gives an illustration for the three-dimensional case. The Walsh functions form an orthogonal basis of $\mathbb{R}^{2^n}$. With a Walsh function of order $|I|$, each node of the hypercube has $n - |I|$ neighbors with the opposite value and I neighbors with the same value. This amounts to

$$\mathbf{L}w_I = 2|I|w_I \ . \tag{3.14}$$

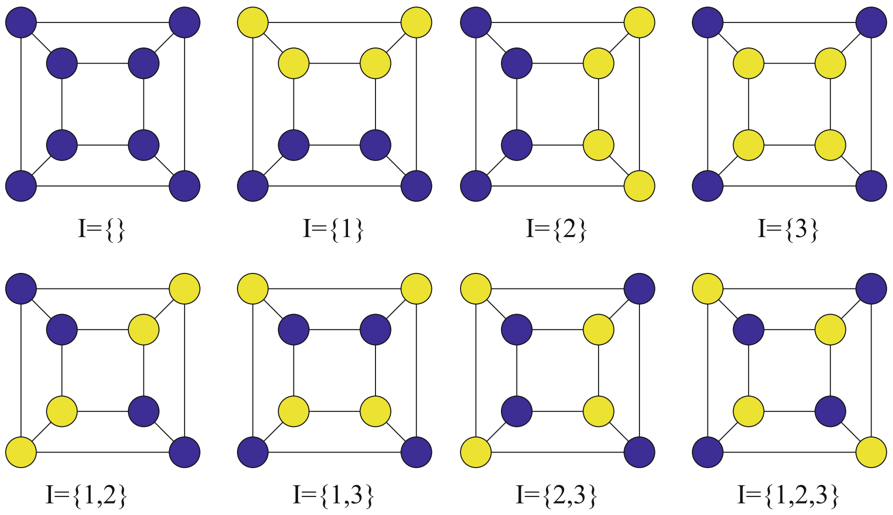


Fig. 3.4 The eight Walsh functions of the hypercube in dimension $n = 3$. *Dark* and *light color* of nodes distinguishes the values $+1$ and -1

Each Walsh function w_I is an eigenvector of $\mathbb{L}$ for eigenvalue $2|I|$. Now consider a 2-spin glass as a landscape on the hypercube with the cost function

$$f(x) = -\sum_{i,j} J_{ij} x_i x_j \tag{3.15}$$

with arbitrary real coefficients J_{ij}. Since $x_i x_j = w_{i,j}(x)$ for all $x \in X$, we see that f is a weighted sum of Walsh functions of order 2. Therefore f is an eigenvector of the Laplacian with eigenvalue $2|I| = 4$. This observation generalizes to p-spin glasses and many other landscapes of interest.

It is convenient to consider landscapes with vanishing average fitness, i.e., instead of f, we use $\tilde{f}(x) = f(x) - \bar{f}$, where $\bar{f} = \frac{1}{|X|}\sum_{x\in X} f(x)$ is the average cost of an arbitrary configuration. A landscape is *elementary* if the zero-mean cost function $\tilde{f}$ is an eigenfunction of the Laplacian of the underlying graph, i.e.,

$$(\mathbf{L}f)(x) = \sum_{y\in N(x)} \left[\tilde{f}(x) - \tilde{f}(y)\right] = \lambda_k \tilde{f}(x) \tag{3.16}$$

Since $\mathbf{L} = \mathbf{H}^{+}\mathbf{H}$, all eigenvalues are non-negative. In the following, they will be indexed in non-decreasing order

$$0 = \lambda_0 \le \lambda_1 \le \lambda_2 \le \cdots \le \lambda_{|X|-1} \, . \tag{3.17}$$

Table 3.1 Examples of elementary landscapes

Problem	Graph	degree	λ	Order	Reference
p-spin glass	$\mathcal{Q}_2^n$	n	$2p$	p	Definition
NAES	$\mathcal{Q}_2^n$	n	4	2	[24]
Weight partitioning	$\mathcal{Q}_2^n$	n	4	2	[24, 55]
GBP (constrained)	$\mathcal{Q}_2^n$	n	4	2	[1]
Max cut	$\mathcal{Q}_2^n$	n	4	2	[1]
Graph α-coloring	$\mathcal{Q}_\alpha^n$	$(\alpha-1)n$	2α	2	[55]
XY-spin glass	$\mathcal{Q}_\alpha^n$	$(\alpha-1)n$	2α	2	[21]
for $\alpha > 2$:	$\mathcal{C}_\alpha^n$	2	$8\sin^2(\pi/\alpha)$	2	[21]
Linear assignment	$\Gamma(\mathsf{S}_n, \mathcal{T})$	n		1	[51]
TSP symmetric	$\Gamma(\mathsf{S}_n, \mathcal{T})$	$n(n-1)/2$	$2(n-1)$	2	[9, 24]
	$\Gamma(\mathsf{S}_n, \mathcal{J})$	$n(n-1)/2$	n	2	[9, 24]
	$\Gamma(\mathsf{A}_n, \mathcal{C}_3)$	$n(n-1)(n-2)/6$	$(n-1)(n-2)$	?	[9]
Antisymmetric	$\Gamma(\mathsf{S}_n, \mathcal{T})$	$n(n-1)/2$	$2n$	3	[2, 55]
	$\Gamma(\mathsf{S}_n, \mathcal{J})$	$n(n-1)/2$	$n(n+1)/2$	$\mathcal{O}(n)$	[2, 55]
Graph matching	$\Gamma(\mathsf{S}_n, \mathcal{T})$	$n(n-1)/2$	$2(n-1)$	2	[55]
Graph bipartitioning	$J(n, n/2)$	$n^2/4$	$2(n-1)$	2	[24, 55, 57]

$\mathcal{Q}_\alpha^n$ is the n-fold Cartesian product of the complete graph K_α, also known as a Hamming graph. $\Gamma(\mathsf{A}, \Omega)$ is the Cayley graph of the group A with generating set Ω, where S_n and A_n denote the symmetric and alternating groups, resp., $\mathcal{T}$, $\mathcal{J}$, and $\mathcal{C}_3$ are the transpositions, reversals, and permutations defined by a cycle of length 3, resp. $J(p, q)$ is a Johnson graph.

First examples of elementary landscapes were identified by Grover and others [9, 24, 55]. Table 3.1 lists some of them. Additional examples are discussed, e.g., by [36, 53, 54, 70]. In most cases, k is small: $\tilde{f}$ lies in the eigenspace of one of the first few eigenvalues of the Laplacian.

Lov Grover [24] showed that, if f is an elementary landscape, then

$$f(\hat{x}_{\min}) \le \bar{f} \le f(\hat{x}_{\max}) \tag{3.18}$$

for every local minimum $\hat{x}_{\min}$ and every local maximum $\hat{x}_{\max}$. This *maximum principle* shows that elementary landscapes are in a sense well-behaved: There are no local optima with worse than average fitness $\bar{f}$. A bound on the global maxima in terms of the maximal local fitness differences can be obtained using a similar argument [12]:

$$\lambda \left| f(\hat{x}) - \bar{f} \right| \le \deg(\hat{x})\varepsilon^* , \tag{3.19}$$

where $\varepsilon^* = \max_{\{x,y\}\in E} |f(x) - f(y)|$ is the “information stability” introduced by [67].

The eigenvalues, λ_i, convey information about the underlying graph [42]. For instance, G is connected if and only if $\lambda_1 > 0$. The value of λ_1 is often called the algebraic connectivity of the graph G [16]. The corresponding eigenfunctions have a particularly simple form: The vertices with non-negative (non-positive) values of f are connected. More generally, a *nodal domain* of a function g on G is a maximal connected vertex set on which g does not change sign. A strong nodal domain is a maximal connected vertex set on which g has the same strict sign, $+$ or $-$. The discrete version of *Courant's nodal domain theorem* [6, 10, 15, 46] asserts that an eigenfunction f to eigenvalue λ_k (counting multiplicities) has no more than $k+1$ weak and $k+m_k$ strong nodal domains, where m_k is the multiplicity of λ_k. The theorem restricts ruggedness in terms of the eigenvalue. Intuitively, the nodal domain theorem ascertains that the number of big mountain ranges (and deep sea valleys) is small for the small eigenvalues of $\mathbf{L}$.

There is, furthermore, a simple quantitative relationship between λ and the autocorrelation function of f of $(X, \mathcal{X})$ [55]. For a D-regular graph G, we have

$$r(s) = (1 - \lambda/D)^s \tag{3.20}$$

$r(s)$ is the autocorrelation function of f along a uniform random walk on G [19, 68]. Similar equations can be derived for correlation measures defined in terms of distances on G [55] and for non-regular graphs [3, 11, 62].

Whitley et al. [71] interpret the eigenvalue equation (3.16) as a statement on the average fitness of the neighbors of a point,

$$\langle f \rangle_x = \frac{1}{\deg(x)} \sum_{y \in N(x)} f(x) \tag{3.21}$$

and discuss some implications for local search processes. Local conditions for the existence of improving moves are considered by [70].

3.4.3 Fourier Decomposition

Of course, in most cases, a natural move set $\mathcal{T}$ does not make f an elementary landscape. In case of the TSP landscape, for example, transpositions and reversals lead to elementary landscapes. One easily checks, on the other hand,

$$\begin{aligned}(\mathbf{L}f)(\pi) = \sum_{i,j} \Big\{ & d_{\pi(i)\pi(i+1)} + d_{\pi(i+1),\pi(i+2)} + d_{\pi(j)\pi(j+1)} \\ & - d_{\pi(i)\pi(i+2)} - d_{\pi(j)\pi(i+1)} - d_{\pi(i+1)\pi(j+1)} \Big\} \\ = 2n(f(\pi) - \bar{f}) + n&f(\pi) - \sum_{i=1}^{n} d_{\pi(i)\pi(i+2)}\end{aligned} \tag{3.22}$$

which is clearly not of the form $a_1 f(\pi) + a_0$ due to the explicit dependence of the last term on the next-nearest neighbors w.r.t. π. Exchange moves therefore do not make the TSP an elementary landscape.

A Fourier-decomposition-like formalism can be employed to decompose arbitrary landscapes into their elementary components [28, 51, 69]:

$$f = a_0 + \sum_{k>0}^{n-1} a_k f_k \tag{3.23}$$

where the f_k form an orthonormal system of eigenfunctions of the graph Laplacian ($\mathbf{L} f_k = \lambda_k f_k$, and $a_0 = \bar{f}$ is the average value of the function f). Let us denote the distinct eigenvalues of $\mathbf{L}$ by $\bar{\lambda}_p$, sorted in increasing order starting with $\bar{\lambda}_0 = \lambda_0 = 0$. We call p the *order* of the eigenvalue $\bar{\lambda}_p$. The *amplitude spectrum* of $f : X \to \mathbb{R}$ is defined by

$$B_p = \sum_{k:\lambda_k = \bar{\lambda}_p} |a_k|^2 \Big/ \sum_{k>0} |a_k|^2 \,. \tag{3.24}$$

By definition, $B_p \geq 0$ and $\sum_p B_p = 1$. The amplitude measures the relative contribution of the eigenspace of the eigenvalue with order p to the function f. Of course, a landscape is elementary if and only if $B_p = 1$ for a single order and 0 for all others. For Ising spin-glass models, for example, the order equals the number of interacting spins, Table 3.1.

In some cases, landscapes are not elementary but at least exhibit a highly localized spectrum. The landscape of the "Low-Autocorrelated Binary String Problem", for instance, satisfies $B_p = 0$ for $p > 4$ [44]. Quadratic assignment problems are also superpositions of eigenfunctions of quasi-abelian Cayleygraphs of the symmetric group with the few lowest eigenvalues [51]. A similar result holds for K-SAT problems. An instance of K-SAT consists of n Boolean variables x_i, and a set of m clauses each involving exactly K variables in disjunction. The cost function $f(x)$ measures the number of clauses that are satisfied. [64] showed that $B_p = 0$ for $p > K$. Similar results are available for frequency assignment problems [72], the subset sum problem [8], or genetic programming parity problems [35]. Numerical studies of the amplitude spectra of several complex real-world landscapes are reported in [4, 25, 51, 65]. A practical procedure for algebraically computing the decomposition of a landscape into its elementary constituents is discussed in [8]. This approach assumes that $f(x)$ is available as an algebraic equation and that an orthonormal basis $\{f_k\}$ is computable.

A *block-model* [70, 71] is a landscape of the form

$$f(x) = \sum_{q \in \mathcal{Q}(x)} w_q \tag{3.25}$$

where $\mathcal{Q}$ is a set of "building blocks" with weights w_q. The subset $\mathcal{Q}(x) \subseteq \mathcal{Q}$ consists of the blocks that contribute to a particular state x. For instance, the TSP is of this type: $\mathcal{Q}$ is the collection of intercity connections, and w_q is their length.

Necessary conditions for block models to be elementary are discussed, e.g., by [70]. Equation (3.25) can be rewritten in the form

$$f(x) = \sum_{q \in \mathcal{Q}} w_q \vartheta_q(x) \tag{3.26}$$

where $\vartheta_q(x) = 1$ if $q \in \mathcal{Q}(x)$, and $\vartheta_q(x) = 0$ otherwise. They form a special case of the additive landscapes considered in the next section.

3.5 Additive Random Landscapes

Many models of landscapes contain a random component. In spin-glass Hamiltonians of the general form

$$f(\boldsymbol{\sigma}) = \sum_{i_1 < i_2 < \cdots < i_p} a_{i_1, i_2, \ldots, i_p} \sigma_1 \sigma_2 \ldots \sigma_{i_p} \tag{3.27}$$

with n Ising spin variables $\sigma_i = \pm 1$, the coefficients $a_{i_1, i_2, \ldots, i_p}$ are usually considered as random variables drawn from some probability distribution. Similarly, the “thermodynamics” of TSPs can be studied with the distances w_q in Eq. (3.25) interpreted as random variables. [48,58] studied this type of model in a more general setting motivated by Eq. (3.26).

Let $\vartheta_q : X \to \mathbb{R}$, $q \in I$, be a family of fitness functions on X, where I is some index set, and let c_q, $q \in I$, be independent, real-valued, random variables. We consider additive random fitness functions of the form

$$f(x) = \sum_{q \in I} c_q \vartheta_q(x) \tag{3.28}$$

on $G = (X, \mathcal{T})$. The associated probability space is referred to *additive random landscape* (a.r.l.) [48]. An a.r.l. is *uniform* if (1) the c_q, $q \in I$, are independently and identically distributed (i.i.d.) and (2) there are constants b_0 and b_1 such that $\sum_{x \in V} \vartheta_q(x) = |V| b_0$ and $\sum_{x \in V} \vartheta_q(x)^2 = |V| b_1$ independent of q. An a.r.l. is *strictly uniform* if, in addition, $\sum_q \vartheta_q(x) = b_2$ and $\sum_q \vartheta_q(x)^2 = b_3$ independently of $x \in X$.

For block models, $\vartheta_q(x) = \vartheta_q(x)^2$. Hence a block model is uniform if each block is contained in the same number configurations and is strictly uniform if in addition, $|\mathcal{Q}(x)|$ is independent of x. Furthermore, every a.r.l. with Gaussian measure is additive. This follows immediately by the Karhunen-Loève theorem [30], using the fact that uncorrelated jointly normal distributed variables are already independent. Random versions of elementary landscapes, i.e., those with i.i.d. Fourier coefficients, are of course also a.r.l.s.

Maybe the most famous a.r.l.s are Kauffman's NK landscape models. Consider a binary "genome" x of length n. Each site is associated with a site fitness that is determined from a set of K sequence positions to which position i is epistatically linked. These state-dependent contributions are usually taken as independent uniform random variables [31]. In order to see that an NK model is an a.r.l., we introduce the functions

$$\vartheta_{i,y(i)}(x) = \prod_{k=0}^{|y(i)|} \delta_{y(i)_k, x_k} \tag{3.29}$$

where $y(i)_k$ denotes a particular (binary) state of position k in the epistatic neighborhood $y(i)$ of i. In other words, $\vartheta_{i,y(i)}(x) = 1$ if the argument x coincides on the K epistatic neighbors of i with a particular binary pattern $y(i)$. It is not hard to check that NK models are strictly uniform [48].

An important observation in this context is that short-range spin-glass models can be understood as a.r.l.s for which the p-spin eigenfunctions (of the Laplacian of the Boolean hypercube)

$$\vartheta_{i_1 i_2 \ldots i_p}(\boldsymbol{\sigma}) = \prod_{k=1}^{p} \sigma_{i_k} \tag{3.30}$$

take on the role of the characteristic functions $\{\vartheta_j\}$. The coefficients are then taken from a mixed distribution of the form $\rho(c) = (1-\mu)\text{Gauss}_{(0,s)}(c) + \mu\delta(c)$. Thus there is a finite probability p that a coefficient c_j vanishes. In this setting one can easily evaluate the *degree of neutrality*, i.e., the expected fraction of neutral neighbors

$$\nu(x) = \frac{1}{|N(x)|}\mathbb{E}\left[\left|\{y \in N(x) : f(x) = f(y)\}\right|\right] \tag{3.31}$$

One finds

$$\nu(x) = \frac{1}{D} \sum_{y \in N(x)} \mu^{c_y(x)} \qquad c_y(x) := |\{q \in I \,|\, \vartheta_q(x) \neq \vartheta_q(y)\}| \tag{3.32}$$

which, for the case of the p-spin models can be evaluated explicitly as $\nu(x) = \mu^{\binom{n-1}{p-1}}$ [48]. The value of ν can thus be tuned by μ, the fraction of vanishing coefficients. On the other hand, the Laplacian eigenvalue and hence the algebraic measures of ruggedness depend only on the interaction order p. Hence, ruggedness and neutrality are *independent* properties of landscapes.

Elementary landscapes have restrictions on "plateaus", that is, on connected subgraphs on which the landscape is flat. In particular, Sutton et al. [64] show for 3-SAT that plateaus cannot contain large balls in G unless their fitness is close to average. A more general understanding of neutrality in elementary landscapes is still missing, however.

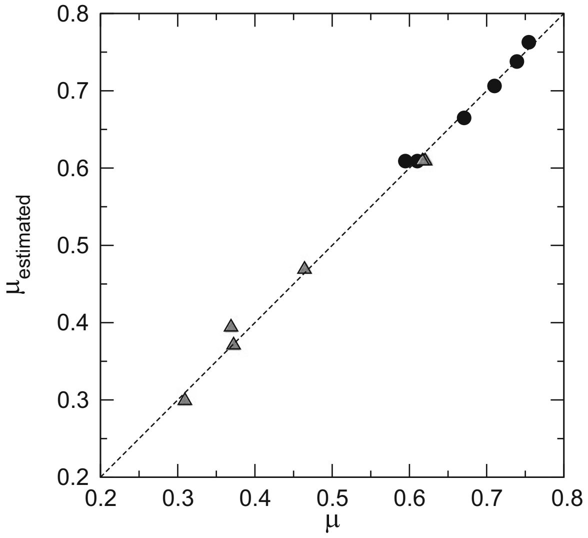


Fig. 3.5 Empirically, the number $\mathcal{N}$ of local optima in many landscape scales exponentially with system size n. The values of $\mu := \lim_{n\to\infty} \frac{1}{n} \log \mathcal{N}(n)$ can be estimated surprisingly accurately by the correlation length conjecture for *-isotropic landscapes (●) and nearly *-isotropic landscapes (▲) (Data are taken from [58])

There is a close connection between the Fourier decomposition of a (random) landscape and a "symmetry" property of random fields: Stadler and Happel [58] call a random field **-isotropic* if its covariance matrix $\mathbf{C}$ is a polynomial of the adjacency matrix $\mathbf{A}$ of the underlying graph. The interest in this symmetry property arises from the observation that *-isotropy is equivalent to three regularity conditions on the distributions of the Fourier coefficients:

1. $\mathbb{E}[a_k] = 0$ for $k \neq 0$.
2. $\mathbb{Cov}[a_k, a_j] := \mathbb{E}[a_k a_j] - \mathbb{E}[a_k]\mathbb{E}[a_j] = \mathbb{Var}[a_k]\delta_{kj}$.
3. $\mathbb{Var}[a_k] = \mathbb{Var}[a_j]$ if $\varphi_k(x)$ and $\varphi_j(x)$ are eigenfunctions to the same eigenvalue of the graph Laplacian.

In Gaussian case, *-isotropy also has an interpretation as a maximum entropy condition: Gaussian random fields satisfying these three conditions maximize entropy subject to the constraint of a given amplitude spectrum [56].

An interesting empirical observation in this context is the *correlation length conjecture* [59]. It states that we should expect about one local optimum within a ball B in G whose radius r is given by the distance covered by random walk of length ℓ on G, where

$$\ell = \sum_{s=0}^{\infty} r(s) \tag{3.33}$$

is the *correlation length* of the landscape. This simple rule yields surprisingly accurate estimates of the number of local optima in isotropic and nearly isotropic landscapes, see Fig. 3.5. The accuracy of the estimate seems to decline with increasing deviations from *-isotropy [21, 44, 58].

3.6 Outlook

Most of the theoretical results for fitness landscapes have been obtained for very simple search processes, i.e., for landscapes on simple graphs. In the field of genetic algorithms, on the other hand, the search operators themselves are typically much more complex, involving recombination of pairs of individuals drawn from a population. A few attempts have been made to elucidate the mathematical structure of landscapes on more general topologies [18, 60, 61], considering generalized versions of barrier trees and basins. A generalization of the Fourier decomposition and a notion of elementary landscapes under recombination was explored in [62], introducing a Markov chain on X that in a certain sense mimics the action of crossover. In this formalism, a close connection between elementary landscapes and Holland's schemata [27] becomes apparent. It remains a topic for future research, however, if and how spectral properties of landscapes can be formulated in general for population-based search operators in which offsprings are created from more than one parent.

Despite the tentative computational results that suggest a tight connection between spectral properties of a landscape, statistical features of the Fourier coefficients, and geometric properties such as the distribution of local minima, there is at present no coherent theory that captures these connections. We suspect that a deeper understanding of relations between these different aspects of landscapes will be a prerequisite for any predictive theory of the performance of optimization algorithms on particular landscape problems.

References

1. E. Angel, V. Zissimopoulos, On the classification of NP-complete problems in terms of their correlation coefficient. Discr. Appl. Math. **99**, 261–277 (2000)
2. J. Barnes, S. Dokov, R. Acevedoa, A. Solomon, A note on distance matrices yielding elementary landscapes for the TSP. J. Math. Chem. **31**, 233–235 (2002)
3. J.W. Barnes, B. Dimova, S.P. Dokov, A. Solomon, The theory of elementary landscapes. Appl. Math. Lett. **16**, 337–343 (2003)
4. O. Bastert, D. Rockmore, P.F. Stadler, G. Tinhofer, Landscapes on spaces of trees. Appl. Math. Comput. **131**, 439–459 (2002)
5. K. Binder, A.P. Young, Spin glasses: experimental facts, theoretical concepts, and open questions. Rev. Mod. Phys. **58**, 801–976 (1986)
6. T. Bıyıkoğlu, J. Leydold, P.F. Stadler, in *Laplacian Eigenvectors of Graphs: Perron-Frobenius and Faber-Krahn Type Theorems*. Lecture Notes in Mathematics, vol. 1915 (Springer, Heidelberg, 2007)
7. Z. Burda, A. Krzywicki, O.C. Martin, Network of inherent structures in spin glasses: scaling and scale-free distributions. Phys. Rev. E **76**, 051107 (2007)
8. F. Chicano, L.D. Whitley, E. Alba, A methodology to find the elementary landscape decomposition of combinatorial optimization problems. Evol. Comp. (2011). doi:10.1162/EVCO_a_00039
9. B. Codenotti, L. Margara, Local properties of some NP-complete problems. Technical Report TR 92-021, International Computer Science Institute, Berkeley, 1992

10. E.B. Davies, G.M.L. Gladwell, J. Leydold, P.F. Stadler, Discrete nodal domain theorems. Lin. Algebra Appl. **336**, 51–60 (2001)
11. B. Dimova, J.W. Barnes, E. Popova, Arbitrary elementary landscapes & AR(1) processes. Appl. Math. Lett. **18**, 287–292 (2005)
12. B. Dimova, J.W. Barnes, E. Popova, E. Colletti, Some additional properties of elementary landscapes. Appl. Math. Lett. **22**, 232–235 (2009)
13. J.P.K. Doye, Network topology of a potential energy landscape: a static scale-free network. Phys. Rev. Lett. **88**, 238701 (2002)
14. G. Dueck, New optimization heuristics: the great deluge algorithm and the record-to-record travel. J. Comp. Phys. **104**, 86–92 (1993)
15. A.M. Duval, V. Reiner, Perron-Frobenius type results and discrete versions of nodal domain theorems. Lin. Algebra Appl. **294**, 259–268 (1999)
16. M. Fiedler, Algebraic connectivity of graphs. Czechoslovak Math. J. **23**, 298–305 (1973)
17. C. Flamm, I.L. Hofacker, P.F. Stadler, M.T. Wolfinger, Barrier trees of degenerate landscapes. Z. Phys. Chem. **216**, 155–173 (2002)
18. C. Flamm, B.M.R. Stadler, P.F. Stadler, Saddles and barrier in landscapes of generalized search operators, in *Foundations of Genetic Algorithms IX*, ed. by C.R. Stephens, M. Toussaint, D. Whitley, P.F. Stadler. Lecture Notes Computer Science, vol. 4436 (Springer, Berlin/Heidelberg, 2007), pp. 194–212. 9th International Workshop, FOGA 2007, Mexico City, 8–11 Jan 2007
19. W. Fontana, P.F. Stadler, E.G. Bornberg-Bauer, T. Griesmacher, I.L. Hofacker, M. Tacker, P. Tarazona, E.D. Weinberger, P. Schuster, RNA folding landscapes and combinatory landscapes. Phys. Rev. E **47**, 2083–2099 (1993)
20. S. Fortunato, Community detection in graphs. Phys. Rep. **486**, 75–174 (2010)
21. R. García-Pelayo, P.F. Stadler, Correlation length, isotropy, and meta-stable states. Physica D **107**, 240–254 (1997)
22. M.R. Garey, D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (W.H. Freeman, San Francisco, 1979)
23. P. Garstecki, T.X. Hoang, M. Cieplak, Energy landscapes, supergraphs, and “folding funnels” in spin systems. Phys. Rev. E **60**, 3219–3226 (1999)
24. L.K. Grover, Local search and the local structure of NP-complete problems. Oper. Res. Lett. **12**, 235–243 (1992)
25. R. Happel, P.F. Stadler, Canonical approximation of fitness landscapes. Complexity **2**, 53–58 (1996)
26. D. Heidrich, W. Kliesch, W. Quapp, in *Properties of Chemically Interesting Potential Energy Surfaces*. Lecture Notes in Chemistry, vol. 56 (Springer, Berlin, 1991)
27. J. Holland, *Adaptation in Natural and Artificial Systems* (MIT, Cambridge, 1975)
28. W. Hordijk, P.F. Stadler, Amplitude spectra of fitness landscapes. Adv. Complex Syst. **1**, 39–66 (1998)
29. L. Kallel, B. Naudts, C.R. Reeves, Properties of fitness functions and search landscapes, in *Theoretical Aspects of Evolutionary Computing*, ed. by L. Kallel, B. Naudts, A. Rogers (Springer, Berlin Heidelberg, 2001), pp. 175–206
30. K. Karhunen, Zur Spektraltheorie Stochasticher Prozesse. Ann. Acad. Sci. Fennicae, Ser. A I **34**, 7 (1947)
31. S.A. Kauffman, *The Origin of Order* (Oxford University Press, New York/Oxford, 1993)
32. S.A. Kauffman, S. Levin, Towards a general theory of adaptive walks on rugged landscapes. J. Theor. Biol. **128**, 11–45 (1987)
33. S. Kirkpatrick, C.D. Gelatt Jr., M.P. Vecchi, Optimization by simulated annealing. Science **220**, 671–680 (1983)
34. K. Klemm, C. Flamm, P.F. Stadler, Funnels in energy landscapes. Europ. Phys. J. B **63**, 387–391 (2008)
35. W.B. Langdon, 2-bit flip mutation elementary fitness landscapes, in *11th International Workshop on Foundations of Genetic Algorithms, FOGA 2011*, Schwarzenberg, ed. by H.G. Beyer, W.B. Langdon (ACM, 2011), pp. 25–42

36. G. Lu, R. Bahsoon, X. Yao, Applying elementary landscape analysis to search-based software engineering, in *2nd International Symposium on Search Based Software Engineering*, Benevento (IEEE Computer Society, Los Alamitos, 2010), pp. 3–8
37. C.A. Macken, P.S. Hagan, A.S. Perelson, Evolutionary walks on rugged landscapes. SIAM J. Appl. Math. **51**, 799–827 (1991)
38. M. Mann, K. Klemm, Efficient exploration of discrete energy landscapes. Phys. Rev. E **83**(1), 011113 (2011)
39. N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, Equation of state calculations by fast computing machines. J. Chem. Phys. **21**, 1087–1092 (1953)
40. M. Mézard, G. Parisi, M. Virasoro, *Spin Glass Theory and Beyond* (World Scientific, Singapore, 1987)
41. P.G. Mezey, *Potential Energy Hypersurfaces* (Elsevier, Amsterdam, 1987)
42. B. Mohar, Graph laplacians, in *Topics in Algebraic Graph Theory, Encyclopedia of Mathematics and Its Applications*, vol. 102, ed. by L.W. Beineke, R.J. Wilson (Cambridge University Press, Cambridge, 2004), pp. 113–136
43. K.i. Okazaki, N. Koga, S. Takada, J.N. Onuchic, P.G. Wolynes, Multiple-basin energy landscapes for large-amplitude conformational motions of proteins: structure-based molecular dynamics simulations. Proc. Natl. Acad. Sci. U.S.A. **103**, 11844–11849 (2006)
44. V.M. de Oliveira, J.F. Fontanari, P.F. Stadler, Metastable states in high order short-range spin glasses. J. Phys. A: Math. Gen. **32**, 8793–8802 (1999)
45. R. Palmer, Optimization on rugged landscapes, in *Molecular Evolution on Rugged Landscapes: Proteins, RNA, and the Immune System*, ed. by A.S. Perelson, S.A. Kauffman (Addison-Wesley, Redwood City, 1991), pp. 3–25
46. D.L. Powers, Graph partitioning by eigenvectors. Lin. Algebra Appl. **101**, 121–133 (1988)
47. R. Rammal, G. Toulouse, M.A. Virasoro, Ultrametricity for physicists. Rev. Mod. Phys. **58**, 765–788 (1986)
48. C.M. Reidys, P.F. Stadler, Neutrality in fitness landscapes. Appl. Math. Comput. **117**, 321–350 (2001)
49. C.M. Reidys, P.F. Stadler, Combinatorial landscapes. SIAM Rev. **44**, 3–54 (2002)
50. H. Rieger, The number of solutions of the Thouless-Anderson-Palmer equations for p-spin interaction spin glasses. Phys. Rev. B **46**, 14655–14661 (1992)
51. D. Rockmore, P. Kostelec, W. Hordijk, P.F. Stadler, Fast Fourier transform for fitness landscapes. Appl. Comput. Harmonic Anal. **12**, 57–76 (2002)
52. P. Sibani, R. van der Pas, J.C. Schön, The lid method for exhaustive exploration of metastable states of complex systems. Comput. Phys. Commun. **116**, 17–27 (1999)
53. A. Solomon, J.W. Barnes, S.P. Dokov, R. Acevedo, Weakly symmetric graphs, elementary landscapes, and the TSP. Appl. Math. Lett. **16**, 401–407 (2003)
54. A. Solomon, B.W. Colletti, Quasiabelian landscapes of the traveling salesman problem are elementary. Discret. Optim. **6**, 288–291 (2009)
55. P.F. Stadler, Landscapes and their correlation functions. J. Math. Chem. **20**, 1–45 (1996)
56. P.F. Stadler, Spectral landscape theory, in *Evolutionary Dynamics—Exploring the Interplay of Selection, Neutrality, Accident, and Function*, ed. by J.P. Crutchfield, P. Schuster (Oxford University Press, New York, 2002), pp. 231–272
57. P.F. Stadler, R. Happel, Correlation structure of the landscape of the graph-bipartitioning-problem. J. Phys. A: Math. Gen. **25**, 3103–3110 (1992)
58. P.F. Stadler, R. Happel, Random field models for fitness landscapes. J. Math. Biol. **38**, 435–478 (1999)
59. P.F. Stadler, W. Schnabl, The landscape of the travelling salesman problem. Phys. Lett. A **161**, 337–344 (1992)
60. B.M.R. Stadler, P.F. Stadler, Generalized topological spaces in evolutionary theory and combinatorial chemistry. J. Chem. Inf. Comput. Sci. **42**, 577–585 (2002)
61. B.M.R. Stadler, P.F. Stadler, Combinatorial vector fields and the valley structure of fitness landscapes. J. Math. Biol. **61**, 877–898 (2010)

62. P.F. Stadler, R. Seitz, G.P. Wagner, Evolvability of complex characters: population dependent Fourier decomposition of fitness landscapes over recombination spaces. Bull. Math. Biol. **62**, 399–428 (2000). Santa Fe Institute Preprint 99-01-001
63. P.F. Stadler, W. Hordijk, J.F. Fontanari, Phase transition and landscape statistics of the number partitioning problem. Phys. Rev. E **67**, 0567011–6 (2003)
64. A.M. Sutton, A.E. Howe, L.D. Whitley, A theoretical analysis of the k-satisfiability search space, in *Proceedings of SLS 2009*, Brussels. Lecture Notes in Computer Science, vol. 5752 (2009), pp. 46–60
65. A.M. Sutton, L.D. Whitley, A.E. Howe, A polynomial time computation of the exact correlation structure of k-satisfiability landscapes, in *Genetic and Evolutionary Computation Conference, GECCO 2009*, Montréal, 2009, ed. by F. Rothlauf, pp. 365–372
66. M. Tomassini, S. Vérel, G. Ochoa, Complex-network analysis of combinatorial spaces: the NK landscape case. Phys. Rev. E **78**, 066114 (2008)
67. V.K. Vassilev, T.C. Fogarty, J.F. Miller, Information characteristics and the structure of landscape. Evol. Comput. **8**, 31–60 (2000)
68. E.D. Weinberger, Correlated and uncorrelated fitness landscapes and how to tell the difference. Biol. Cybern. **63**, 325–336 (1990)
69. E.D. Weinberger, Local properties of Kauffman's N-K model: a tunably rugged energy landscape. Phys. Rev. A **44**, 6399–6413 (1991)
70. L.D. Whitley, A.M. Sutton, Partial neighborhoods of elementary landscapes, in *Genetic and Evolutionary Computation Conference, GECCO 2009*, Montréal, 2009, ed. by F. Rothlauf, pp. 381–388
71. L.D. Whitley, A.M. Sutton, A.E. Howe, Understanding elementary landscapes, in *Genetic and Evolutionary Computation Conference, GECCO 2008*, Atlanta, ed. by C. Ryan, M. Keijzer (ACM, 2008), pp. 585–592
72. L.D. Whitley, F. Chicano, E. Alba, F. Luna, Elementary landscapes of frequency assignment problems, in *Proceedings of the 12th Annual Conference of Genetic and Evolutionary Computation GECCO*, Portland, ed. by M. Pelikan, J. Branke (ACM, 2010), pp. 1409–1416
73. M.T. Wolfinger, W.A. Svrcek-Seiler, C. Flamm, I.L. Hofacker, P.F. Stadler, Exact folding dynamics of RNA secondary structures. J. Phys. A: Math. Gen. **37**, 4731–4741 (2004)
74. P. Wolynes, J. Onuchic, D. Thirumalai, Navigating the folding routes. Science **267**, 1619–1620 (1995)
75. S. Wright, The roles of mutation, inbreeding, crossbreeeding and selection in evolution, in *Proceedings of the Sixth International Congress on Genetics*, New York, vol. 1, ed. by D.F. Jones (Brooklyn Botanic Gardens, New York, 1932), pp. 356–366
76. S. Wright, "Surfaces" of selective value. Proc. Natl. Acad. Sci. U.S.A. **58**, 165–172 (1967)

Chapter 4
Single-Funnel and Multi-funnel Landscapes and Subthreshold-Seeking Behavior

Darrell Whitley and Jonathan Rowe

Abstract Algorithms for parameter optimization display subthreshold-seeking behavior when the majority of the points that the algorithm samples have an evaluation less than some target threshold. Subthreshold-seeking algorithms avoid the curse of the general and Sharpened No Free Lunch theorems in the sense that they are better than random enumeration on a specific (but general) family of functions. In order for subthreshold-seeking search to be possible, most of the solutions that are below threshold must be localized in one or more regions of the search space. Functions with search landscapes that can be characterized as single-funnel or multi-funnel landscapes have this localized property. We first analyze a simple "Subthreshold-Seeker" algorithm. Further theoretical analysis details conditions that would allow a Hamming neighborhood local search algorithm using a Gray or binary representation to display subthreshold-seeking behavior. A very simple modification to local search is proposed that improves its subthreshold-seeking behavior.

4.1 Background and Motivation

When can we say that a search algorithm is robust? And what does this mean? The "Sharpened No Free Lunch" results tell us that no search algorithm is on average better than another over all possible discrete functions, or over any set of functions closed under permutation [12].

D. Whitley (✉)
Department of Computer Science, Colorado State University, Fort Collins, CO 80523, USA
e-mail: whitley@cs.colostate.edu

J. Rowe
Department of Computer Science, University of Birmingham, Birmingham, B15 2TT, UK
e-mail: J.E.Rowe@cs.bham.ac.uk

One way that a search algorithm might be "robust" is if we could show that the search algorithm is guaranteed to yield results that are guaranteed to be better than random search on a broad class of problems. This paper illustrates how this can be done using "subthreshold-seeking" search strategies. Assume we are minimizing an objective function. By subthreshold-seeking search, we mean that a performance threshold can be established and that the majority of the time search will sample points in the search space that are below the established threshold. The concept of "submedian seeking" behavior was first introduced by Christensen and Oppacher [4]. Whitley and Rowe [16] generalized this idea to address general subthreshold-seeking search strategies.

Some of the results presented in this chapter are exact, but are largely of theoretical interest. Other results are less exact; instead of guaranteeing subthreshold-seeking behavior, the result describe conditions under which subthreshold-seeking behavior can occur in realistic search algorithms.

Clearly, a search algorithm cannot be said to be robust, or to display subthreshold-seeking behavior over all possible functions. But what classes of functions should we expect to be able to optimize? Again, Christensen and Oppacher [4] first introduced the idea that functions that can be described by polynomials of bounded complexity form a general class of functions where search algorithms can be designed that are guaranteed to yield results better than random enumeration. Christensen and Oppacher [4] also point out that another way to define a restricted subclass of functions is to limit the number of local optima in the search space relative to the size of the search space.

We can take this idea one step further. If we are interested in subthreshold points in the search space, then what matters is not the number of local optima; instead, search can be robust when the points which are subthreshold tend to be clustered together, so that search more often than not can move from one subthreshold point in the search space to another subthreshold point in the search space. We introduce the concept of "quasi-basin" to describe this condition. This also naturally leads to considering single-funnel and multi-funnel functions and landscapes, since the concept of a "funnel" is related to the concept of a quasi-basin.

Section 4.2 of this chapter discusses single-funnel and multi-funnel functions and landscapes. Section 4.3 briefly reviews No Free Lunch concepts and how they relate to subthreshold-seeking behaviors. Section 4.4 then introduces subthreshold-seeking algorithms and describes conditions where algorithms can display subthreshold-seeking behavior. The subthreshold-seeking algorithm (like Christensen and Oppacher's submedian-seeking algorithm) is guaranteed to display subthreshold sampling behavior, but the algorithm is not practical. So, can we say anything about a real search algorithm? In Sect. 4.5, we look at a local search algorithm using a bit representation, in part because this is one of the most general-purpose search methods that can be used for both parameter optimization and combinatorial optimization. We then outline sufficient conditions that would allow local search to display subthreshold-seeking behavior. We also introduce a Subthreshold Local Search Algorithm and show that is can be superior to a simple local search with restarts.

4.2 Single-Funnel and Multi-funnel Functions

Over the years, several binary classifications of test functions have been used to make the argument that one set of test functions is more challenging than another, and thus to argue that better performance on a particular type of test problem translates into one algorithm being better than another. Here are a few of these classifications:

1. Linear versus Nonlinear
2. Unimodal versus Multimodal
3. Separable versus Nonseparable
4. Single-Funnel versus Multi-funnel

Generally, multimodal problems are harder than unimodal problems, but this is not always true. Generally, nonlinear problems are harder than linear problems. Nonseparable problems are usually harder than separable problems. Yet despite empirically backed claims that one algorithm is better than another on a particular type of problem, there are rarely proofs that a specific algorithm performs better than another algorithm (or even better than random search) on a specific family of functions.

The characterization of optimization problems as *single-funnel* versus *multi-funnel functions* is one of the more recent descriptive models that attempts to explain why some optimization problems are harder than others.

We will sometimes need to distinguish between a function and the landscape that is searched. When real-valued representations are used, the function and the landscape might be viewed as being (approximately) the same. But for local search algorithms that might be applied to combinatorial optimization problems, the landscape can change depending on what type of representation is used, what neighborhood move operator is used and, if the problem is a discretization of a real-valued parameter problem, what precision is used.

Unfortunately, there does not appear to be a concise definition of a "single-funnel" landscape. Clearly unimodal landscapes are also single-funnel landscapes. But single-funnel problems also can be highly multimodal; nevertheless in a single-funnel landscape there is still some kind of global structure that dominates the entire search space and causes the best local optima to be concentrated in one region of the search space. Such landscapes are consistent with the big valley hypothesis [2, 3].

In rough terms, the big valley hypothesis states that there exist landscapes that are multimodal, but that these landscapes have the property that the evaluation of a local optimum is strongly correlated with the distance of that local optimum from the global optimum. In other words, the closer a local optimum is to the global optimum, the better the evaluation of the local optimum is going to be in expectation. The Traveling Salesman Problem under k-opt move operators is often given as an example of a problem that displays a big valley landscape.

A "multi-funnel" landscape is one where there are multiple clusters of local optima in very different regions of the search space; in other words, clusters corresponding to the best local optima in the search space can be far apart and

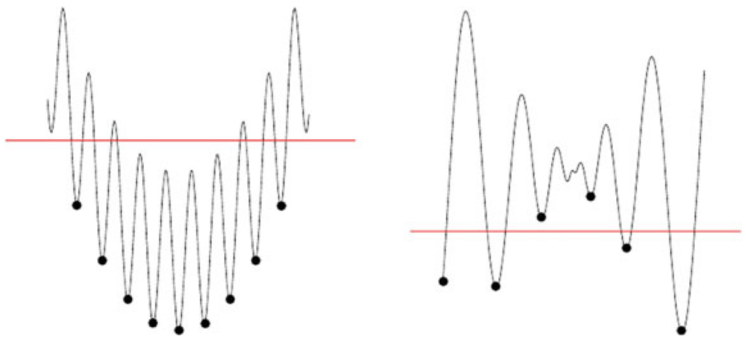


Fig. 4.1 On the *left* is a one-dimensional version of Rastrigin's function. On the *right* is a one-dimensional version of Schwefel's function. A threshold is shown for each function

are not concentrated in one region in the search space. Therefore, multi-funnel landscapes can be difficult to search for algorithms that use some form of localized non-improving move (that moves only a short distance) in an attempt to escape local optima.

Doye used the concept of a "double-funnel" energy landscape in 1999 to explain why finding the optimal energy state for the 38-atom Lennard-Jones cluster was difficult [6]. Doye et al. [7] indicate that a *funnel* "consists of a collection of local minima such that all monotonically decreasing sequences of successively adjacent local minima entering the funnel terminate at the funnel bottom." And the concept of "Multi-funnel Optimization" is becoming more common [10, 14].

If one considers parameter optimization problems, then Rastrigin's function, shown in the left-hand side of Fig. 4.1, is an excellent example of a single funnel function; it also displays a big valley structure. Again considering parameter optimization problems, then Schwefel's function, shown in the right-hand side of Fig. 4.1, is an example of a multi-funnel function. (We should note that what is usually called "Schwefel's function" is technically a variant of Problem 2.3 in Schwefel's classic book, *Evolution and Optimum Seeking* [13], which contains several test functions.) On both functions in Fig. 4.1, a *threshold* has been established that divides each function into two sets: those points in the search space with an evaluation below some threshold codomain value and those points in the search below above the threshold codomain value. We will assume in these examples that a real-valued representation is used so that a local extremum in the function is also a local optimum in the landscape.

For Rastrigin's function the points that are below threshold are largely concentrated in one region of the function. If the threshold is progressively lowered, the points below threshold would be increasingly concentrated in a smaller region of the search space. This is characteristic of a single-funnel function. A set of progressively lower thresholds and the local optima that are isolated at lower thresholds is shown in Fig. 4.2 for Rastrigin's function.

For Schwefel's function the points that are below threshold are *not* concentrated in a single region of the function. If the threshold is progressively lowered, the points

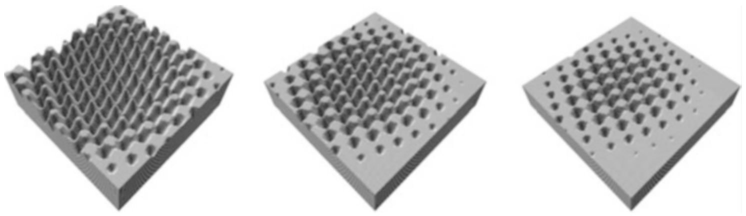


Fig. 4.2 The graph shows two-dimensional slices of Rastrigin's function. The function is clipped so that only points below a threshold are evaluated. As the threshold is lowered, the points that are below threshold are increasingly concentrated in one region of the search space, so that the average distance between the below threshold points decreases

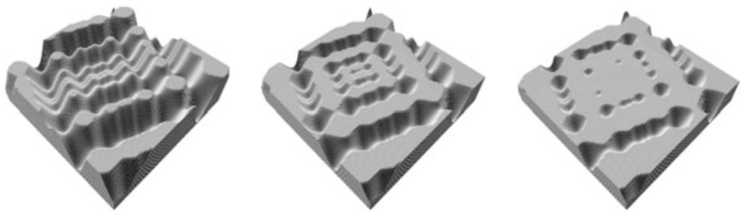


Fig. 4.3 The graph shows two-dimensional slices of Schwefel's function. The function is clipped so that only points below a threshold are evaluated. As the threshold is lowered, the points that are below threshold continue to be at the same (or even slightly increasing) average distance from each other

below threshold remain distributed across the entire search space. The points are distributed across multiple "funnels" and these funnels generally continue to exist as the threshold is lowered. Only when the threshold becomes sufficiently close to the global optimum do the other funnels in the search space suddenly disappear. A set of progressively lower thresholds and the local optima that are isolated at lower thresholds is shown in Fig. 4.3 for Schwefel's function.

In reality, the set of all possible landscapes cannot be neatly classified into single-funnel and multi-funnel landscapes. Lunacek and Whitley introduced a metric that measures the *dispersion* of a function [9]. Single-funnel landscapes would display low dispersion, while landscapes with many funnels spread across the entire search space would display high dispersion. But a simple "double-funnel" might display an intermediate level of dispersion. Let $d(x_1, x_2)$ be a distance measure that returns the distance between points x_1 and x_2. The dispersion metric also uses a threshold. Consider all the points in a set T such that the points in the domain in T are below a threshold evaluation denoted by t.

We will initially consider real-valued functions. Given a function f, the threshold t and the distance metric d, we can then compute the following measurement of the *dispersion* of the function:

$$\sum_{x_i, x_j \in T} d(x_i, x_j)$$

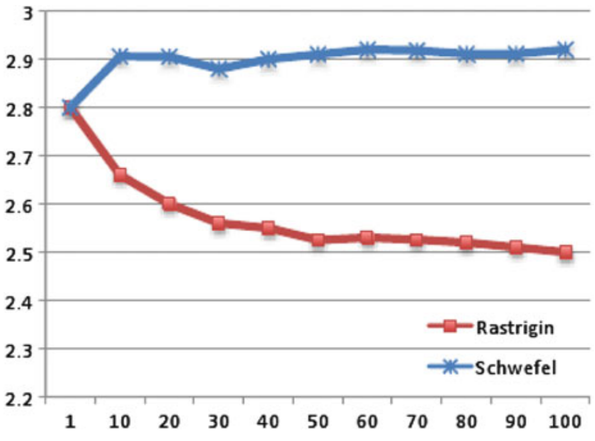


Fig. 4.4 A graph showing the dispersion metric at various thresholds. On the x-axis is the sample size when N represents $100N$ samples (i.e., 10 = 1,000 samples). On the y-axis is the pairwise distance between the best 100 sample points. The domain values are drawn between -5.11 and $+5.12$. The problems from which these measurements were taken were 50 dimensional

where $d(x_i, x_j)$ is a measurement of the distance between points x_i, x_j – two points in the search space. The summation measures the pairwise distance between all points in the subthreshold set T. Assuming we wish to minimize, we could then check the dispersion at different thresholds: $t, t-x, t-2x, \ldots, t-kx$. Assuming the dispersion metric is decreasing as the threshold decreases, this would indicate that the landscape for function f displays properties associated with single-funnel landscapes. However, if the dispersion metric is relatively constant or increasing, this would strongly characterize the landscape as either a multi-funnel landscape, or perhaps a more randomized landscape where local optima are randomly distributed over the search space.

One nice aspect of the dispersion metric is that the dispersion of a function can be estimated via random sampling. Figure 4.4 graphs dispersion measurements at different thresholds on Rastrigin's function and Schwefel's function. The x-axis is the number of points that are randomly sampled. The y-axis is the dispersion over the best 100 points out of the total number sampled. For example, when dispersion is measured over 100 of 1,000 points, the dispersion is being measured over the best 10 % of the sample. If dispersion is measured over the 100 of 10,000 points, then dispersion is being measured over the best 1 % of the sample. As the sample percentage is decreased (10, …, 5, …, 1 %) we are in effect sampling at lower thresholds. As Fig. 4.4 clearly shows, the trend is that as Rastrigin's function is sampled at lower thresholds, the dispersion decreases, but as Schwefel's function is sampled at lower thresholds, the dispersion increases. This is consistent with the fact that Rastrigin's function has a single funnel and Schwefel's function is multi-funnel. The trend need not be monotonic for two reasons. First, sampling is a noisy process. Second, local minima can disappear from view as the threshold is lowered and this can cause small fluctuations in the dispersion measurements.

One can also generalize the concept of dispersion to combinatorial landscapes. To do this, some natural concept of distance between local optima is needed. Indeed, in "big valley" studies sometimes pairwise distance between local optima is used instead of measuring the distance between a sample of local optima and the distance between those local optima and the global optimum.

4.3 No Free Lunch and Subthreshold-Seeking Behavior

So what do No Free Lunch theorems and multi-funnel functions have to do with subthreshold-seeking behavior? We will first briefly review No Free Lunch concepts to set the stage for discussing subthreshold-seeking algorithms.

4.3.1 No Free Lunch and Funnels

The original No Free Lunch theorem of Wolpert and Macready [20, 21] is quite general. The "No Free Lunch" assertion is basically a zero-sum argument that states that no search algorithm is better than another when the performance of the search algorithms are compared over all possible discrete functions.

In this form, No Free Lunch applies only to black box optimization. This means the structure of the function being optimized is invisible to the search algorithm. Also, since the theorem holds over the space of all possible functions, it is impossible to predict anything about the structure of the function being optimized.

No Free Lunch theorems for search have been refined in the last decade to be more specific, but there continues to be confusion about what different variants of the No Free Lunch theorem actually mean. The "Sharpened No Free Lunch" result shows that a general No Free Lunch result only holds over sets of functions that are closed under permutation. We might express the "Sharpened No Free Lunch" theorem as follows:

> The behaviors of any two arbitrarily chosen search algorithms are guaranteed to be equivalent if and only if the algorithms are compared on a set of functions that is closed under permutation.

This version of "Sharpened No Free Lunch" is more carefully stated than is usually the case. But the wording here is important. What if we wish to compare two specific search algorithms, A_i and A_k? Does the "Sharpened No Free Lunch" result still apply? The answer can be no.

To be more formal, consider any algorithm A_i applied to function f_j. Let $Apply(A_i, f_j, m)$ represent a "meta-level" algorithm that outputs the order in which A_i visits m elements in the codomain of f_j after m steps. To begin with, we will assume m is the size of the search space; this means the search algorithms visit every point in the search space. We will also assume that the algorithms are deterministic. In this case, for every pair of algorithms A_k and A_i and for any function f_j, there exists another function f_l such that

$$Apply(A_i, f_j, m) \equiv Apply(A_k, f_l, m)$$

We can also reconfigure *Apply* to be a function generator which we will denote by **APPLY** such that:

$$f_{out} = \mathbf{APPLY}(A_i, A_k, f_{in}, m) \iff Apply(A_i, f_{in}, m) \equiv Apply(A_k, f_{out}, m)$$

We can also define a set that is closed with respect to the operation of the **APPLY** function. Assume that we will be given some as yet unknown algorithms A_i and A_k, and we start with a set F which contains a single function f_1. We assign $f_{in} = f_1$ and we generate a function $f_{out} = f_2$.

Define the set $C(F)$ such that the set F is a subset of $C(F)$, and if f_{in} is a member of $C(F)$ then $f_{out} = \mathbf{APPLY}(A_i, A_j, f_{in})$ is also a member of $C(F)$.

The "Sharpened No Free Lunch" theorem asserts that $C(F)$ must be a set that is closed under permutation if A_k and A_i are arbitrarily chosen (as yet unknown algorithms) and we require that algorithms A_k and A_i have identical performance when compared on all in the functions in $C(F)$. To see why this is true, assume that an "unfriendly adversary" is allowed to pick the algorithms and we have no idea which algorithms will be used. Then in the worst case, the set $C(F)$ must be closed under permutation to guarantee that algorithms A_k and A_l have the same performance under all possible comparative measures.

But what if we wish to compare exactly two algorithms, A_1 and A_2, and we are told in advance what algorithms are going to be compared? In this case we can potentially find a closure $C(F)$ defined with respect to the **APPLY** function such that the set $C(F)$ need not be closed under permutation in order for algorithms A_1 and A_2 to display the same aggregate performance over the set of function in $C(F)$ for all possible comparative measures. This leads to the following "Focused No Free Lunch" theorem:

Let A_1 and A_2 be two predetermined algorithms and let F be a set of functions. The aggregate performance of A_1 and A_2 are equivalent over the set $C(F)$; furthermore, the set $C(F)$ need not be closed under permutation.

This can happen in different ways. First, assume that the algorithms are deterministic and that m is the size of the search space. The search behaviors of A_1 and A_2 when executed on a function f_1 can induce a permutation group such that the orbit of the group is smaller than the permutation closure. Whitley and Rowe [17,18] give examples of how this can happen.

Second, in virtually all real applications m is polynomial with respect to input size of the problem, while the search space is exponential. Let N denote the size of the search space. Reconsider the computation where $m << N$.

$$f_{out} = \mathbf{APPLY}(A_i, A_j, f_{in}, m)$$

There now can be exponentially many functions that can play the role of f_{out} since the behavior of f_{out} is defined at only m point in the search space, and the other points in the search space can be reconfigured in $(N - m)!$ ways, all of which are unique if the function f_{in} is a bijection. Intuitively, we should no longer need the entire permutation closure to obtain identical performance over some set of functions when only a tiny fraction of the search space is explored. In fact, under certain conditions one can prove that given two predetermined algorithms A_1 and A_2 there can exist functions f_1 and f_2 such that

$$Apply(A_1, f_1, m) \equiv Apply(A_2, f_2, m)$$

$$Apply(A_1, f_2, m) \equiv Apply(A_2, f_1, m)$$

so that a Focused No Free Lunch result holds over a set of only two functions such that: $C(F) = \{f_1, f_2\}$. Unfortunately, it is not widely understood that there are significant differences in the Focused and Sharpened No Free Lunch results. Furthermore, Focused No Free Lunch results (unlike Sharpened No Free Lunch results) need not be restricted to black box optimization [17, 18].

What does all of this have to do with funnels and dispersion? The point of looking at single-funnel and multi-funnel functions is to notice that such functions (even multi-funnel functions) display a tremendous amount of locality. Points in the search space that are near to each other tend to have similar evaluations. Any reasonable algorithm should exploit this property; furthermore, we should be able to prove that relatively traditional search methods are better than random search when applied to such functions.

Assuming we can define some target threshold value, we might be able to define a search algorithm that spends more than half the time exploring parts of the search space that are below this threshold value. In this way we can "avoid" the general and Sharpened No Free Lunch results and claim that our search algorithm is in some sense robust. (Focused No Free Lunch results only compare specific algorithms and do not necessarily address whether a search algorithm is better than random search.)

But to "avoid" general No Free Lunch results, an algorithm must target a particular family of functions. We might also like to have some assurance that an algorithm is relatively effective on a wide range of problems. As we apply a threshold, we can start to ask questions about how many basins of attraction are below threshold, and how large they are. Actually, we often are not really interested in basins of attraction, but rather in "funnels" or clusters of local optima that are all below threshold. Later in this chapter, we introduce the concept of a "quasi-basin" to capture this idea.

Going back to No Free Lunch, these proofs make it clear that one cannot claim that one search algorithm is better than another without also describing the functions and landscapes where one search algorithm will out-perform another. It has also sometimes been suggested that one search algorithm is more robust than another, in the sense that it will perform well across a wide range of problems. However, the concept of robustness also leads back to the same question: if an algorithm performs well on a wide range of problems, then on what problems does it do well, and where does it fail to do well?

Thus, if we want to formalize the idea that a search algorithm has robust performance across a wide range of search problems, we have to say that an algorithm has robust performance on a *particular* family of problems. In the remainder of this chapter, we look at a kind of problem structure that might be found in either single-funnel or multi-funnel landscapes. We then define a class of subthreshold-seeking algorithms that can exploit this type of landscape.

4.4 Subthreshold-Seeking Algorithms

Christensen and Oppacher [4] have shown that the No Free Lunch theorem does not hold over broad classes of problems that can be described using polynomials of a single variable. The algorithm that Christensen and Oppacher propose is in some sense robust in as much as it is able to outperform random enumeration on a general class of problems. But the algorithm they propose is not practical as a search algorithm. Can we do better than this? And what theoretical and practical implications does this question imply?

We will first generalize the approach of Christensen and Oppacher. We will say that an algorithm has *subthreshold-seeking behavior* if the algorithm establishes a performance threshold, and then spends more than half of its time sampling points that are below threshold. An algorithm with subthreshold-seeking behavior can beat random enumeration and avoids the Sharpened No Free Lunch result by focusing on a special, but nevertheless general, class of functions. We will also say that an algorithm is *robust* if it is able to outperform random enumeration across a general class of functions, such as functions with a bounded number of optima or the set of functions that can be described using polynomials.

We next ask to what degree does a Hamming neighborhood local search algorithm using a bit representation display robust, subthreshold-seeking behavior. In addition to the theoretical analysis presented, we empirically show that a local search algorithm with sufficient precision will spend most of its time "subthreshold" on a number of common benchmark problems. We next make a very simple modification to a local search algorithm to allow it to spend more time subthreshold.

In the heuristic search community, local search with restarts is generally seen as a broadly effective mean of sampling many local optima, and therefore of finding globally competitive solutions. A Subthreshold Local Search algorithm is proposed that samples the search space to estimate a threshold and to generate a sample of subthreshold points from which to search. After this initial sample, local search is always restarting from subthreshold points in the search space. Empirically, a Subthreshold Local Search algorithm is both more efficient and effective than simple local search with restarts, finding better solutions faster on common benchmark problems.

Subthreshold Local Search is not a state-of-the-art heuristic search method; but the algorithm is a viable alternative to local search with restarts, and the algorithm and the various proofs provide new insights into the general robustness of local search. The results may also have implications for population-based search methods, such as evolutionary algorithms. By using a population combined with selection, there is an explicit sampling mechanism that attempts to focus search in the best (potentially subthreshold) regions of the search space; this is particularly true for steady-state genetic algorithms and $(\mu + \lambda)$ evolution strategies, where the population is made up of the best-so-far solutions.

4.4.1 The SubMedian-Seeker

Let f be an objective function $f : [a, b] \rightarrow R$, where $[a, b]$ is a closed interval. We discretize this interval by taking N uniformly sampled points, which we label with the set $\mathcal{X} = 0, 1, \ldots, N - 1$. By abuse of notation, we will consider $f : \mathcal{X} \rightarrow R$, such that $f(x)$ takes on the evaluation of the point labeled x. Assume f is bijective as a function of $\mathcal{X}$ and that the median value of f is known and denoted by $med(f)$.

Christensen and Oppacher define a minimization algorithm called *Sub-Median-Seeker*. The original SubMedian-Seeker is able to detect and exploit functions where every second point is below submedian and thus a local optimum. However, such functions are maximally multimodal. We present a simplified form of SubMedian-Seeker called EZ-SubMedian-Seeker that does not detect this regularity, but otherwise retains all of the critical behaviors of the original SubMedian-Seeker. The algorithm presented here is a similar to SubMedian-Seeker but is simpler and easier to understand.

```
EZ-SubMedian-Seeker
```

1. Choose a random sample point, $x \in \mathcal{X}$.
2. While $f(x) < med(f)$ pick next sample $x = x + 1$.
3. If less than $\frac{|\mathcal{X}|}{2}$ points have been sampled, then goto step 1. Otherwise terminate.

Without loss of generality, we assume that x and its successor $x + 1$ are integers. The algorithm exploits the fact that for certain classes of functions, points that are adjacent to submedian points are more often than not also submedian points. The actual performance depends on $M(f)$, which measures the number of submedian values of f that have *successors* with supermedian values. Let M_{crit} be a critical value relative to $M(f)$ such that when $M(f) < M_{crit}$ SubMedian-Seeker (or EZ-SubMedian-Seeker) is better than random search.

Christensen and Oppacher [4] then prove:

If f is a uniformly sampled polynomial of degree at most k and if $M_{crit} > k/2$ then SubMedian-Seeker beats random search.

The Christensen and Oppacher proof also holds for EZ-SubMedian-Seeker. Note there are at most k solutions to $f(x) = y$, where y can be any particular codomain value. If y is a threshold, then there are at most k crossings of this threshold over the sampled interval. Half, or $k/2$, of these are crossings from subthreshold to superthreshold values. Thus, $M(f) <= k/2$ for polynomials of degree k. In the case where the median is the threshold, step 1 has equal probability of sampling either a submedian or supermedian value. Therefore, as long as step 2 generates a surplus of submedian points before terminating at a supermedian point, the algorithm beats random search. We can think of step 1 as an exploration phase with balanced cost and step 2 as an exploitation phase that accumulates submedian points. If $M(f) \leq k/2 < M_{crit}$, then SubMedian-Seeker (and EZ-SubMedian-Seeker) will perform better than random enumeration because more time is spent below threshold during step 2. Christensen and Oppacher offer extensions of the proof for certain multivariate polynomials as well. In the next section we characterize a more general Subthreshold-Seeker algorithm.

4.4.2 Subthreshold-Seeker

We still assume f is one-dimensional and bijective and $N = |\mathcal{X}|$. Set a threshold of α between 0 and $1/2$. We are interested in spending time in the αN best points of the search space (ordered by f). We refer to these as *subthreshold* points. Addition is $modulo\ N$ and the search space is assumed to wrap around so that points 0 and $N - 1$ are neighbors.

Let $\Theta(f)$ denote a threshold codomain value such that exactly αN points of $\mathcal{X}$ have evaluations $f(x) < \Theta(f)$. Subthreshold-Seeker works as follows:

1. Pick a random element $x \in \mathcal{X}$ that has not been seen before.
2. If $f(x) < \Theta(f)$ let $x = x + 1$ and $y = x - 1$; otherwise goto 1.
3. While $f(x) < \Theta(f)$ pick next sample $x = x + 1$.
4. While $f(y) < \Theta(f)$ pick next sample $y = y - 1$.
5. If Stopping-Condition not true, goto 1.

Once a subthreshold region has been found, this algorithm searches left and right for subthreshold neighbors. This minor variation on SubMedian-Seeker means that a well-defined region has been fully exploited. This is critical to our quantification of this process. We will address the "Stopping-Condition" later.

For theoretical purposes, we assume that the function $\Theta(f)$ is provided. In practice, we can select $\Theta(f)$ based on an empirical sample.

4.4.2.1 Functions with Uniform Quasi-basins

We formally define a **quasi-basin** for a one-dimensional function, f, with respect to a threshold value, v, where v is a codomain value of f: a quasi-basin is a set of contiguous points in f that are below value v. Note this is different from the usual definition of a basin: a quasi-basin may contain multiple local optima. Also, points in a basin that are above threshold are not in the quasi-basin. An example of two

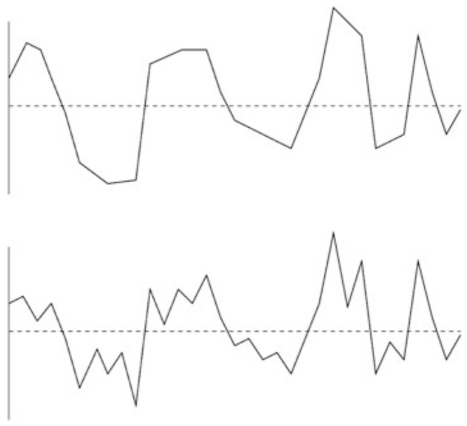


Fig. 4.5 The graph shows two examples of functions and the corresponding subthreshold quasi-basins. The quasi-basins are exactly the same size and cover the same intervals for this particular threshold: in this case the threshold is the median

functions with the same intervals as quasi-basins is given in Fig. 4.5. Note that the Subthreshold-Seeking algorithm as well as SubMedian-Seeker are only sensitive to the size and number of quasi-basins and are not sensitive to the actual number of local optima.

This concept of a quasi-basin can be related back to the concept of single-funnel and multi-funnel functions. At least one type of single-funnel function is one where there exists one large quasi-basin (potentially surrounded by small quasi-basins) that contains the global optimum as well as most of the better local optima in the space. Again, refer to the example of Rastrigin's function in Fig. 4.1. However, if there are multiple quasi-basins of similar size that are not localized in one particular region of the search space, this suggests a multi-funnel landscape. However, even on multi-funnel landscapes, simple search strategies can still be effective. Consider a function f where all subthreshold points are contained in B equally sized quasi-basins of size $\alpha N/B$. We then ask how many superthreshold points are visited before all the subthreshold points are found. Suppose k quasi-basins have already been found and explored. Then there remain $B-k$ quasi-basins to find, each containing $\alpha N/B$ points. There are at most $N-k\alpha N/B$ points unvisited. So the probability of hitting a new quasi-basin is (slightly better than)

$$\frac{(B-k)(\alpha N/B)}{N-k\alpha N/B} = \frac{(B-k)\alpha}{B-k\alpha}$$

This calculation is approximate because it assumes that superthreshold points are sampled with replacement. As long as the probability of randomly sampling the same superthreshold point twice is extremely small, the approximation will be accurate. For large search spaces this approximation should be good.

If the probability of "hitting" a quasi-basin is p, the expected number of trials until a "hit" occurs is $1/p$. This implies that the expected number of misses before a successful hit occurs is $1/p-1$. So the expected number of superthreshold points that are sampled before finding a new quasi-basin is approximately (slightly less than)

$$\frac{B - k\alpha}{(B - k)\alpha} - 1 = \frac{B(1 - \alpha)}{(B - k)\alpha}$$

This means that the expected number of superthreshold points seen before the algorithm has found all quasi-basins is bounded above by

$$\sum_{k=1}^{B-1} \frac{B(1 - \alpha)}{(B - k)\alpha} = \frac{B(1 - \alpha)}{\alpha} \sum_{k=1}^{B-1} \frac{1}{(B - k)} = \frac{B(1 - \alpha)}{\alpha} H(B - 1) \qquad (4.1)$$

where H is the harmonic function. Note $H(B - 1)$ is approximated by $\log(B - 1)$.

In general terms, we might expect functions that display several similarly sized quasi-basins to be one type of multi-funnel landscape. In contrast, in the case of a single-funnel landscape we might expect there to be a single larger quasi-basin that dominates other subthreshold regions of the search space. We next consider the case where there exist unevenly sized quasi-basins.

4.4.2.2 Functions with Unevenly Sized Quasi-basins

Theorem 4.1. *Let f be a one-dimensional function uniformly sampled by N points. Let α be a threshold such that $\alpha \leq 1/2$, and suppose there are B subthreshold quasi-basins which are not uniform in size, where $B \geq 2$. Run Subthreshold-Seeker until one quasi-basin of size at least $\alpha N/B + 1$ is found. In expectation, more subthreshold points will be sampled than superthreshold points when*

$$2B^2 + \alpha + \alpha N - 2\alpha B - \frac{B^2}{\alpha} - \frac{\alpha^2 N}{B} > 0$$

A weaker sufficient condition is

$$\alpha > \frac{B}{\sqrt{N + B^2}}$$

Proof. Since the quasi-basins are not uniform in size, there must be at least one subthreshold quasi-basin of size $\alpha N/B + 1$ or larger. We will explicitly search for a targeted subset of exactly $\alpha N/B$ adjacent points in some quasi-basin that is of size $\alpha N/B + 1$ or larger. For purposes of the proof, we assume we know when we have found the targeted set. For the actual algorithm, some smaller quasi-basins may be enumerated and this enumeration is not included in the current calculation.

The expected waiting time to find the targeted set of points is $\frac{N}{\alpha N/B} - 1$. Since we are looking for a specific target set of points, under random sampling we will sample both superthreshold and subthreshold points before finding this target set.

The set of nontargeted points is of size $N - \alpha N/B$, and $\alpha N - \alpha N/B$ of these are subthreshold. Therefore $\frac{\alpha N - \alpha N/B}{N - \alpha N/B}$ of the points sampled before finding the targeted region will be subthreshold in expectation. Thus the expected number of subthreshold points sampled before a targeted point is found is $\frac{\alpha N - \alpha N/B}{N - \alpha N/B}(\frac{N}{\alpha N/B} - 1)$, and when a targeted point is found then $\alpha N/B$ subthreshold points are sampled.

At the edges of the target at most one superthreshold point is sampled, balanced by one subthreshold point, so these can be ignored. The expected number of superthreshold points sampled before a targeted point is found is $\frac{(1-\alpha)N}{N-\alpha N/B}(\frac{N}{\alpha N/B} - 1)$.

Therefore more subthreshold points are sampled when

$$\frac{\alpha N - \alpha N/B}{N - \alpha N/B}(\frac{N}{\alpha N/B} - 1) + \alpha N/B > \frac{(1-\alpha)N}{N - \alpha N/B}(\frac{N}{\alpha N/B} - 1)$$

Simplifying yields:

$$\frac{(\alpha - \alpha/B)(B/\alpha - 1)}{1 - \alpha/B} + \frac{\alpha}{B}N > \frac{(1-\alpha)(B/\alpha - 1)}{1 - \alpha/B} \tag{4.2}$$

$$2B^2 + \alpha + \alpha N - 2\alpha B - \frac{B^2}{\alpha} - \frac{\alpha^2 N}{B} > 0$$

We next used relaxed bounds to show that more subthreshold points are sampled in expectation when $\alpha > \frac{B}{\sqrt{N+B^2}}$. Given $\alpha \leq 1/2$ and $B \geq 2$, the following inequalities hold with respect to Eq. (4.2).

$$\frac{(\alpha - \frac{\alpha}{B})(\frac{B}{\alpha} - 1)}{1 - \alpha/B} + \frac{\alpha}{B}N > (\alpha - \frac{\alpha}{B})(\frac{B}{\alpha} - 1) + \frac{\alpha}{B}N = B - \alpha - 1 + \frac{\alpha}{B} + \frac{\alpha}{B}N > \alpha B + \frac{\alpha}{B}N - 1$$

$$\text{and} \quad B/\alpha - 1 > \frac{(1-\alpha)(B/\alpha - 1)}{1 - \frac{\alpha}{B}}$$

It therefore follows that Eq. (4.2) holds when $\alpha B + \frac{\alpha}{B}N - 1 > \frac{B}{\alpha} - 1$.

Using these more conservative bounds

$$\alpha B + \frac{\alpha}{B}N - 1 > \frac{B}{\alpha} - 1$$

$$B^2 + N > \frac{B^2}{\alpha^2}$$

$$\sqrt{N + B^2} > \frac{B}{\alpha}$$

$$\alpha > \frac{B}{\sqrt{N + B^2}} \qquad \square$$

Corollary. *If f is a polynomial of degree z and we run Subthreshold-Seeker until a quasi-basin of size at least $\alpha N/(z/2) + 1$ is found, then more subthreshold points will be sampled than superthreshold points if $\alpha > \frac{z}{(2\sqrt{N+(z/2)^2})}$.*

Proof. A polynomial of degree z can have at most $z/2$ quasi-basins. $\square$

4.5 Quasi-basins and Local Search in Hamming Neighborhoods

In this section, we outline sufficient conditions to ensure that the majority of Hamming distance 1 neighbors under Gray and binary encodings are in the same quasi-basin. We also look at how precision affects the number of neighbors that lie in the same quasi-basin.

Observation. *Given a 1-D function of size $N = 2^L$ and a reference point R in the function, under a Gray or binary encoding at most $\lceil \log(Q) \rceil$ bits encode for points that are more than a distance of D points away from R, where $D = \frac{1}{Q}N$.*

In the one-dimensional case when the highest order bit is changed under either a Gray or binary encoding this accesses the only neighbor that is in the opposite half of the search space.

We first will select a reference point R in the quasi-basin. Next, we can recursively eliminate the highest-order bit so as to remove the half of the search space which does not contain the reference point. We continue to reduce the search space around the reference point by removing bits until $\lceil \log(Q) \rceil$ bits have been eliminated. The remaining search space is then at most $D = N/Q$ points since

$$\log(N/Q) + \log(Q) = \log(N) \ \text{, and} \ N(1/2)^{\lceil \log(Q) \rceil} \leq N/Q$$

At higher precision under both Gray representation and binary representations, more points are sampled that are near to the reference point. Thus, as precision increases, the quantity N/Q becomes larger and thus $\log(N/Q)$ increases. However Q and $\log(Q)$ remain constant. Therefore, at higher precision, the number of neighbors within a distance of N/Q points increases.

Expressed another way, consider a quasi-basin of size D and a search space of size N where the quasi-basin spans $1/Q$ of the search space (i.e., $D = \frac{1}{Q}N$): under a bit representation at most $\lceil \log(Q) \rceil$ bits encode for points that are more than a distance of D points away from R. Note that an increase in precision also increases the size of the search space, so that the quantity N/Q becomes larger and thus $\log(N/Q)$ increases. However Q and $\log(Q)$ remain constant. Thus, at higher precision, the number of neighbors within a distance of N/Q points increases.

Whitley and Rowe [16] present proofs that show that the expected number of neighbors of R that fall inside the quasi-basin under a reflected Gray code as well as under the standard binary encoding is greater than $\lfloor\log(N/Q)\rfloor - 1$. This involves a very detailed analysis of the Gray code and binary code neighborhoods. The proof looks at the expected number of neighbors of reference point R that fall into a quasi-basin by averaging over all of the possible placements of R in the Gray and binary neighborhood. Expressed another way, the proof considers all the ways that the Gray or binary neighborhood could span the quasi-basin, regardless of where reference point R falls inside of the quasi-basin. Thus, if $1/Q$ is the fraction of the search space that is occupied by a given quasi-basin, and reference point R falls into the quasi-basin, then in expectation more than $\lfloor\log(N/Q)\rfloor - 1$ of the $\log(N)$ neighbors of reference point R will also fall into the same quasi-basin.

The significance of this result is that we can outline conditions that would allow a steepest-ascent local search algorithm to spend the majority of its time sampling points that are contained in the same quasi-basin and therefore below threshold. This makes it possible to outline sufficient conditions such that steepest ascent local search is provably better than random search.

Note that for single-funnel functions where the search space is dominated by one large quasi-basin, we can assume that the number of neighbors that fall into the same quasi-basin should be a relatively good approximation of the number of neighbors that are below threshold. For multi-funnel functions, the count of the expected number of neighbors that fall in the same quasi-basin will fail to count neighbors that might fall into other quasi-basins.

The result is limited by the fact that the analysis holds for one-dimensional functions. This clearly is not a problem for separable functions however, since each subproblem is an independent one-dimensional problem. And because a Hamming neighborhood local search algorithm changes only one bit at a time, when we vary the bits for a single parameter this dynamically isolates a one-dimensional slice of the search space and, because the reference point R is subthreshold, there is a corresponding quasi-basin that is sampled in that slice. However, the size of the quasi-basin can change as we move from one parameter to the next. But the size of the quasi-basin can also change depending on the reference point. In a sense, the real limitation is not that the analysis holds for one-dimensional functions, but rather that we do not really know the size of the quasi-basins or how this size varies in different slices of the search space. Nevertheless, regardless of what one-dimensional slice of the search space we sample, we do have a characterization of what it means for the quasi-basin to be sufficiently large to allow a local search algorithm that is currently below threshold to sample below threshold the majority of the time.

4.5.1 A Subthreshold Local Search Algorithm

A local search algorithm without restarts that is currently at a subthreshold point can only move to an equal or better point which must also be subthreshold. The real question becomes, "Are the majority of neighbors also below threshold?" For a Hamming neighborhood, as precision increases, the number of subthreshold neighbors also increases, since $\lfloor(\log(N/Q))\rfloor - 1$ increases while Q remains constant. This assumes the quasi-basin is not divided by increasing the precision. The above analysis would need to hold for each dimension of a multidimensional search space, or at least hold in a cumulative sense over all dimensions. Nevertheless, our results suggest there are very general conditions where a Hamming neighborhood local search algorithm can display subthreshold-seeking behavior. This also assumes local search can absorb the start-up costs of locating a subthreshold starting point.

We will limit our implementation to Gray code representations. Under favorable conditions a Hamming neighborhood local search algorithm using a Gray code representation can display subthreshold-seeking behavior, but does local search display subthreshold-seeking behavior on common benchmarks? In this section, we compare two versions of Hamming neighborhood local search algorithms. Both algorithms use steepest-ascent Local Search (LS) which evaluates all neighbors before moving. One algorithm, denoted LS-Rand, uses random restarts. The other algorithm, denoted LS-SubT, uses sampling to start local search at a subthreshold point.

LS-SubT first samples 1,000 random points, and then applies local search from the 100 best of these points. In this way, LS-SubT estimates a threshold value and attempts to stay in the best 10 % of the search space.

LS-Rand does 100+y random restarts. LS-Rand was given y additional random starts to compensate for the 1,000 sample evaluations used by the LS-SubT algorithm. To calculate y we looked at the size of the bit encoding and the average number of moves needed to reach a local optimum.

4.5.2 Experiments and Results

Both LS-Rand and LS-SubT were tested on benchmarks taken from Whitley et al. [15], who also provide function definitions. The test function included Rastrigin (F6) and Schwefel (F7), which are both separable. The other functions include Rosenbrock, F101 and Rana functions as well as a *spike* function similar to one defined by Ackley [1], where:

$$F(x, y) = -20e^{-0.2\sqrt{(x^2+y^2)/2}} - e^{(cos2\pi x+cos2\pi y)/2} + 22.7, \quad x_i \in [-32.77, 32.77]$$

All problems were posed as 2-dimensional search problems. Experiments were performed at 10- and 20-bits of resolution per parameter. A *descent* corresponds to one iteration of local search, which will locate one local optimum. A *trial*

Table 4.1 Results of steepest-ascent search at 10-bit resolution per parameter in two-dimensional space. LS-Rand (here *Rand*) used 104 restarts. LS-SubT (here *SubT*) restarted from best 100 of 1,000 random points. Evals were rounded to the nearest integer

Function	ALG	Mean	σ	Best	σ	Sub	Evals	σ
Ackley	Rand	2.72	0.71	0.18	0.0[b]	62.4	19,371	663
	SubT	0.79[a]	0.32	0.18	0.0[b]	79.7	16,214[a]	163
F101	Rand	−29.2	0.0[b]	−29.2	0.0[b]	71.7	22,917	288
	SubT	−29.2	0.0[b]	−29.2	0.0[b]	84.0	18,540[a]	456
Rosenbrock	Rand	0.10	0.01	0.001	0.002	61.4	23,504	3,052
	SubT	0.10	0.01	0.0004	0.0[b]	72.0	666[a]	1,398
Griewangk	Rand	0.86	0.16	0.010	0.011	59.5	13,412	370
	SubT	0.75[a]	0.11	0.005	0.009	80.1	9,692[a]	125
Rana	Rand	−37.8	0.84	−49.65	0.59	49.5	22,575	2,296
	SubT	−39.7[a]	0.68	−49.49	0.52	57.6	19,453[a]	1,288
Rastrigin	Rand	4.05	0.20	0.100	0.30	63.5	18,770	495
	SubT	4.00	0.28	0.0	–	75.4	14,442[a]	343
Schwefel	Rand	−615.8	11.8	−837.9	0.0[b]	53.5	17,796	318
	SubT	−648.0[a]	10.1	−837.9	0.0[b]	68.0	14,580[a]	414

[a] Denotes a statistically significant difference at the 0.05 level using a t-test

[b] Denotes a value less than 1×10^{-13}

corresponds to 1 run of the respective algorithm, composed of 100 descents for LS-SubT and $100 + y$ descents for LS-Rand. An *experiment* corresponds to 30 trials. Each experiment is a configuration of search algorithm, test function and parameter resolution. Statistics are computed over each experiment. All chromosomes were encoded using standard Gray code.

The results of 10- and 20-bit resolution experiments are given in Tables 4.1 and 4.2, respectively. $Mean$ denotes mean solution over all descents in all trials. (This is also the mean over all local optima found.) $Best$ denotes the *best solution per trial* (i.e., the best optimum found over 100 or $100 + y$ descents). Sub denotes the percentage of all evaluations that were subthreshold. $Evals$ denotes the mean number of test function evaluations per trial averaged over all trials in the experiment; σ denotes the standard deviation of the value given in the adjacent left-hand column.

In general, the results indicate that LS-SubT sometimes produces statistically significant better solution quality compared to LS-Rand. LS-SubT never produces statistically significant worse performance than LS-Rand.

The data suggest two observations about subthreshold-seeking behavior. First, the sampling used by LS-SubT results in a higher proportion of subthreshold points compared to LS-Rand, as shown in Tables 4.1 and 4.2. Second, a larger proportion of subthreshold neighbors are sampled for searches using higher precision. At 20 bits of precision per parameter, at least 70 % of the points sampled by LS-Rand were subthreshold, and at least 80 % of the points samples by LS-SubT were subthreshold. At 10 bits of precision per parameter, LS-SubT sampled subthreshold points 57–84 % of the time.

Table 4.2 Results of steepest-ascent search at 20-bit resolution per parameter in two-dimensional space. LS-Rand (here *Rand*) used 101 restarts. LS-SubT (here *SubT*) restarted from best 100 of 1,000 random points. Evals were rounded to the nearest integer

Function	ALG	Mean	σ	Best	σ	Sub	Evals	σ
Ackley	Rand	2.84	0.66	0.0001	0.0[b]	75.1	77,835	1,662
	SubT	0.65[a]	0.28	0.0001	0.0[b]	89.9	73,212[a]	1,194
F101	Rand	−29.2	0.0[b]	−29.22	0.0[b]	84.7	84,740	1,084
	SubT	−29.2	0.0[b]	−29.22	0.0[b]	92.3	77,244[a]	1,082
F2	Rand	0.0[b]	0.0[b]	0.0[b]	0.0[b]	86.0	2×10^7	4×10^5
	SubT	0.0[b]	0.0[b]	0.0[b]	0.0[b]	85.9	2×10^7	3×10^5
Griewangk	Rand	0.75	0.20	0.0045	0.003	80.3	66,609	1,109
	SubT	0.60[a]	0.09	0.0049	0.003	90.0	59,935[a]	1,103
Rana	Rand	−40.63	0.93	−49.76	0.47	74.2	3×10^6	8×10^5
	SubT	−42.54[a]	0.66	−49.83	0.51	85.0	3×10^6	8×10^5
Rastrigin	Rand	4.10	0.22	0.033	0.18	81.5	76,335	1,734
	SubT	3.94[a]	0.21	0	–	88.5	68,019[a]	1,018
Schwefel	Rand	−622.7	13.8	−837.97	0.0[b]	73.5	75,285	969
	SubT	−660.4[a]	13.4	−837.97	0.0[b]	84.8	69,372[a]	1,340

[a] Denotes a statistically significant difference at the 0.05 level using a t-test

[b] Denotes a value less than 1×10^{-7}

At 10 bits of precision, LS-SubT also did fewer evaluations, meaning that it reached local optima faster than LS-Rand. This makes sense in as much as it starts at points with better evaluations. Sometimes the difference was dramatic. Thus, the majority of the time LS-SubT also produced solutions as good or better than LS-Rand, and it did so with less effort.

At 20 bits of precision, there is less difference between LS-Rand and LS-SubT. This follows from our theory, since higher precision implies that both algorithms spend more time subthreshold after a subthreshold point is found, but this does not necessarily result in faster search.

The number of evaluations that were executed on the Rana and F2 functions at 20-bit resolution is huge. Examination of the search space shows that both of these functions contain "ridges" that run at almost 45° relative to the (x, y) coordinates. In this context, the local search is forced to creep along the ridge in very small, incremental steps. Higher precision exaggerates this problem, which is hardly noticeable at 10 bits of precision. This is a serious problem for search algorithms that are not rotationally invariant [19]; it is also a good argument for the use of algorithms such as CMA-ES which are rotationally invariant [8].

4.6 Conclusions

The No Free Lunch theorem formalizes the idea that all black box search algorithms have identical behavior over the set of all possible discrete functions [5, 11, 21]. The Sharpened No Free Lunch theorem extends this idea to sets of functions closed under permutations. In both cases, one is unable to say that any algorithm is better than random enumeration. In this paper, conditions are outlined that allow a subthreshold-seeking algorithm to beat random enumeration on problems of bounded complexity. The Subthreshold-Seeker algorithm is able to focus search in the better regions of the search space.

The paper also examines the potential for subthreshold-seeking behavior for local search algorithms using binary and Gray code representations. Subthreshold-seeking behavior can be increased by using higher bit precision, but this also reduces exploration. A simple modification to local search is proposed that improves its subthreshold-seeking behavior. A simple sampling mechanism can be used to initialize local search at subthreshold points, thereby increasing the potential for subthreshold-seeking behavior. Experiments show that this modification results in faster convergence to equally good or better solutions compared to local search without subthreshold initialization. Of course this strategy also has its own failure modes. Assume that an "important" basin of attraction, or a quasi-basin, is very large above threshold, yet small below threshold; then it is possible that random restarts could have an advantage over subthreshold restarts if success were measured in terms of finding and exploiting this "important" region. Of course, the problem with random restarts is that the search can also converge to local optima that are superthreshold.

The trend in recent years has been to use real-valued representations for parameter optimization problems. One reason for using bit representations in the current study is that it produces a well-defined finite search space that allows one to explicitly count neighbors. But in principle, the same general ideas could be applied.

Acknowledgements This research was sponsored by the Air Force Office of Scientific Research, Air Force Materiel Command, USAF, under grant number FA9550-07-1-0403. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. We would also like to thank Dagstuhl for giving us the chance to meet and exchange ideas.

References

1. D. Ackley, *A Connectionist Machine for Genetic Hillclimbing* (Kluwer Academic, Boston, 1987)
2. K.D. Boese, A.B. Kahng, S. Muddu. On the big valley and adaptive multi-start for discrete global optimizations. Technical report, UCLA CS Department, 1993
3. K.D. Boese, A.B. Kahng, S. Muddu, A new adaptive multi-start technique for combinatorial global optimizations. Oper. Res. Lett. **16**, 101–113 (1994)
4. S. Christensen, F. Oppacher, What can we learn from no free lunch? in *GECCO-01*, San Francisco, 2001 (Morgan Kaufmann, 2001), pp. 1219–1226
5. J. Culberson, On the futility of blind search. Evolut. Comput. **6**(2), 109–127 (1998)
6. J. Doye, M. Miller, D. Wales, The double-funnel energy landscape of the 38-atom Lennard-Jones cluster. J. Chem. Phys. **110**(14), (1999)
7. J. Doye, R. Leary, M. Locatelli, F. Schoen, Global optimization of Morse clusters by potential energy transforms. INFORMS J. Comput. **16**(4), 371–379 (2004)

8. N. Hansen, S. Kern, Evaluating the CMA evolution strategy on multimodal test functions, in *Proceedings of 8th International Conference on Parallel Problem Solving from Nature*, Birmingham (Springer, 2004), pp. 282–291
9. M. Lunacek, L.D. Whitley, The dispersion metric and the CMA evolution strategy, in *GECCO'06: proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, Seattle (ACM, New York, 2006), pp. 477–484
10. R. Marcia, J. Mitchell, J. Rosen, Multi-funnel optimization using Gaussian underestimation. J. Glob. Optim. **39**(1), 39–48 (2007)
11. N. Radcliffe, P. Surry, Fundamental limitations on search algorithms: evolutionary computing in perspective, in *Computer Science Today*, ed. by J. van Leeuwen. Lecture Notes in Computer Science, vol. 1000 (Springer, Berlin/Heidelberg, 1995)
12. C. Schumacher, M. Vose, L. Whitley, The no free lunch and problem description length, in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)*, San Francisco, 2001, pp. 565–570
13. H.-P. Schwefel, *Evolution and Optimum Seeking* (Wiley, New York, 1995)
14. A.M. Sutton, L.D. Whitley, M. Lunacek, A.E. Howe, PSO and multi-funnel landscapes: how cooperation might limit exploration, in *Genetic and Evolutionary Computation Conference (GECCO 2006)*, Seattle, 2006
15. D. Whitley, K. Mathias, S. Rana, J. Dzubera, Evaluating evolutionary algorithms. Artif. Intell. J. **85**, 1–32 (1996)
16. D. Whitley, J. Rowe, Subthreshold-seeking local search. Theor. Comput. Sci. **361**(1), 2–17 (2006)
17. D. Whitley, J. Rowe, Focused no free lunch theorems, in *GECCO-08*, Atlanta (ACM, 2008)
18. D. Whitley, J. Rowe, A no free lunch tutorial: sharpened and focused no free lunch, eds. A. Auger, B. Doerr. *Theory of Randomized Search Heuristics* (World Scientific, Singapore, 2010)
19. L.D. Whitley, M. Lunacek, J. Knight, Ruffled by ridges: how evolutionary algorithms can fail, in *Genetic and Evolutionary Computation Conference*, Seattle, vol. 2, 2004, pp. 294–306
20. D. H. Wolpert, W.G. Macready, No free lunch theorems for search. Technical report, SFI-TR-95-02-010, Santa Fe Institute, July 1995
21. D.H. Wolpert, W.G. Macready, No free lunch theorems for optimization. IEEE Trans. Evol. Comput. **4**, 67–82 (1997)

Chapter 5
Black-Box Complexity for Bounding the Performance of Randomized Search Heuristics

Thomas Jansen

Abstract In black-box optimization a search algorithm looks for a global optimum of an unknown objective function that can only be explored by sampling the function values of some points in the search space. Black-box complexity measures the number of such function evaluations that any search algorithms needs to make in the worst case to locate a global optimum of any objective function from some class of functions. The black-box complexity of a function class thus yields a lower bound on the performance for all algorithms. This chapter gives a precise and accessible introduction to the notion of black-box complexity, explains important properties and discusses several concrete examples. Starting with simple examples and proceeding step-wise to more complex examples an introduction that explains how such results can be derived is presented.

5.1 Introduction

When algorithms are applied to solve computational problems, one of the most important issues is that of efficiency. Does the proposed algorithm solve the problem at hand efficiently? Efficiency of algorithms is measured with respect to the algorithm's use of some or several resources and a performance criterion. What resources are taken into account depends on the specific application. Most often the computation time is considered to be most crucial. Other resources like the memory use are also of general importance. In specific situations, still other resources may be important; the number of sent messages is an example for algorithms that are running in a distributed computing environment. Performance criteria are usually given by the computational task. Typically, a performance criterion is solving a

T. Jansen (✉)
Department of Computer Science, Aberystwyth University, Aberystwyth SY23 3DB, UK
e-mail: t.jansen@aber.ac.uk

well-defined computational task like sorting n elements, finding some $x^* \in S$ that maximizes a given objective function $f\colon S \to \mathbb{R}$, or finding an approximation of certain quality to such an optimal solution x^*.

When one is concerned with a specific algorithm one considers its efficiency. Clearly, one is interested in finding an algorithm for the problem at hand that is as efficient as possible. This leads to a shift in perspective in a natural way. One is interested in finding out how efficient algorithms for a specific problem can be. This is usually denoted as the complexity of the problem. While the analysis of a single algorithm can be a very difficult and challenging task the analysis of a problem's complexity is obviously even more difficult. It may be surprising that for some problems tight upper and lower bounds on their complexity can be proven.

Results on the complexity of a problem are much more general than results on the performance of a specific algorithm for a problem. This implies that results on the complexity are necessarily weaker and usually deliver smaller lower bounds. They need to hold for all possible algorithms, not just a single one. The advantage of their generality is they tell us about general limits of what algorithms can achieve and let us realize when we found an optimal algorithm.

Research on algorithms and complexity theory is most often concerned with problem-specific algorithms. These algorithms are tailored towards solving a very specific computational task, like finding a minimum cost tour in the traveling salesperson problem (TSP). Randomized search heuristics, however, are very different. They are not tailored towards a specific application but implement a very general idea of how search should be conducted. They are often not tailored towards a specific task and very often are oblivious to the concrete problem instance at hand. Therefore, comparing randomized search heuristics based on classic complexity theory is inherently unfair. A more appropriate kind of complexity theory, called black-box complexity [5], exists. Within this formal framework meaningful upper and lower bounds for randomized search heuristics in black-box optimization can be derived. This chapter describes this framework and gives examples for results obtained within it.

In the next section we give formal definitions for black-box complexity. This includes the definition of a black-box algorithm, its expected worst-case optimization time, and the black-box complexity of a problem class. We continue with highlighting the difference between black-box complexity and classical computational complexity by means of an illustrative example and motivating black-box complexity as an informative measure in Sect. 5.3. Considering only the size of the function class or search space already non-trivial results on black-box complexity can be obtained. We derive such results in Sect. 5.4 and develop an intuitive understanding of important properties of black-box complexity this way. Research in the theory of evolutionary algorithms is often driven by the consideration of example functions. We connect black-box complexity to this branch of research in Sect. 5.5. Section 5.6 demonstrates how results for less simple and artificial examples can be derived by analyzing the black-box complexity of meaningful and natural classes of functions. We summarize and conclude in Sect. 5.7.

5.2 Randomized Search Heuristics and Black-Box Complexity

5.2.1 Basic Concepts

Complexity theory starts with the definition of a performance criterion and a computational resource. For the definition of black-box complexity we consider the optimization of a function $f: S \rightarrow \mathbb{R}$ as performance criterion. An algorithm completes this task when it finds some $x^* \in S$ that maximizes f, i.e. $f(x^*) = \max\{f(x) \mid x \in S\}$ holds. It is not difficult to adapt the framework to other performance criteria like approximation. We restrict our attention to finite search spaces S. The task that we consider is black-box optimization. This means that the algorithm does not know about the function f. What is known to the algorithm is that the unknown objective function f is member of some class of function $\mathcal{F} \subseteq \{g: S \rightarrow \mathbb{R}\}$. Note that the search space S is the same for all potential objective functions g. The only way for an algorithm to gather knowledge about the objective function f is to sample some points in the search space. We can imagine that the objective function f is hidden from the algorithm in a black box. The algorithm may present points $x \in S$ to the black box and receives the function value $f(x)$ as answer. In this sense the black box serves as an oracle to the algorithm. Although the definition of black-box complexity works for all kinds of algorithms, we define it with randomized search heuristics in mind. Randomized search heuristics tend to be simple and computationally cheap. In most cases it is reasonable to assume that their computational effort is tightly connected to the number of function evaluations, i.e. accesses to the oracle, they require. We take this idea to the extreme by using this number of function evaluations as the only resource. Thus, the resources used by a run of an algorithm solving this black-box problem equal the number of function evaluations carried out. This abstract definition has the immediate advantage that we do not need to define a specific model of computation. We do not care about the kind of computations the algorithm carries out and only take the function evaluations into account. We discuss some implications of this choice not to restrict the computational powers of black-box algorithms in any way in Sect. 5.3.

Most randomized search heuristics are incomplete algorithms. Such algorithms may find an optimal solution but they do not notice that. We take this into account by counting the number of function evaluations until a global optimum of the objective function f is found for the first time without asking whether the algorithm has any proof or knowledge that it actually found a global optimum. While it is clear that this may be difficult when comparing complete and incomplete algorithms [10] it is sensible to use this generous definition to investigate the potential of randomized search heuristics.

Having defined the performance criterion (optimization of some unknown function $f: S \rightarrow \mathbb{R}$ with $f \in \mathcal{F}$, $\mathcal{F}$ known) and the resource (number of f-evaluations) as well as what we mean by optimization (sample some point $x^* \in S$

with $f(x^*) = \max\{f(x) \mid x \in S\}$), we are now ready to formally define black-box complexity. We use the usual approach considering the performance of algorithms, taking the worst case over all possible objective functions, and minimizing over all possible algorithms.

5.2.1.1 Problem Class

Complexity theory cannot be carried out for single problem instances. For a single problem instance an optimal algorithm is the trivial one that produces an optimal solution in the first. Even though we may not know this algorithm we know that it exists. In classical complexity theory one considers a problem to be set of problem instances where the instances have some size. In some sense we fix the size of the instances (by fixing the search space S) in the following definition. We clarify the differences and similarities to classical complexity theory by means of an example.

Definition 5.1. Let S be a finite set, called the *search space*, and let $\mathcal{F} \subseteq \{f: S \to \mathbb{R}\}$ be a non-empty set of potential objective functions. We denote $\mathcal{F}$ as a generic problem class.

Consider the well-known satisfiability problem (MAX-SAT). We have variables $x_1, x_2, \ldots, x_n$ and clauses $c_1, c_2, \ldots, c_m$. A clause is a disjunction of some literals. A literal is a variable or its negation. We are looking for an assignment of the n variables that satisfies as many clauses as possible. For a specific instance, i.e. for a specific set of m clauses over n variables, we can easily define a function $f: \{0,1\}^n \to \{0, 1, \ldots, m\}$ such that $f(x)$ denotes the number of clauses that the assignment $x \in \{0,1\}^n$ satisfies. In classical complexity theory one analyzes the number of computation steps an algorithm that is given $x_1, x_2, \ldots, x_n$ and $c_1, c_2, \ldots, c_m$ needs to find an assignment satisfying a maximum number of clauses and is interested in the worst-case taken over all possible instances. The complexity of the problem is the best worst case number of computation steps that is achievable. In black-box complexity we consider the class of functions $\mathcal{F}$ that contains all such functions f for a fixed value of n. Note that the black-box algorithm does not know about the specific problem instance, in particular, it does not even know about the number of clauses. We are interested in the best an algorithm can achieve under these circumstances. We restrict our interest to algorithms that actually solve the problem, i.e. they eventually solve every possible instance. We formalize this in the following subsection.

5.2.1.2 Black-Box Algorithms

Definition 5.2. An algorithm A is called a *black-box algorithm* for $\mathcal{F}$ if for all $f \in \mathcal{F}$ the expected number of function evaluations until the algorithm samples some $x^* \in S$ with $f(x^*) = \max\{f(x) \mid x \in S\}$ is finite.

This general definition captures all algorithms that guarantee global optimization. It includes very simple algorithms like pure random sampling as well as more advanced ones like simulated annealing and most evolutionary algorithms. We require, however, the algorithm to find a solution for each $f \in \mathcal{F}$. This implies that, for example, local search is a black-box algorithm for the class of unimodal problems, but it is not a black-box algorithm for the class of multimodal problems.

Note that we do not make any assumptions about the number of times a specific search point is sampled. This differs from the definition of algorithms in the No Free Lunch (NFL) scenario, where the algorithms are usually assumed to be non-repeating, i.e. never to sample any point twice ([8, 16, 18], also see the chapter on the NFL theorems, Chap. 1).

5.2.1.3 Performance Criteria and Black-Box Complexity

Just like in classical complexity theory we are interested in the best achievable worst-case performance. We formalize this based on the notion of the worst-case expected optimization time.

Definition 5.3. The *optimization time* $T_{A,f}$ of black-box algorithm A on objective function $f \in \mathcal{F}$ is the number of function evaluations A makes up to and including the first function evaluation of some $x^* \in S$ with $f(x^*) = \max\{f(x) \mid x \in S\}$. Its mean $\mathrm{E}\left(T_{A,f}\right)$ is called its *expected optimization time*, where the expectation is taken over the random choices the algorithm A makes.

The *worst-case expected optimization time* $T_{A,\mathcal{F}}$ of black-box algorithm A on the function class $\mathcal{F}$ is defined as $T_{A,\mathcal{F}} = \sup\left\{\mathrm{E}\left(T_{A,f}\right) \mid f \in \mathcal{F}\right\}$.

The *black-box complexity* $B_{\mathcal{F}}$ of the function class $\mathcal{F}$ is defined as $B_{\mathcal{F}} = \inf\{T_{A,\mathcal{F}} \mid A \text{ black-box algorithm for } \mathcal{F}\}$.

Black-box algorithms may be either randomized or deterministic. Even though most search heuristics are randomized and we are definitely most interested in randomized search heuristics, it makes sense to consider deterministic black-box algorithms first. They are structurally simpler and help us to better understand the limitations due to the scenario of black-box optimization. We consider randomized black-box algorithms afterwards.

5.2.2 Deterministic Algorithms

The first important observation is that a black-box algorithm does not have any inputs. It knows about the class $\mathcal{F}$ of potential objective functions but this knowledge was already available when the black-box algorithm was designed. It can be incorporated in the algorithm, but since it is fixed and thus cannot change, it is not an input. A deterministic black-box algorithm A may initially perform some computations, but since the algorithm has no input and since it is deterministic these computations are always the same any time A is started. At the end of these

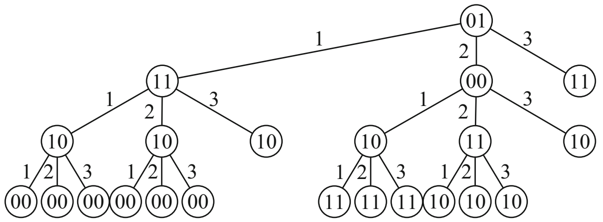


Fig. 5.1 Deterministic black-box algorithm for some $\mathcal{F} \subseteq \{f : S \to \mathbb{R}\}$ with $S = \{00, 01, 10, 11\}$ and $\{f(x) \mid f \in \mathcal{F}, x \in S\} = \{1, 2, 3\}$

computations A needs to decide about the first point in the search space it wants to sample. Let $x_1 \in S$ be this first point it samples. Clearly, this point is always the same regardless of the concrete objective function $f \in \mathcal{F}$. If x_1 happens to be a global optimum for f the algorithm A may stop. Otherwise it can again perform arbitrary computations before deciding about the second point in the search space it wishes to sample. These computations now can depend on $f(x_1)$. This value was not known to the algorithm beforehand and can thus be regarded as a kind of input. We thus denote the second point in the search space that A samples as $x_2^{(f(x_1))}$. If a third point in the search space is sampled this can, of course, depend on $f(x_1)$, $x_2^{(f(x_1))}$, and $f\left(x_2^{(f(x_1))}\right)$. We see how this continues. It is worth mentioning that this insight paves the way for a compact and very convenient notation for deterministic black-box algorithms. The key observation is that we do not care about the computations A carries out at all. We only are concerned with the points in the search space it samples and their function value. We can organize these points in a tree. The root of the tree is a node that we label with x_1, the first point in the search space sampled by A. For each possible function value of x_1 we have one edge leaving the root connecting it with a node that is labeled with $x_2^{(f(x_1))}$. For the sake of clarity we label the edges by $f(x_1)$. We continue this way, constructing a complete tree. In cases where $\{f(x) \mid f \in \mathcal{F}, x \in S\}$ is a finite set we obtain a tree where each node has finite degree. For the investigation of black-box complexity we care about optimal algorithms. Clearly, such algorithms sample any point in the search space at most once. So, if we have $\{f(x) \mid f \in \mathcal{F}, x \in S\}$ finite we know that optimal black-box algorithms for $\mathcal{F}$ are finite trees. It is convenient to identify any deterministic black-box algorithm by its tree. Figure 5.1 shows an example. We can replace these trees by more compact trees by having more than one label at an edge. The more compact version of the tree in Fig. 5.1 can be seen in Fig. 5.2.

5.2.3 Randomized Algorithms

For randomized black-box algorithms things are less simple. Since randomized black-box algorithms can make use of random choices there is no single fixed $x_1 \in S$ that is sampled as first point in the search space. If we restrict our attention

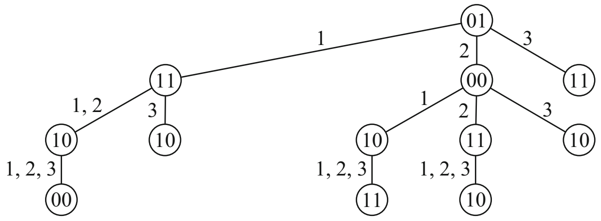


Fig. 5.2 Compact description of the deterministic black-box algorithm for some $\mathcal{F} \subseteq \{f : S \to \mathbb{R}\}$ with $S = \{00, 01, 10, 11\}$ and $\{f(x) \mid f \in \mathcal{F}, x \in S\} = \{1, 2, 3\}$ from Fig. 5.1

to cases where algorithms do not resample any point in the search space and $\{f(x) \mid f \in \mathcal{F}, x \in S\}$ is finite, we can use a well-known trick from complexity theory [17] to have a very similar representation. In formal settings algorithms are often described using Turing machines as model of computation. Randomized algorithms are then described as Turing machines with additional tape where uniformly distributed random bits can be read. It is easy to "move the randomness to the beginning of the computation" in the following way. In the beginning the Turing machine copies a sufficiently large number of random bits to a working tape. After this initialization it proceeds deterministically. Any time that original Turing machine accesses the tape with random bits the modified Turing machine accesses the copies of the random bits. Clearly, this does not change anything. Moreover, we see that a randomized algorithm can be described as making a (possibly very complicated) random decision in the very beginning and proceeding deterministically afterwards. Thus, we can describe a randomized black-box algorithm as a probability distribution over deterministic black-box algorithms. Note that requiring that S and $\{f(x) \mid f \in \mathcal{F}, x \in S\}$ be finite is necessary. Otherwise the number of random bits that are sufficient for the computation may not be known in advance.

5.3 Black-Box Complexity and Practice

Our first concrete example highlights the difference between algorithmic complexity and black-box complexity. Remember that we do not restrict the computational power of black-box algorithms in any way and only take into account the number of function evaluations. Sometimes this can make an important difference. We consider the search space $S = \{0, 1\}^n$ and the class of functions

$$\mathcal{F} = \left\{ f(x) = w_0 + \sum_{i=1}^{n} w_i x[i] + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} w_{i,j} x[i]x[j] \mid \right.$$
$$\left. \forall\, i, j \in \{0, 1, \ldots, n\} : w_i, w_{i,j} \in \mathbb{R} \right\},$$

i.e. the class of polynomials over $\{0, 1\}^n$ of degree at most 2. The best-known NP-complete problem is the satisfiability problem, SAT. In SAT we are given a set of n variables, $x_1, x_2, \dots, x_n$, and a set of m clauses, i.e. disjunctions of some variables and negated variables. The problem is to decide whether there is an assignment of the n variables that satisfies all m clauses simultaneously. The optimization variant, MAX-SAT, asks for an assignment that satisfies as many clauses simultaneously as possible. Of course, MAX-SAT is NP-hard. It remains NP-hard if we restrict the length of each clause to 2, i.e. each clause may contain at most two (negated) variables [6]. The key observation is that this restricted problem, MAX-2-SAT, is contained in $\mathcal{F}$. Consider, for example, the clause $x_3 \vee \overline{x_7}$. It is easy to verify that the polynomial $1 - x_7 + x_3x_7$ assumes the value 0 for $x_3 = 0, x_7 = 1$ and the value 1 for all other assignments of the two variables. Thus, it coincides in value with the clause. Moreover, it has degree 2. Clearly, using such a polynomial for each of the m clauses and adding up these polynomials yields a polynomial of degree at most 2 that yields as value the number of satisfied clauses. This is a polynomial reduction of MAX-2-SAT to the optimization of $\mathcal{F}$ and, thus, optimization of $\mathcal{F}$ is also NP-hard. The black-box complexity of $\mathcal{F}$, however, is rather small. Consider the following algorithm A:

```
1. For $i := 1$ to $n$ do { $x[i] := 0$ }
2. $v_0 := f(x)$; $m := v_0$; $y := x$
3. For $i := 1$ to $n$ do { $x[i] := 1$; $v_i := f(x) - v_0$; $x[i] := 0$ }
4. For $i := 1$ to $n - 1$ do {
      For $j := i + 1$ to $n$ do {
         $x[i] := 1$; $x[j] := 1$; $v_{i,j} := f(x) - v_i - v_j - v_0$; $x[i] := 0$;
         $x[j] := 0$ } }
5. For all $x \in \{0, 1\}^n$ do {
      $v := v_0 + \sum_{i=1}^{n} v_i x[i] + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} v_{i,j} x[i]x[j]$
      If $v > m$ Then $m := v$; $y := x$ }
6. Compute $f(y)$.
```

We claim that A is a black-box algorithm for $\mathcal{F}$. First, we count the number of f-evaluations. There is one f-evaluation in line 2, n in line 3, $\binom{n}{2} = n(n - 1)/2$ in line 4 and one more in line 6. Thus, algorithm A performs exactly $(n^2 + n + 4)/2$ f-evaluations. Since we know that $f \in \mathcal{F}$ holds, we have

$$f(x) = w_0 + \sum_{i=1}^{n} w_i x[i] + \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} w_{i,j} x[i]x[j]$$

for unknown weights w. Thus, we have $f(0^n) = w_0$ and see that $v_0 = w_0$ holds after line 2. In the same way, we have $f(0^{i-1}10^{n-i}) = w_0 + w_i$ and see that $v_i = w_i$ holds for all $i \in \{1, \dots, n\}$ after line 3. Finally, we have $f(0^{i-1}10^{j-i-1}10^{n-j}) = w_0 + w_i + w_j + w_{i,j}$ and $v_{i,j} = w_{i,j}$ holds for all i, j after line 4. Thus, algorithm A manages to learn the objective function within $(n^2 + n + 4)/2$ function evaluations.

We now see the idea in algorithm A clearly. It makes use of function evaluations not to actually locate an optimal search point (but this may still happen accidentally) but to reconstruct the problem instance at hand. After that it can compute the function value of f itself without needing access to the oracle any more. This is what happens in line 5. The maximal function value is stored in m and the according search point is stored in y without any actual function evaluation of f. Thus, in line 6, a global optimum of f can be sampled. We see that $T_{A,\mathcal{F}} \leq (n^2 + n + 4)/2$ holds and have $B_{\mathcal{F}} \leq (n^2 + n + 4)/2 = O(n^2)$ as immediate consequence. This holds in spite of $\mathcal{F}$ being NP-hard to optimize. We see that black-box complexity can be misleading if a black-box algorithm has computational effort that is not polynomially bounded in the number of function evaluations. Since randomized search heuristics are usually algorithmically very simple this is not an issue. Moreover, more restrictive definitions of black-box complexity allow for the derivation of more realistic results for more restricted classes of algorithms [12]. In any case, black-box complexity always yields correct lower bounds.

Bounds obtained by means of complexity theory tend to be small. This is also true for black-box complexity. One may wonder what the point of analyzing black-box complexity is. Analyses of concrete algorithms on specific problems typically yield more precise results for the algorithm under consideration. Such an analysis, however, can never deliver the kind of general result that we obtain by means of black-box complexity. There are three main motivations for studying the black-box complexity of problems. First, black-box complexity can tell us if we already have found an optimal algorithm for a problem. If the upper bound obtained in a specific analysis matches the black-box complexity, the search for a better algorithm can stop since we know that there is no better algorithm. While rare, we present examples in the following where this is the case. The second major advantage of black-box complexity is to inform us about the intrinsic difficulty of a problem. We can learn if a problem is actually hard for all algorithms. Finally, results on the black-box complexity of problems can yield additional insight in the structure of the problem that can guide the search for more efficient algorithms.

5.4 Bounds for Generic Classes of Functions

The black-box complexity $B_{\mathcal{F}}$ is always bounded below by 1 since each black-box algorithm needs to sample at least a global optimum for any objective function due to Definition 5.3. On the other hand, the black-box complexity $B_{\mathcal{F}}$ is always bounded above by $|S|$ since sampling each point in the search space implies that an optimum has also been sampled. While these two observations are very obvious non-trivial results about black-box complexity can also be obtained. We start with a few general observations at the end of this section and a few simple concrete results in the next section. Before we do this we take a closer look at black-box algorithms that will help us to develop a more concrete understanding and prove lower bounds on the black-box complexity of various function classes. The idea is to develop a deeper understanding of black-box complexity, not to develop realistic black-box algorithms.

In the following, we make very general observations that hold for arbitrary function classes $\mathcal{F}$. The statements we make only depend on the size of the function class $|\mathcal{F}|$ or the size of the search space $|S|$. Note that this is different in spirit from NFL results (see Chap. 1). There an assumption about the structure of $\mathcal{F}$, being closed under permutations of the search space, needs to be made. The statements in this section are independent of such structural assumptions.

Using the notion of black-box complexity and our simple description of black-box algorithms, a few observations can be made. We already argued that $1 \leq B_{\mathcal{F}} \leq |S|$ holds. In fact, it is not difficult to prove stronger upper bounds on the black-box complexity for arbitrary function classes. We begin with a simple observation that is even independent of the concrete finite search space S.

Theorem 5.1. *Let $\mathcal{F}$ be a finite set of functions. $B_{\mathcal{F}} \leq (|\mathcal{F}| + 1)/2$.*

Proof. Since $\mathcal{F}$ is finite we have $\mathcal{F} = \{f_1, f_2, \ldots, f_s\}$ for some fixed $s \in \mathbb{N}$. Let x_i^* be a global optimum of f_i for $i \in \{1, 2, \ldots, s\}$. First consider the deterministic black-box algorithm A that samples x_1^*, x_2^*, …, x_s^* in this fixed ordering. Obviously, we have $\mathrm{E}(T_{A,f_i}) = i$ for all $i \in \{1, 2, \ldots, s\}$ and $T_{A,\mathcal{F}} = s$ follows. Since the black-box complexity of $\mathcal{F}$ is defined as $B_{\mathcal{F}} = \inf\{T_{A,\mathcal{F}} \mid A \text{ black-box algorithm for } \mathcal{F}\}$, $T_{A,\mathcal{F}} = s$ imposes s as an upper bound and $B_{\mathcal{F}} \leq s = |\mathcal{F}|$ follows. We can improve on this by considering a randomized black-box algorithm for $\mathcal{F}$. This algorithm samples x_1^*, x_2^*, …, x_s^* in an ordering selected uniformly at random. It samples the global optimum of $f = f_i$ in the tth step with probability $1/s$ for any i and t. Thus

$$T_{A,\mathcal{F}} \leq \sum_{t=1}^{s} t \cdot \frac{1}{s} = \frac{1}{s} \cdot \frac{s(s+1)}{2} = (s+1)/2 = (|\mathcal{F}| + 1)/2$$

holds. □

For an upper bound on $B_{\mathcal{F}}$ it suffices to have an upper bound on $T_{A,\mathcal{F}}$ for any specific black-box algorithm A for $\mathcal{F}$. For the proof of Theorem 5.1 a very simple black-box algorithm was sufficient. Considering another extremely simple randomized black-box algorithm, another upper bound can be proved.

Theorem 5.2. *Let S be a finite search space and $\mathcal{F} \subseteq \{f : S \to \mathbb{R}\}$ a set of functions. $B_{\mathcal{F}} \leq (|S| + 1)/2$.*

Proof. We consider the randomized black-box algorithm A that samples all $|S|$ points of S in a ordering that is determined uniformly at random at the beginning of the computation. This is feasible since S is finite. Since the ordering is determined uniformly at random we have for any $x \in S$ and any $t \in \{1, 2, \ldots, |S|\}$ that A samples x as tth point in the search space equals $1/|S|$. Clearly, any objective function $f \in \mathcal{F}$ has at least one global optimum x^*. We have that the expected position of x^* in the sequence of points sampled by A equals

$$\sum_{t=1}^{|S|} t \cdot \frac{1}{|S|} = \frac{1}{|S|} \cdot \frac{|S| \cdot (|S|+1)}{2} = \frac{|S|+1}{2}.$$

Since this holds for any function $f \in \mathcal{F}$ we have $\mathrm{E}(T_{A,\mathcal{F}}) \leq (|S|+1)/2$ and the same upper bound on $B_{\mathcal{F}}$. □

In complexity theory it is sometimes the case that removing items from a set can increase the complexity of this set. In black-box complexity, however, this cannot happen.

Theorem 5.3. *Let $\mathcal{F} \subseteq \{f : S \to \mathbb{R}\}$ and $\mathcal{F}' \subseteq \{f : S \to \mathbb{R}\}$ be two sets of functions. $(\mathcal{F} \subseteq \mathcal{F}') \Rightarrow (B_{\mathcal{F}} \leq B_{\mathcal{F}'})$.*

Proof. By definition we have $B_{\mathcal{F}} = \inf\{T_{A,\mathcal{F}} \mid A \text{ black-box algorithm for } \mathcal{F}\}$ and $T_{A,\mathcal{F}} = \sup\left\{\mathrm{E}\left(T_{A,f}\right) \mid f \in \mathcal{F}\right\}$. Since $\mathcal{F} \subseteq \mathcal{F}'$ holds the supremum in $T_{A,\mathcal{F}}$ is taken only over $\mathrm{E}\left(T_{A,f}\right)$ that are also considered for the supremum in $T_{A,\mathcal{F}'}$. Thus, $T_{A,\mathcal{F}} \leq T_{A,\mathcal{F}'}$ holds for any black-box algorithm A. Thus, $B_{\mathcal{F}} \leq B_{\mathcal{F}'}$ follows. □

For the proof of more specific results it is useful to have an appropriate tool. A very useful tool for proving lower bounds (not only) in black-box complexity is known as Yao's minimax principle [13, 19].

Theorem 5.4 (Yao's Minimax Principle). *Consider a problem where the set of possible inputs is finite and each input has finite size. Such a problem allows only for a finite number of different deterministic algorithms. Moreover, a randomized algorithm A_q can be described as probability distribution q over the set of deterministic algorithms $\mathcal{A}$.*

For all probability distributions p over the set of inputs $\mathcal{I}$ and all randomized algorithms A_q

$$\min_{A \in \mathcal{A}} E(T(A, I_p)) \leq \max_{I \in \mathcal{I}} E(T(A_q, I))$$

holds.

Yao's minimax principle is a quite direct consequence from game theory. The idea is to consider algorithm design as a two-player zero-sum game, where one player chooses an algorithm and the other chooses an input. Clearly, the player choosing the algorithm wants to minimize the run time while the opponent aims at maximizing it. The importance of Yao's minimax principle becomes clearer if we rephrase its meaning in a less formal way. Lower bounds on the worst-case optimization time of randomized algorithms can be proved by proving lower bounds on the expected optimization time of optimal deterministic algorithms, where the expectation is taken over a distribution over the inputs. The strength of this result has two sources. First, it is much simpler to analyze deterministic algorithms than randomized algorithms (even though considering only optimal deterministic algorithms makes it harder). Second, we are free to choose any distribution over the input that we feel may be difficult.

5.5 Bounds for Typical Benchmark Functions

When theoretically analyzing the performance of randomized search heuristics like evolutionary algorithms, it is common practice to start with proving results for artificial example functions [4]. We will consider three such functions in this section, namely Needle, OneMax, and BinVal. We already know the black-box complexity for each of these functions: It is 1 – like it is for each single function. To be able to derive meaningful results we need to generalize single objective functions to a function class. Whether such a generalization is meaningful depends on the search heuristic we consider. Most randomized search heuristics have the property that they are completely symmetric with respect to the role of 1- and 0-bits. This implies that we can exchange the roles of 1- and 0-bits at each of the n positions without changing the behavior of the randomized search heuristic. Since there 2^n ways of doing this we arrive at a function class of size 2^n, where such randomized search heuristics behave exactly the same for each function in this class. We formalize this idea in the following definition and discuss concrete examples when considering the three different example functions. We make use of the notion $x \oplus y$ for bit strings x and y. For $x, y \in \{0,1\}^n$ let $x \oplus y$ denote the bitwise exclusive or of x and y. For example, $0101 \oplus 1100 = 1001$.

Definition 5.4. For $f\colon \{0,1\}^n \to \mathbb{R}$ and $a \in \{0,1\}^n$, let $f_a(x) := f(x \oplus a)$. Moreover, we define $f^* := \{f_a \mid a \in \{0,1\}^n\}$.

The generalization in Definition 5.4 has been introduced together with the notion of black-box complexity [3]. It is important that it differs from the notion of an orbit [15] that is tied to a specific search operator. This makes this similar to the notion of a fitness landscape [14] that always depends completely on the search operator. The generalization in Definition 5.4 is well-defined independent of any algorithm and operator. However, it only captures the structure of a function in an appropriate way for randomized search heuristics that treat the two different bit values symmetrically. This is still much more general than notions that depend on specific search operators [2].

The generalization in Definition 5.4 is not the only conceivable generalization of this kind. We will discuss another generalization when we observe limitations of what we can be achieved using Definition 5.4.

5.5.1 *Needle*

One well-known example is the function Needle: $\{0,1\}^n \to \mathbb{R}$ with

$$\text{Needle}(x) := \prod_{i=1}^{n} x[i]$$

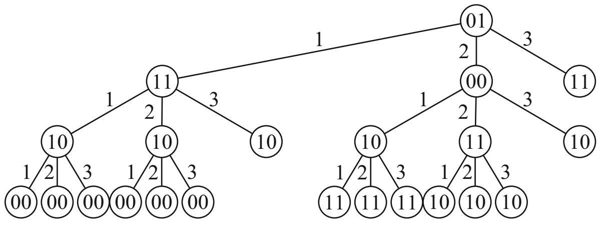


Fig. 5.3 Visualization of the function Needle

Obviously, Needle yields the function value 1 for the all 1 bit string 1^n and the function value 0 anywhere else. Since the number of 1-bits in the all 1 bit strings 1^n is n and since it is strictly smaller for all other bit strings, Needle is a special case of a function where the function value depends only on the number of 1-bits in the bit string and not the specific bit string itself. Such functions are called symmetric and can easily be visualized. The at most $n + 1$ different function values can be plotted over the number of 1-bits as is done in Fig. 5.3 for Needle.

The function Needle is usually considered as a difficult function. In particular, it is known that a very simple evolutionary algorithm, the so-called (1+1) EA, has expected optimization time $\Theta(2^n)$ on Needle [7]. In order to prove that Needle is actually a hard function we would like to show that it has large black-box complexity. However, $B_{\{\text{Needle}\}} = 1$ holds (Theorem 5.1). Obviously, an algorithm that samples 1^n is a black-box algorithm for {Needle} and achieves this bound. For a stronger result we need to consider Needle* instead as defined by means of Definition 5.4. We consider Needle* and discuss its meaning prior to determining the black-box complexity. The idea of Needle is to have a flat fitness landscape with a single spike that may be anywhere. It should not matter to have it at 1^n. Having the spike at an arbitrary position is exactly captured by $\text{Needle}^* = \{\text{Needle}_a \mid a \in \{0,1\}^n\}$ with

$$\text{Needle}_a(x) = \begin{cases} 1 & \text{if } x = \overline{a} \\ 0 & \text{otherwise} \end{cases}$$

where $\overline{a}$ denotes the bitwise complement of a. Now we can move on to establish an exact bound of the black-box complexity for Needle*. We achieve this by applying Yao's minimax principle.

Theorem 5.5. $B_{Needle^*} = (2^n + 1)/2$

Proof. All functions Needle_a are defined on the search space $S = \{0,1\}^n$ and $|S| = 2^n$ holds. Thus, $B_{\text{Needle}^*} \leq (2^n + 1)/2$ is a direct consequence of Theorem 5.2.

We prove a lower bound on B_{Needle^*} by applying Yao's minimax principle (Theorem 5.4). We choose the uniform distribution over Needle* as probability distribution. We need to consider an optimal deterministic black-box algorithm A

for Needle* and prove a lower bound on its expected performance. A deterministic black-box algorithm for Needle* is a tree that contains at least 2^n nodes. If there are less than 2^n nodes than there is at least one $x \in \{0,1\}^n$ that is not the label of any node of A. Clearly, this x is never sampled by A. But x is the unique global optimum of $\text{Needle}_{\overline{x}} \in \{\text{Needle}^*\}$. So, A does not optimize all functions in $\{\text{Needle}^*\}$ and is therefore not a black-box algorithm for $\{\text{Needle}^*\}$. We conclude that A contains at least 2^n nodes.

We have $|\{f(x) \mid f \in \text{Needle}^*, x \in \{0,1\}^n\}| = 2$ since Needle only yields function values 0 and 1. Clearly, optimal black-box algorithms for Needle* do not sample any more points after sampling one point with function value 1. Thus, each node in A has at most 1 outgoing edge. Therefore, an optimal deterministic black-box algorithm for Needle* is a degenerated tree that equals a chain of 2^n nodes. The expected performance of such an algorithm when the expectation is taken over the uniform distribution equals

$$\sum_{t=1}^{|\{0,1\}^n|} t \cdot \frac{1}{|\{0,1\}^n|} = \frac{1}{|\{0,1\}^n|} \cdot \frac{|\{0,1\}^n| \cdot (|\{0,1\}^n| + 1)}{2} = \frac{2^n + 1}{2}$$

and this establishes the lower bound. □

The result $B_{\text{Needle}^*} = 2^{n-1} + 1/2 = \Theta(2^n)$ is remarkable since it shows that Needle is not a particularly difficult function for evolutionary algorithms. The performance of very simple evolutionary algorithm, $\Theta(2^n)$ [7], is in fact asymptotically optimal. Thus, Needle is an example for an intrinsically difficult function where evolutionary algorithms perform particularly well. Note that this is the case even for a very simple evolutionary algorithm that reevaluates many search points and we take this reevaluations into account when assessing its performance.

It is interesting to note that the bound on the black-box complexity of Needle* could have been established in a much simpler way. It suffices to note that Needle* is closed under permutations of the search space. Permuting any Needle_a can only result in another function with exactly one point yielding function value 1 and all other points having function value 0, i.e. a function $\text{Needle}_{a'}$ for some $a' \in \{0,1\}^n$. Due to the sharpened NFL theorem (see Theorem 1.2 in Chap. 1) we know that all algorithms are guaranteed to have equal performance on Needle*. Thus, we could simply analyze an arbitrary black-box algorithm for Needle*. This, of course, simply yields the same calculation as in the proof of Theorem 5.2 (where we considered random enumeration) and Theorem 5.5 (where a deterministic algorithm is considered). Black-box complexity, however, is in no way restricted to classes of functions that are closed under permutations of the search space. Thus, we can prove non-trivial results for classes of functions that are not closed under permutations. We consider such examples in the following.

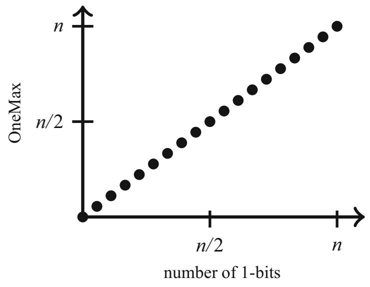


Fig. 5.4 Visualization of the function OneMax

5.5.2 *OneMax*

One of the best studied example functions in the context of evolutionary algorithms and other randomized search heuristics [4,9,20] is OneMax: $\{0,1\}^n \to \mathbb{R}$ that yields as function value simply the number of 1-bits, $\text{OneMax}(x) = \sum_{i=1}^{n} x[i]$. Clearly, OneMax is also a symmetric function and can be visualized as introduced for Needle (see Fig. 5.4).

Since the function value strictly increases with the number of 1-bits the unique global optimum, the all 1 bit string 1^n, is easy to find. Thus, for example, evolutionary algorithms [4], simulated annealing [9], and appropriately configured artificial immune systems [21] have no problems at all locating it, a very simple evolutionary algorithm requires on average $\Theta(n \log n)$ function evaluations to achieve this [4].

Clearly, it makes no sense to investigate the black-box complexity of OneMax. As for all single functions, $B_{\{\text{OneMax}\}} = 1$ holds. Thus, we need to identify a class of functions the captures the idea of OneMax. We consider OneMax to be a function with a unique global optimum where the function value for any $x \in \{0,1\}^n$ is decreased from its maximal value n by the Hamming distance between x and the unique global optimum. This idea is exactly captured by $\text{OneMax}^* = \{\text{OneMax}_a \mid a \in \{0,1\}^n\}$. For OneMax^* no results can by obtained by means of NFL results since OneMax^* is not closed under permutations of the search space. If it was, all black-box algorithms for OneMax^* would have equal performance. Random enumeration of the search space takes on average $2^{n-1} + 1/2$ function evaluations for optimization of OneMax^* as it does for all functions $f\colon \{0,1\}^n \to \mathbb{R}$ with unique global optimum. Simple evolutionary algorithms, however, optimize OneMax^* in $\Theta(n \log n)$ function evaluations. Thus, OneMax^* cannot be closed under permutation of the search space. By analyzing its black-box complexity we see that $\Theta(n \log n)$ is off from optimal performance only by a factor of $\Theta(\log^2 n)$ so that evolutionary algorithms are rather efficient on OneMax^*.

Theorem 5.6. $B_{\mathit{OneMax}^*} = \Theta(n/\log n)$

Proof. We begin with the proof of the lower bound $B_{\text{OneMax}^*} = \Omega(n/\log n)$ and apply Yao's minimax principle. As in the proof of Theorem 5.5 we consider the uniform distribution. The unique global optimum of OneMax_a is $\overline{a}$. Thus, for each $x \in \{0,1\}^n$ there is exactly one function in OneMax^* that has x as its unique global optimum. We conclude that each black-box algorithm for A contains at least 2^n nodes. We have $|\{f(x) \mid f \in \text{OneMax}^*, x \in \{0,1\}^n\}| = n + 1$ since each function OneMax_a yields exactly all function values from $\{0, 1, \ldots, n\}$. We can conclude that each node in the tree of an optimal deterministic black-box algorithm for OneMax^* has degree at most n. The expected number of function evaluations of such an algorithm equals the average depth of this tree. It is well known that the average depth of a tree with 2^n nodes and degree bound n is $\Omega(\log_n 2^n) = \Omega(n/\log n)$ [11]. This establishes $B_{\text{OneMax}^*} = \Omega(n/\log n)$.

The proof of the upper bound $B_{\text{OneMax}^*} = O(n/\log n)$ is due to Anil and Wiegand [1]. We consider the following algorithm A that keeps track of all candidate objective functions (having at all times $F = \{a \in \{0,1\}^n \mid \text{OneMax}_a$ may be the objective function$\}$) and stops when $|F| = 1$ holds.

```
1. $F := \{0,1\}^n$; $S := \emptyset$
2. While $|F| > 1$ do
3.     Select $x \in \{0,1\}^n \setminus S$ uniformly at random. $S := S \cup \{x\}$
4.     $v := f(x)$
5.     $F := F \setminus \{a \mid \text{OneMax}_a \neq v\}$
6. Choose $a$ such that $F = \{a\}$. Compute $f(\overline{a})$.
```

We claim that A is a black-box algorithm for OneMax^*. The idea is to keep track of all functions OneMax_a that cannot yet be ruled out being the unknown objective function $f \in \text{OneMax}^*$. It is known that there is some $a^* \in \{0,1\}^n$ such that $f = \text{OneMax}_{a^*}$ holds. Initially all 2^n different $a \in \{0,1\}^n$ are possible and are thus stored in F. The set S helps to keep track of points already sampled and thus helps to avoid resampling. The main loop (lines 2–5) ends when only one possible $a \in \{0,1\}^n$ is left. Clearly, if the elimination process is correct we have $a = a^*$. Remember that the unique global optimum of OneMax_a is $\overline{a}$. Thus, given the correctness of the elimination process in line 6, the unique global optimum of the objective function f is sampled. The elimination process in the main loop (lines 2–5) works as follows. Some still unseen $x \in \{0,1\}^n$ is selected randomly, the set of seen points updated accordingly (line 3), and the objective function is sampled (line 4). In line 5 all $a \in F$ such that OneMax_a differs in function value from f at x are eliminated from F. Clearly, a^* cannot be eliminated this way and remains in F. Moreover, for each $a \neq a^*$ there is at least one $x \in \{0,1\}^n$ such that $\text{OneMax}_a(x) \neq \text{OneMax}_{a^*}(x)$ holds. For example, this is the case for $x = \overline{a}$ since $\overline{a}$ is the unique global optimum of OneMax_a but not of OneMax_{a^*} so that $\text{OneMax}_{a^*}(x) < \text{OneMax}_a(x) = n$ holds. Thus, F will be eventually reduced to just a^*, and we conclude that A is indeed a black-box algorithm for OneMax^*. We remark that A is not efficient with respect to computational effort. However, we are only concerned with the number of function evaluations.

We need an upper bound on the expected number of function evaluations A makes before identifying the unknown objective function $f = \text{OneMax}_{a^*}$. Consider some fixed $a \in F$ with $a \neq a^*$. Let d denote the Hamming distance between a and a^*, $d := \text{H}(a, a^*)$. We select some $x \in \{0,1\}^n$ uniformly at random and are interested in $\text{Prob}(\text{OneMax}_a(x) = \text{OneMax}_{a^*}(x))$. This is the probability that a cannot be ruled out as objective function by sampling $f(x)$. We have

$$\text{OneMax}_a(x) = \sum_{i=1}^{n} x[i] \oplus a[i] = \sum_{i=1}^{n} x[i] + a[i] - 2x[i]a[i] = \text{H}(a, x)$$

and call $x[i] + a[i] - 2x[i]a[i]$ the contribution of $a[i]$ to the function value $\text{OneMax}_a(x)$. There are $n - d$ bits where a and a^* are equal. Thus, the contribution of these $n - d$ bits is equal for a and a^*. We have $\text{OneMax}_a(x) = \text{OneMax}_{a^*}(x)$ if and only if the contribution of the other d bits is equal for a and a^*, too. Let $a_d, a_d^*, x_d \in \{0,1\}^d$ denote these n bits in a, a^*, and x, respectively. The contribution of a_d equals $\text{OneMax}_a(x_d) = \text{H}(a_d, x_d)$, while the contribution of a_d^* equals $\text{OneMax}_{a^*}(x_d) = \text{H}(a_d^*, x_d)$. Note that $a_d^* = \overline{a_d}$ holds. Thus, we have $\text{OneMax}_a(x) = \text{OneMax}_{a^*}(x)$ if and only if $\text{H}(a_d, x_d) = \text{H}(\overline{a}_d, x_d)$ holds. We have $\text{H}(\overline{a}_d, x_d) = d - \text{H}(a_d, x_d)$ and thus require $\text{H}(a_d, x_d) = d/2$ for $\text{OneMax}_a(x) = \text{OneMax}_{a^*}(x)$ to hold. We conclude that for d odd we have $\text{Prob}(\text{OneMax}_a(x) = \text{OneMax}_{a^*}(x)) = 0$ since $d/2 \notin \mathbb{N}$ in this case. For even d we can pick $d/2$ positions among the d positions in x_d where x_d and a_d disagree. Thus we have $\text{Prob}(\text{OneMax}_a(x) = \text{OneMax}_{a^*}(x)) = \binom{d}{d/2} \cdot 2^{-d}$ in this case. We conclude that

$$\text{Prob}(\text{OneMax}_a(x) = \text{OneMax}_{a^*}(x)) \le \frac{\binom{d}{d/2}}{2^d} \le \sqrt{\frac{1}{d}}$$

holds in both cases where the last inequality follows from application of Stirling's formula for $(d!)$.

We summarize what we have so far. For all $a \neq a^*$ we have that a is not removed from F with probability at most $\sqrt{1/d}$ for $1 < d \le n$ while a is definitely removed for $d = 1$ (remember that $d = \text{H}(a, a^*)$). Thus, on expectation $|F|$ is reduced to $\sqrt{1/d}|F|$ in one execution of the main loop (lines 2–5) of algorithm A. It follows from Markov's inequality that we have the new F with at most $2\sqrt{1/d}|F|$ elements after one round with probability at least $1/2$.

Nothing changes if instead of considering F completely, we partition F into $n - 2$ disjoint sets where the set F_d contains all a with $\text{H}(a, a^*) = d$ (for $d \in \{2, 3, \ldots, n\}$). Remember that we can ignore $d = 0$ and $d = 1$ since a^* is never removed and all a with $\text{H}(a, a^*) = 1$ are removed certainly in the first round.

Now we consider $8n/(\ln(n) - \ln(2))$ rounds of algorithm a. Clearly, in these rounds $O(n/\log n)$ function evaluations are made. In each round each of the sets F_d is reduced to at most $2\sqrt{1/d}|F_d|$ elements with probability at least $1/2$. Applying Chernoff bounds [13], we have at least $2n/(\ln(n) - \ln(2))$ such rounds with probability $1 - e^{-\Omega(n/\log n)}$. The initial size is $|F_d| = \binom{n}{d}$, we thus have with this probability

$$|F_d| \le \left(\frac{2}{\sqrt{d}}\right)^{2n/(\ln(n)-ln(2))} \cdot \binom{n}{d} < 1$$

for each set. Thus, on expectation after $O(n/\log n)$ function evaluations algorithm A samples the unique global optimum and stops. □

Note that the longer part of the proof of Theorem 5.6 is concerned with the upper bound on the black-box complexity. We see that establishing that evolutionary algorithms are quite efficient on OneMax was rather simple since for this the lower bound suffices.

5.5.3 BinVal

Another interesting example function where evolutionary algorithms are similarly efficient is BinVal: $\{0,1\}^n \to \mathbb{R}$ with $\text{BinVal}(x) = \sum_{i=1}^{n} 2^{n-i} x[i]$. The function BinVal yields as function value the number represented by x read as standard binary encoding of a non-negative integer. It is known that evolutionary algorithms are similarly efficient on BinVal as they are for OneMax, in fact, for a very simple evolutionary algorithm the performance is asymptotically equal $\Theta(n \log n)$ [4]. We want to compare this to the black-box complexity. While it is clear that we have $B_{\{\text{BinVal}\}} = 1$, it may come as a surprise that the black-box complexity B_{BinVal^*} is not much larger.

Theorem 5.7. $B_{BinVal^*} = 2 - 2^{-n}$.

Proof. We begin with the proof of a lower bound for B_{BinVal^*}. Since BinVal has a unique global optimum (the all ones bit string) the same holds for any BinVal_a. As for OneMax_a, we have that $\overline{a}$ is the unique global optimum of BinVal_a. Thus BinVal^* contains 2^n different functions with 2^n different global optima. We conclude that for any black-box algorithm for BinVal^* the probability to be successful with the very first point it samples is bounded above by 2^{-n} in the worst case (where the probability is taken over the random choices of the algorithm). The best a black-box algorithm can achieve is to sample the unique global optimum with probability 1 after at most two sampled points. The average number of samples of such a hypothetical black-box algorithm for BinVal^* would be

$$2^{-n} \cdot 1 + (1 - 2^{-n}) \cdot 2 = 2 - 2^{-n}$$

and is a lower bound on B_{BinVal^*}.

For the upper bound we prove that such a black-box algorithm for BinVal^* is not merely hypothetical. Consider the following algorithm A.

1. Select $x \in \{0,1\}^n$ uniformly at random. $v := f(x)$

```
2. For $i := 1$ to $n$ do
       $a[n+1-i] := v \bmod 2$; $v := \lfloor v/2 \rfloor$
3. Compute $f(\overline{x \oplus a})$.
```

We observe that A performs at most two function evaluations. According to Definition 5.3 the second function evaluation (in line 3) is not taken into account if the first function evaluation (in line 1) has already sampled the unique global optimum of f. Since x is chosen uniformly at random this happens with probability 2^{-n}. We see that algorithm A performs in expectation $2 - 2^{-n}$ function evaluations. What needs to be shown is that algorithm A actually is a black-box algorithm for BinVal*.

We have $f = \text{BinVal}_{a^*}$ for some unknown $a^* \in \{0,1\}^n$. Thus,

$$v = f(x) = \text{BinVal}_{a^*}(x) = \sum_{i=1}^{n} 2^{n-i}(x[i] \oplus a^*[i])$$

holds after the first function evaluation. We see that $v \in \{0,1,\ldots,2^n\}$ is a non-negative integer that is represented by $x \oplus a^*$ in standard binary encoding. This standard binary encoding is computed and stored in a in line 2. Thus, we have $a = x \oplus a^*$ after line 2. We conclude that $x \oplus a = x \oplus x \oplus a^* = a^*$ holds. Thus, in line 3 the function value of $\overline{a^*}$, the unique global optimum of f, is sampled. Therefore, algorithm A is a black-box algorithm for BinVal* and establishes $B_{\text{BinVal}^*} \leq 2 - 2^{-n}$. □

We see the reason for the extremely small black-box complexity of BinVal* in the proof of Theorem 5.7. The position of the unique global optimum is given away with a single function value. We see that the 2^n different function values that BinVal yields give away much more information than the $n+1$ different function values that OneMax yields. Randomized search heuristics, however, typically do not exploit function values to this degree of detail. In particular, many evolutionary algorithms are completely oblivious with respect to the true function values and are only sensitive with respect to the ordering of the function values. One may argue that for these evolutionary algorithms the class of functions BinVal* does not adequately capture the idea of BinVal. For these evolutionary algorithms it makes more sense to consider the class of functions

$$\text{BinVal}^{**} := \{f \circ g \mid g \in \text{BinVal}^*, g\colon \mathbb{R} \to \mathbb{R} \text{ strictly increasing}\}$$

that "hides" the function values but preserves the ordering. Note that BinVal** is not finite and thus Yao's minimax principle (Theorem 5.4) cannot be applied. Yet, $B_{\text{BinVal}^{**}} = \Omega(n/\log n)$ can be shown with similar techniques not very different from the proof of the lower bound on B_{OneMax^*} [5].

The proofs of lower bounds on the black-box complexity we presented so far have all been concerned with classes of functions that are based on one single objective function. Capturing the idea of this objective function in a way that lets the randomized search heuristic of interest performs provably equally on all members of this function class, we have been able to prove non-trivial lower bounds on the black-box complexity. This way we can compare the performance of the randomized search heuristic we are interested in with the lower bound on the black-box complexity and see if there can be much more efficient algorithms. This is a useful approach since randomized search heuristics are often analyzed on such single objective functions. But there can be no doubt that it would also be very useful to have non-trivial lower bounds on the black-box complexity of more natural classes of functions. This can also be achieved as the following two examples show.

5.6 Bounds for Natural Classes Functions

5.6.1 Monomials

We begin with a simple example inspired by the class of polynomials of degree at most 2 that we considered as our very first example. A conjunction of several variables and negated variables is called a monomial, i.e. a polynomial with just one term. The number of variables in a monomial is called its degree. Let M_d denote the class of monomials $m\colon \{0,1\}^n \to \mathbb{R}$ of degree at most d.

Theorem 5.8. $2^{d-1} + 1/2 \leq B_{M_d} \leq 2^d$.

Proof. Consider some monomial $m \in M_d$ with $d' \leq d$ variables. There is exactly one setting of the d' variables involved in m that yields function value 1; the other $2^{d'} - 1$ settings yield value 0. Clearly, the setting of the other $n - d'$ variables has no influence at all. Thus, there are $2^{n-d'}$ settings of the n variables that yield function value 1; all other settings yield function value 0. When sampling the search space $\{0,1\}^n$ uniformly at random, each sample finds one of the $2^{n-d'}$ optimal settings with probability $2^{n-d'}/2^n \geq 2^{-d}$. Thus, the worst-case expected optimization time of pure random sampling on M_d is bounded above by 2^d. This establishes $B_{M_d} \leq 2^d$.

For the lower bound we consider the set

$$M'_d = \{z_1 z_2 \cdots z_d \mid \forall\, i \in \{1,2,\ldots,d\}\colon z_i \in \{x_i, 1-x_i\}\}\,.$$

Clearly, $M'_d \subseteq M_d$ holds. This implies $B_{M'_d} \leq B_{M_d}$ (Theorem 5.3) and it suffices to prove a lower bound on $B_{M'_d}$. Moreover, we have $M'_d = \text{Needle}^*$ for the function Needle: $\{0,1\}^d \to \mathbb{R}$. Thus, $B_{M'_d} = 2^{d-1} + 1/2$ holds (Theorem 5.5). □

5.6.2 Unimodal Functions

Clearly, the class of monomials of degree at most d is not a very interesting class of functions. But we can also obtain results on the black-box complexity of practically relevant function classes like the class of unimodal problems. Consider some objective function $f\colon \{0,1\}^n \to \mathbb{R}$. We call two points $x, y \in \{0,1\}^n$ *Hamming neighbors* (or *neighbors* for short) if they differ in exactly 1 bit, i.e. their Hamming distance equals one, $\mathrm{H}(x,y) = 1$. We call a point $x \in \{0,1\}^n$ a *local optimum* of f if no Hamming neighbor of x has larger function value. We call f *unimodal* if f has exactly one local optimum. We call f *weakly unimodal* if every local optimum of f is also a global optimum of f. Let

$$\mathcal{U} = \{f\colon \{0,1\}^n \to \mathbb{R} \mid f \text{ weakly unimodal}\}$$

denote the class of all weakly unimodal problems.

Weakly unimodal objective functions have the property that there exist paths leading to a global optimum. If $f \in \mathcal{U}$ then for all $x \in \{0,1\}^n$ we have that either x is a global optimum or there is a Hamming neighbor y of x such that $f(y) > f(x)$ holds. This implies that local search is guaranteed to find a global optimum of any $f \in \mathcal{U}$ in finite time regardless of the starting point. One may be tempted to believe that this means that weakly unimodal functions are in some sense easy to optimize. We can prove rigorously that this is not true by means of black-box complexity. We prove an exponential lower bound on $B_{\mathcal{U}}$. More specifically, we identify a subclass $\mathcal{F} \subseteq \mathcal{U}$ and prove that $B_{\mathcal{F}} > 2^{n^\delta}$ holds for any constant $\delta < 1$. Since $\mathcal{F} \subseteq \mathcal{U}$ implies $B_{\mathcal{F}} \leq B_{\mathcal{U}}$ (Theorem 5.3), we have an exponential lower bound on the black-box complexity of weakly unimodal functions this way.

Theorem 5.9. *$\forall$ constants $\delta < 1$: $B_{\mathcal{U}} > 2^{n^\delta}$.*

Proof. We call a sequence of points $P = (x_1, x_2, \ldots, x_l)$ a *path* of length l if for all $i \in \{1, 2, \ldots, l-1\}$ we have that x_i and x_{i+1} are Hamming neighbors. We call P a *simple path* if the points of P are pairwise disjoint, i.e. $|\{x_1, x_2, \ldots, x_l\}| = l$.

We restrict our attention to paths P that start in the all 1 bit string, i.e. $P = (x_1 = 1^n, x_2, \ldots, x_l)$. Given such a path P we define its path function f_P by

$$f_P(x) = \begin{cases} n + i & \text{if } i = \max\{j \in \{1, 2, \ldots, l\} \mid x = x_j\}, \\ n - \mathrm{OneMax}(x) & \text{otherwise.} \end{cases}$$

For $x \in \{0,1\}^n$ not on the path P the function value $f_P(x)$ equals the Hamming distance between x and the starting point of the path. For path points $x \in P$ there may be several $i \in \{1, 2, \ldots, l\}$ such that $x = x_i$ holds (unless P is a simple path). For such points the function value $f_P(x)$ is by n larger than the position of point $x_i = x$ with largest index i. Clearly, f_P is unimodal and has x_l as its unique global optimum.

Such paths $P = (x_1 = 1^n, x_2, \ldots, x_l)$ can easily be constructed randomly. We start with $i = 1$ in $x_1 = 1^n$. As long as $i < l$ we choose x_{i+1} uniformly at random from the set of the n Hamming neighbors of x_i and increase i to $i + 1$.

We fix an arbitrary constant ε with $\max\{0, \delta\} < \varepsilon < 1$ and define $l := l(n) = 2^{n^\varepsilon}$ for the rest of the proof. With this l we consider the random paths $P = (x_1 = 1^n, x_2, \ldots, x_l)$ as described above. For each P we consider its corresponding path function f_P and collect all these path functions in $\mathcal{F} = \{f_P: \{0,1\}^n \to \mathbb{N} \mid P = (x_1 = 1^n, x_2, \ldots, x_l)\}$. We want to prove $B_{\mathcal{F}} > 2^{n^\delta}$ by means of Yao's minimax principle (Theorem 5.4). To this end we define a probability distribution on $B_{\mathcal{F}}$ by assigning to $f_P \in \mathcal{F}$ the probability by which the path P is created in the random path construction. Now we need to bound by below the number of function evaluations an optimal deterministic black-box algorithm for $\mathcal{F}$ makes on average in the worst case. To achieve this it is useful to have a result on a central property of the random paths P. It can informally be described in the following way. If we make a linear number of steps on such a path it is highly likely that we arrive at a point that has a linear Hamming distance to the point where we started. We make this precise in the following lemma.

Lemma 5.1. *Consider a randomly constructed path* $P = (x_1 = 1^n, x_2, \ldots, x_l)$.
$\forall$ *constants* $\beta > 0$: $\exists$ *constant* $\alpha(\beta) > 0$: $\forall\, j \geq \beta n$:
Prob$\left(H\left(p_i, p_{i+j}\right) \leq \alpha(\beta) n\right) = 2^{-\Omega(n)}$.

Proof. The random path construction works in the same way for any current position i. Thus, it suffices to prove the statement for $i = 1$ and some fixed $j \geq \beta n$ with $i + j < l$ We consider the random Hamming distance from the first path point $H_t := \mathrm{H}(1^n, x_{t+1})$ for all $t \in \{1, 2, \ldots, j\}$. We need to prove $\text{Prob}\left(H_j \leq \alpha n\right) = 2^{-\Omega(n)}$ for some constant α that we are free to choose depending on β.

We have $H_{t+1} \in \{H_t - 1, H_t + 1\}$ since the next point on the path is chosen among the n Hamming neighbors of the current point. Moreover, we have $\text{Prob}(H_{t+1} = H_t - 1) = H_t/n$ since, in order to decrease the Hamming distance by 1, we need to change one of the H_t out of n bits that differ from x_1 in x_{1+t}. Thus, $\text{Prob}(H_{t+1} = H_t + 1) = 1 - H_t/n$ holds.

We define $\alpha := \min\{1/50, \beta/5\}$. Since we have $\beta > 0$, $\alpha > 0$ follows and this is a valid choice of α. Moreover, we define $\gamma := \min\{1/10, j/n\}$. Note that γ is not necessarily a constant since j may depend on n. However, due to its definition we have $5\alpha \leq \gamma \leq 1/10$. We consider the last γn steps of the random path construction until x_j is constructed. It is important to note that $\gamma \leq j/n$ holds. This implies that the last γn steps of the random path construction actually do exist.

We make a case distinction with respect to the value of H_T, where T is the first of these last γn steps we consider. First, consider the case where $H_T \geq 2\gamma n$ holds. We consider γn steps and in each step the value of H_t can decrease by at most one. Thus, at the end we have $H_t \geq 2\gamma n - \gamma n = \gamma n$ with probability 1. We see that if we are sufficiently far away at the beginning of those last steps we cannot be too close at the end.

Now we consider the other case, where $H_t < 2\gamma n$ holds. Again, since we consider only γn steps and in each step the value of H_t can increase by at most one, we have $H_t < 2\gamma n + \gamma n = 3\gamma n$ all the time. This implies that $\text{Prob}(H_{t+1} = H_t + 1) > 1 - 3\gamma \geq 7/10$. We consider a series of γn random experiments with success probability at least $7/10$ in each experiment. We are interested in the number of successes M'. Now consider another random process where we consider a series of γn independent random experiments each with success probability exactly $7/10$. Let M denote the number of successes in this second random process. Clearly, for any $v \in \mathbb{R}$, we have $\text{Prob}\,(M' \geq v) \geq \text{Prob}\,(M \geq v)$. Thus M stochastically dominates M' and we may analyze this second random process instead. Due to the independence of the random experiments we are in a situation where we can apply Chernoff bounds [13]. We have $\text{E}(M) = (7/10)\gamma n$ and get

$$\text{Prob}\,(M < (3/5)\gamma n) = \text{Prob}\,(M < (1-1/7)(7/10)\gamma n) = e^{-\Omega(\gamma n)} = 2^{-\Omega(n)}.$$

With probability $1 - 2^{-\Omega(n)}$ we have at least $(3/5)\gamma n$ steps where H_t is increased among those last γn steps. Thus there can be at most $(2/5)\gamma n$ steps where H_t is decreased. Since H_t is non-negative, we have

$$H_j \geq 0 + (3/5)\gamma n - (2/5)\gamma n = (1/5)\gamma n$$

with this probability $1 - 2^{-\Omega(n)}$. Together, we have with probability $1 - 2^{-\Omega(n)}$ a Hamming distance of at least $(1/5)\gamma n \geq \alpha n$ between x_1 and x_{1+j}. □

Making use of this lemma we can now complete the proof of Theorem 5.9. Our aim is to prove that no deterministic black-box algorithm can make much progress on f_P in any single step. More concretely, we will show that it is highly unlikely to make progress of at least n (with respect to fitness or, equivalently, on the path). Note that this is an even stronger statement.

In a first step we recognize that "shortcuts" on the path may exist. If the path is not a simple path, it may be actually possible to make a rather big step forward on the path by considering a Hamming neighbor. This is the case if this Hamming neighbor is the first and last point of a circle. We solve this problem by conceptually removing all circles. Clearly, this reduces the path length. Using Lemma 5.1 it is easy to see that with probability $1-2^{-\Omega(n)}$ the remaining path length is at least l/n.

In order to simplify the proof, we consider a simplified scenario where we equip the black-box algorithm with additional knowledge and ask for less than optimization. Before the first point is sampled, a black-box algorithm knows that the path starts in $x_1 = 1^n$. In addition to that, we equip it with the knowledge of all path points x_i with $f_P(x_i) \leq f_P(x_1)$ and with $f_P(1^n)$. This corresponds to removing all circles. If there is a circle coming back to the first point x_1, we let the algorithm know everything about this circle. Clearly, this can only improve the algorithm's performance. At any point, i.e. after any number of points sampled, we can describe the algorithm's knowledge in the following way. Among all path points it knows there is some path point x_j with maximal function value $f_P(x_j)$. In addition, we equip the algorithm with knowledge about all path points x_i with $f_P(x_i) \leq f_P(x_j)$. This covers our practice of removing cycles. Moreover, there is a set of points $N \subseteq \{0,1\}^n$ known not to be on the path. Every $x \in N$ has been sampled by the algorithm and could easily be identified as not being a path point by its function value $f_P(x) < n$.

In the beginning we have $j = 1$ and $N = \emptyset$. In each round the algorithm has to decide which point $x \in \{0,1\}^n$ to sample next. Clearly, since we are considering an optimal black-box algorithm this will be some point not sampled before. We distinguish three different cases with respect to this newly sampled point x.

1. **Case: $x \notin P$** In this case we first update N to $N \cup \{x\}$. Moreover, we update j to $j + n$ and additionally equip the algorithm with the knowledge about all path points x_i with $f_P(x_i) \leq f_P(x_j)$ for the new value of j.
2. **Case: $x \in P$ and $f_P(x) < f_P(x_{j+n})$** In this case we again update j to $j + n$ and additionally equip the algorithm with the knowledge about all path points x_i with $f_P(x_i) \leq f_P(x_j)$ for the new value of j. This is similar to the first case.
3. **Case: $x \in P$ and $f_P(x) \geq f_P(x_{j+n})$** In this case we update j to l so that the optimum is found. The algorithm stops successfully in this case.

What we only ask is that an optimal black-box algorithm only makes an advance on the path of at least n in a single function evaluation. If this succeeds, we stop the process and pretend that the optimum was found. Clearly, this way we can at best prove a lower bound of $((l/n) - n)/n = \Theta\left(2^{n^{\varepsilon}}/n^2\right)$. Since we only need to prove a lower bound of $2^{n^{\delta}}$ for some $\delta < \varepsilon$ this is sufficient.

We start with $N = \emptyset$ and $j = 1$. If the algorithm samples some point x with $\mathrm{H}(x, 1^n) \leq \alpha(1)n$ (relatively close to the best known path point 1^n where $\alpha(1)$ is the constant from Lemma 5.1 with $\beta = 1$), we have a success with probability only $2^{-\Omega(n)}$ due to Lemma 5.1. If the algorithm samples some point x far way, i.e. $\mathrm{H}(x, 1^n) > \alpha(1)n$ we consider the random process of path construction as if it had not yet taken place completely. This way we see that the path construction hits exactly some x with $\mathrm{H}(x, 1^n) > \alpha(1)n$ only with probability $2^{-\Omega(n)}$. Thus, the first step is almost completely hopeless as claimed.

In later steps we have $j > 1$ and may have $N \neq \emptyset$. While $j > 1$ is not a big change, having $N \neq \emptyset$ may change things significantly. In order to deal with this we separate the set of known points $N \cup \left\{x_i \mid f_P(x_i) \leq f_P(x_j)\right\}$ into points close $(\{x \in K \mid \mathrm{H}(x, x_j) < \alpha(1/2)n\})$ and far away $(\{x \in K \mid \mathrm{H}(x, x_j) < \alpha(1/2)n\})$. If all known points are far away it is again easy to prove that not much changes using Lemma 5.1. If the set of known points that are close is not empty we make use of a simple trick. We ignore the next $n/2$ steps of random path construction and advance the algorithm by this on the path. After that we are in a situation where all known points are again far away with probability $1 - 2^{-\Omega(n)}$ (Lemma 5.1).

So, we have that in each step the algorithm succeeds with probability $2^{-\Omega(n)}$. Applying the union bound yields that the algorithm succeeds in $2^{n^{\delta}}$ steps with probability at most $2^{n^{\delta}} \cdot 2^{-\Omega(n)} = 2^{-\Omega(n)}$. Thus, we have $B_{\mathfrak{U}} > 2^{n^{\delta}}$ as claimed. □

5.7 Conclusions

Black-box complexity is a very general notion that allows us to establish lower bounds on the difficulty of problems in the black-box scenario that hold for all algorithms, in particular for all randomized search heuristics. It only takes into account the number of times the objective function is evaluated until a global optimum is sampled for the first time. Not requiring that a black-box algorithm must recognize an optimal solution allows for the inclusion of incomplete heuristics. Not taking into account the computational effort of the heuristics allows for the rigorous proof of absolute bounds not depending on any unproven assumptions (like $P \neq NP$). But this comes at the price of delivering unrealistic results when black-box algorithms are excessively expensive. In practice, this is not an issue as randomized search heuristics are almost always simple.

The notion of black-box complexity allows for general observations that rely only on very general properties like the size of the class of functions or the search space. Considering more concrete classes of functions and using Yao's minimax principle as the most important tool, a number of non-trivial lower bounds on the black-box complexity are proven. This holds for results on single example functions, where the key for deriving interesting bounds on the black-box is capturing the idea of the example function in an appropriate class of functions. Such results can prove actual and concrete randomized search heuristics like evolutionary algorithms to be very efficient on such problems. By proving their performance to be close to the black-box complexity, it is proven that no other algorithm can be much more efficient. In addition, non-trivial results for more natural classes of functions like monomials of bounded degree or unimodal functions are proven. It can be expected that black-box complexity will prove a useful concept for the rigorous proof of lower bounds on the difficulty of problems in the future, too.

Acknowledgements This material is based upon work supported by Science Foundation Ireland (SFI) under Grant No. 07/SK/I1205.

References

1. G. Anil, R.P. Wiegand, Black-box search by elimination of fitness functions, in *Proceedings of the Tenth ACM SIGEVO Workshop on Foundations of Genetic Algorithms (FOGA 2009)*, Orlando, ed. by I. Garibay, T. Jansen, R.P. Wiegand, A.S. Wu (ACM, 2009), pp. 67–78
2. Y. Borenstein, R. Poli, Structure and metaheuristics, in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2006)*, Seattle, ed. by M. Keijzer et al. (ACM, 2006), pp. 1087–1094
3. S. Droste, T. Jansen, K. Tinnefeld, I. Wegener, A new framework for the valuation of algorithms for black-box optimization. in *Foundations of Genetic Algorithms 7 (FOGA 2002)*, Torremolinos, ed. by K.A. De Jong, R. Poli, J. Rowe (Morgan Kaufmann, San Francisco, 2003), pp. 253–270
4. S. Droste, T. Jansen, I. Wegener, On the analysis of the (1+1) evolutionary algorithm. Theor. Comput. Sci. **276**, 51–81 (2002)
5. S. Droste, T. Jansen, I. Wegener, Upper and lower bounds for randomized search heuristics in black-box optimization. Theory Comput. Syst. **39**, 525–544 (2006)
6. M.R. Garey, D.S. Johnson, *Computers and Intractability. A Guide to the Theory of NP-Completeness* (Freeman, New York, 1979)
7. J. Garnier, L. Kallel, M. Schoenauer, Rigorous hitting times for binary mutations. Evol. Comput. **7**(2), 173–203 (1999)

8. C. Igel, M. Toussaint, A no-free-lunch theorem for non-uniform distributions of target functions. J. Math. Model. Algorithms **3**(4), 313–322 (2004)
9. T. Jansen, I. Wegener, A comparison of simulated annealing with simple evolutionary algorithms on pseudo-boolean functions of unitation. Theor. Comput. Sci. **386**, 73–93 (2007)
10. D.S. Johnson, A theoretician's guide to the experimental analysis of algorithms, in *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*, ed. by M.H. Goldwasser, D.S. Johnson, C.C. McGeoch (American Mathematical Society, Providence, 2002), pp. 215–250
11. D. Knuth, *The Art of Computer Programming. Volume 3: Sorting and Searching*, 2nd edn. (Addison-Wesley, London, 1997)
12. P.K. Lehre, C. Witt, Black-box search by unbiased variation, in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2010)*, Portland, ed. by M. Pelikan, J. Branke (ACM, 2010), pp. 1441–1448
13. R. Motwani, P. Raghavan, *Randomized Algorithms* (Cambridge University Press, Cambridge, 1995)
14. C.M. Reidys, P.F. Stadler, Combinatorial landscapes. SIAM Rev. **44**(1), 3–54 (2002)
15. J.E. Rowe, M.D. Vose, A.H. Wright, Structural search spaces and genetic operators. Evol. Comput. **12**(4), 461–493 (2004)
16. C. Schumacher, M.D. Vose, L.D. Whitley, The no free lunch and description length, in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)*, San Francisco, ed. by L. Spector et al. (Morgan Kaufmann, 2001), pp. 565–570
17. I. Wegener, *Complexity Theory* (Springer, Berlin, 2005)
18. D.H. Wolpert, W.G. Macready, No free lunch theorems for optimization. IEEE Trans. Evol. Comput. **1**(1), 67–82 (1997)
19. A. Yao, Probabilistic computations: towards a unified measure of complexity, in *Proceedings of the 17 Annual IEEE Symposium on the Foundations of Computer Science (FOCS '77)*, Providence (IEEE, Piscataway, 1977), pp. 222–227
20. C. Zarges, Rigorous runtime analysis of inversely fitness proportional mutation rates, in *Proceedings of the 10th International Conference on Parallel Problem Solving from Nature (PPSN 2008)*, Dortmund, ed. by G. Rudolph, T. Jansen, S. Lucas, C. Poloni, N. Beume (Springer, Berlin, 2008), pp. 112–122
21. C. Zarges, On the utility of the population size for inversely fitness proportional mutation rates, in *Proceedings of the Tenth ACM SIGEVO Workshop on Foundations of Genetic Algorithms (FOGA 2009)*, Orlando, ed. by I. Garibay, T. Jansen, R.P. Wiegand, A.S. Wu (ACM, 2009), pp. 39–46

Chapter 6
Designing an Optimal Search Algorithm with Respect to Prior Information

Olivier Teytaud and Emmanuel Vazquez

Abstract There are many optimization algorithms, most of them with many parameters. When you know which family of problems you face, you would like to design *the* optimization algorithm which is the best for this family (e.g., on average against a given distribution of probability on this family of optimization algorithms). This chapter is devoted to this framework: we assume that we know a probability distribution, from which the fitness function is drawn, and we look for the optimal optimization algorithm. This can be based (i) on experimentations, i.e. tuning the parameters on a set of problems, (ii) on mathematical approaches automatically building an optimization algorithm from a probability distribution on fitness functions (reinforcement learning approaches), or (iii) some simplified versions of the latter, with more reasonable computational cost (Gaussian processes for optimization).

6.1 Introduction

The No Free Lunch (NFL) theorem states that all optimization algorithms (OA) perform equally, on average, on the set of all fitness functions for a given finite domain and a given finite codomain (see e.g. [41]). The NFL, however, does not hold in continuous domains [2]. Even in discrete domains, it is restricted to specific distributions of fitness functions (for example, uniform distribution over all functions from a finite domain to a finite codomain), which are probably far

O. Teytaud (✉)
TAO, Inria Saclay IDF, LRI, University of Paris-Sud, UMR CNRS 8623, France
e-mail: olivier.teytaud@inria.fr

E. Vazquez
Department of Computer Science and Information Engineering, National University of Tainan, Tainan, Taiwan SUPELEC, Gif-sur-Yvette, France
e-mail: emmanuel.vazquez@supelec.fr

from real-world fitness functions [7, 13, 39]. In fact, a given OA will probably have varying performance depending on the type of fitness functions. This raises the two following questions when dealing with an optimization problem:

1. Given prior knowledge about a function to be optimized, which OA should be considered?
2. How to tune the parameters of a given OA?

A simple empirical solution in order to take into account prior knowledge is to compare the performances of different OAs on a given family of test problems, which are deemed to be close to the real optimization problem to be solved. In some cases, the choice of the parameters of an OA (question 2 above) can also be considered as an optimization problem; optimizing the parameters of an OA can lead to major improvements.

Figure 6.1 illustrates three approaches for designing optimal algorithms for a given prior. The first approach [8, 21, 22], already suggested above, consists in optimizing the parameters of an OA. This approach is easy to use, immediately operational and efficient in practice. The second approach [2, 10, 26] consists in considering the problem specified by Eqs. (6.1) and (6.2) and solving it as a Markov decision process. This second approach is mathematically appealing as it provides a provably optimal algorithm; unfortunately, the approach is computationally very expensive and might be difficult to use in practice. Therefore, the third approach considers a criterion that measures the progress of an optimization algorithm (e.g., Eq. (6.12)), and a prior about the function to be optimized under the form of random process (e.g., it assumes that the fitness function is drawn according to a Gaussian process), and proposes an algorithm which is optimal (within the limit of the computational cost) for optimizing the criterion on average on the sample paths of the random process.

The first approach based on testbeds is discussed in Sect. 6.2. Section 6.3 discusses the second approach. The approach will be illustrated on a particular case. Section 6.4 discusses the third approach, and in particular gaussian process optimization, which is relevant for expensive optimization, i.e., when the function to be optimized is expensive by itself. It is then possible to achieve more reasonable computational costs, whilst preserving a much better algorithm than usual techniques for small numbers of function evaluations.

In this chapter, $\#E$ denotes the cardinality of the set E, X_n is the nth visited point of the considered OA, and x^* is (when existing and unique) the global optimum. $\mathbb{E}_a$ denotes the expectation operator, with respect to random variable a. We will restrict our attention to black-box optimization, in which we have no closed-form expression (i.e., fitness functions with no available analytical expression) for the fitness function and for which we cannot use specific algorithms like linear programming, quadratic programming, or shortest path. Our focus is also essentially on functions for which the gradient is not available; discrete optimization is considered as well as continuous optimization.

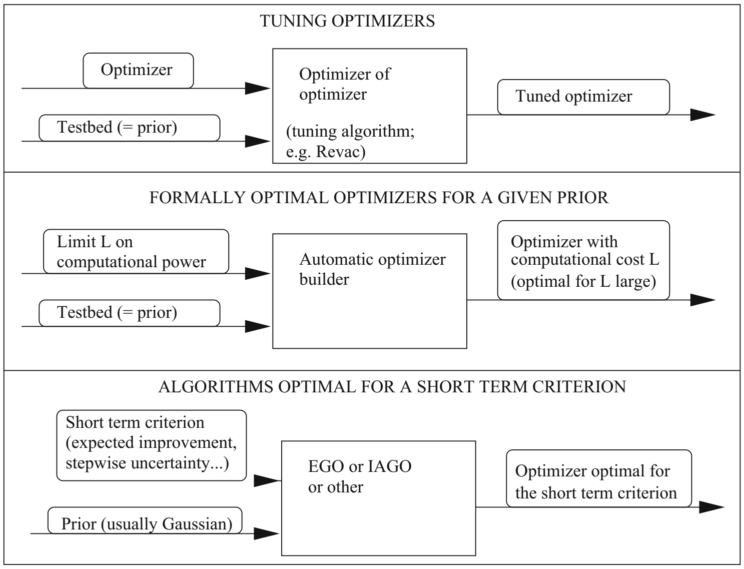

TUNING OPTIMIZERS
Optimizer
Testbed (= prior)
Optimizer of optimizer
(tuning algorithm; e.g. Revac)
Tuned optimizer
FORMALLY OPTIMAL OPTIMIZERS FOR A GIVEN PRIOR
Limit L on computational power
Testbed (= prior)
Automatic optimizer builder
Optimizer with computational cost L (optimal for L large)
ALGORITHMS OPTIMAL FOR A SHORT TERM CRITERION
Short term criterion (expected improvement, stepwise uncertainty...)
Prior (usually Gaussian)
EGO or IAGO or other
Optimizer optimal for the short term criterion

Fig. 6.1 Three approaches for optimization given a prior knowledge (typically a distribution on possible fitness functions). These three approaches are detailed in Sects. 6.2–6.4, respectively

6.2 Testbeds for Black-Box Optimization and Parameter Tuning

The simplest way to solve the main question in this chapter, i.e., which OA should we use for a given probability distribution of optimization problems, is to select a family of OAs, and to test this family of OAs on a family of fitness functions. This does not create new OAs, but it provides a ranking of existing algorithms. It can also be used for tuning OAs, i.e., choosing which values should be assigned to each parameter.

6.2.1 Tuning OAs

It is a common practice to design testbeds for tuning OAs. Given an OA $Optimize_\theta$ parametrized by θ, the idea is to tune θ in order to minimize a loss function which characterizes the performance of $Optimize_\theta$ on a given testbed.

In the unconstrained case, the most widely known test bed is probably Black-Box Optimization Benchmarking (BBOB) [1]; [11], another optimization benchmark, includes also constrained optimization.

For example, in the BBOB optimization test bed, probably the biggest available test bed, the functions in dimension D are listed in Table 6.1, where $z = x - x^*$

Table 6.1 Functions in dimension D for BBOB optimization test bed

$f(z)$	Name	Remarks
$\|\|z\|\|^2 + f_{opt}$	Sphere	Simplest case
$\sum_{i=1}^{D} 10^{6\frac{i-1}{D-1}} z_i^2 + f_{opt}$	Cigar	Ill conditioned
$\sum_{i=1}^{D-1} \left(100(z_i^2 - z_{i+1})^2 + (z_i - 1)^2\right) + f_{opt}$	Rosenbrock	Non-convex
$10\left(D - \sum_{i=1}^{D} cos(2\pi x_i)\right) + \|\|z\|\|^2 + f_{opt}$	Rastrigin	Many local minima
$10^6 z_1^2 + \sum_{i=2}^{D} z_i^2 + f_{opt}$	Discus	Ill conditioned; one critical variable
$z_1^2 + 10^6 \sum_{i=2}^{D} z_i^2 + f_{opt}$	Bent cigar	Ill conditioned; one low importance variable
z_1	Linear slope	One critical variable

for some x^* randomly drawn (often far from the boundaries of the domain), and x^* is the optimum; and f_{opt} randomly drawn as well. Random rotations of these functions are also included.

6.2.2 *Limitations of Test Beds*

Some elements on the limitations of testbeds are given below. We also suggest guidelines for the design of future test beds.

Overfitting. An important issue is overfitting. Overfitting occurs when an OA is itself optimized on a given family of optimization problems. Certainly the algorithm becomes very strong on this family of problems, but it may also happen that the family of test problems is not sufficiently representative of the real optimization problem, for which the optimized algorithm may turn out to be weak. It is therefore important to develop good and large test beds as discussed in Sect. 6.2, including training set, test set, validation set, as is usually done in machine learning. Then, it makes sense to look for the best OA, in average, on this family of functions. BBOB has been designed with a careful look at overfitting issues: Random rotations of these functions are considered as well, in addition to smooth local irregularities (see [1] for details). Noisy versions of these functions are also included, with additive noise or multiplicative noise.

Small dimension. Experimenting in high dimension is time-consuming: Many algorithms become very slow. This is probably the main reason for having only very low dimensionality (for example, all the BBOB benchmarks in 2009 were for dimension less than 20).

Differentiability. All the functions considered in most benchmarks can be differentiated almost everywhere, and everywhere for many of them. With tools like Ampl [9] (automatic differentiation), one can easily interface a function with a nonlinear solver using an automatically computed gradient. Then, optimization in dimension 100,000 is possible, a number which is not possible for many evolutionary algorithms. This means that algorithms are tested:

- On families of fitness functions for which some programs can reach dimension 100,000,
- But the tests are only performed in dimension less than 30 (often less than 20).

Some of the most efficient algorithms on BBOB are unable to run with dimension 100 (and are even almost intractable in dimension 30).

Tweaking effect. Usually, the benchmarks are provided in order to be tested intensively: People using the test bed are allowed to perform hundreds of runs. They know the objective functions in advance, and therefore they can (and must, if they want good results) tweak their program specifically for these fitness functions. What is the robustness of these results? An important point is that the experiments provided in [8] show that they could take an algorithm among the competitors, and tune it so that it outperforms all the algorithms on a function of the CEC 2005 competition (the older version of the BBOB test bed). In machine learning, nearly all the challenges (e.g., Pascal challenges [23]) are protected against the tweaking effect by hiding the real instances; we might guess that this will be also the case in the future of optimization. In the white box case (i.e., fitness function explicitly available for analysis), the ROADEF challenge http://challenge.roadef.org/2010/index.en.htm, for example, is already protected against the tweaking effect, as the real instances are not known in advance (however, usually, related instances are provided for test).

Closed-form fitness function. A more subtle limitation is that only closed-form fitness functions are considered. These fitness functions give a strong advantage to algorithms which cumulate information from one iteration to another because they have nearly the same shape at all scales (e.g., $f(x) = \sum_{n\geq 1} \frac{1}{2^n} sin(4^n x)^2$, see Fig. 6.2). For fitness functions in which things are different at all scales, results might be quite different. We believe that this somehow subtle effect is nonetheless critical and might explain why practitioners sometimes strongly disagree with theoretical analysis on closed-form fitness function.

Bias in the criteria used for ranking algorithms on a given family of problems. The usual criteria used in publications are not always that good for a real-world application. For example, in the continuous domain, authors usually consider ([1] and many others) as a criterion the expected log of the distance to the optimum. This means that it is better (for this criterion) to fail with probability 49/50, and to have a very fast convergence one run out of 50, rather than having a good precision all the time.

We will need some definitions to mathematically rephrase this. Define x_n as the nth search point of your optimization run, and that $x_n \to x^*$, where x^* is the optimum of the fitness function. Assume that

$$\frac{||x_n - x^*||}{||x_{n-1} - x^*||^q} \to \mu \geq 0.$$

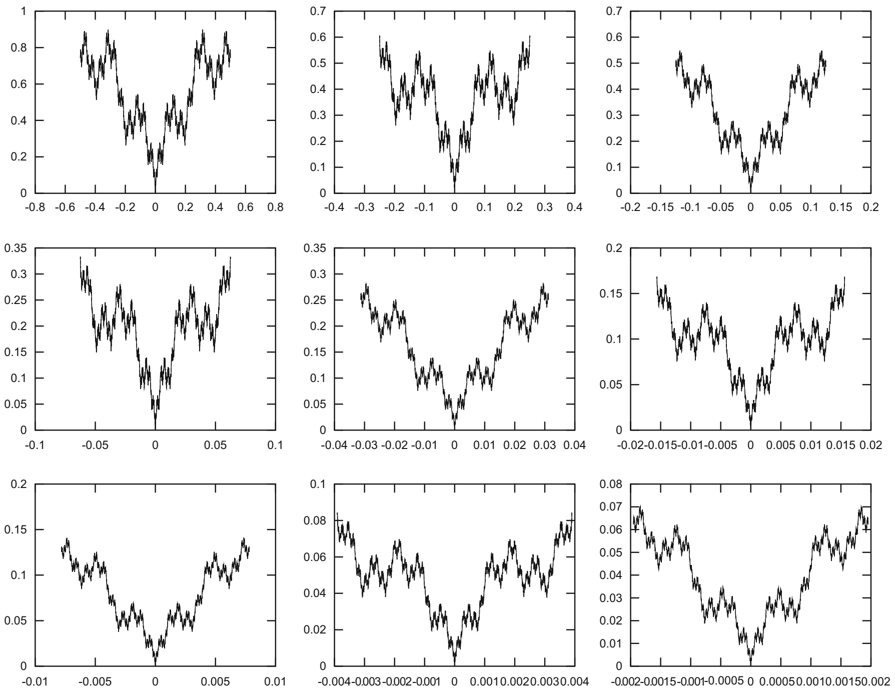


Fig. 6.2 An example of function with nearly the same shape at all scales; $x \mapsto \sum_{n \geq 1} \frac{1}{2^n} \sin(4^n x)$. Each plot is a zoom of the previous plot. A good point in such a fitness function is that the optimization algorithm cannot cheat by knowing in advance the scale at which there are local minima since there are local minima at all scales

then, this convergence is termed:

- A linear convergence if $q = 1, 0 < \mu < 1$;
- A superlinear convergence if $q = 1, \mu = 0$;
- A quadratic convergence if $q = 2, \mu > 0$;
- A convergence of order q, if $\mu > 0$.

More formally, with quadratic convergence once out of 50 runs, and no convergence at all in all other runs, you get a mean log distance to the optimum $\simeq -c_{\text{quadratic}} \cdot 2^N/50$ with N the number of iterations; whereas a linear convergence leads to $\simeq -c_{\text{linear}} \cdot N$. Also, a linear convergence once out of 50 runs (and no convergence at all in other cases) leads to $\simeq -c_{\text{linear}}/50$, while a fixed step-size finding the optimum with precision ϵ in all runs leads to $\simeq \log(\epsilon)$, whereas it's more likely to be satisfactory for the user.

Considering the expected fitness is less usual, because without the log-scale you do not see the nice linear curves predicted by the theoretical analysis, but it might be irrelevant in many cases. This is related to the fact that test beds are usually used with huge numbers of fitness evaluations, which is much better for getting nice curves with linear convergence in log-scale. Is it consistent with real-world criteria?

No real-world fitness function. Working on a real-world fitness function is often quite a time-consuming work, and people will rarely spend time to compare many algorithms on their real-world application. However, the published results around real-world applications can be quite useful. In particular, they show new horizons in terms of experimental setup, for example, optimization runs with far less than the 10^6 or 10^8 fitness evaluations that we often see in publications, with more than dimension 300, with noise, or parallel optimization with many computation units. Also, they show that practitioners sometimes prefer simple understandable algorithms (the $(1+1)$-ES with one-fifth rule [25]), which can be easily plugged into an application (you can implement it in 5 min on any machine with any language) than a complicated algorithm. In some cases, they prefer constant step-size than rules ensuring linear convergence in log-scale.

6.3 Reinforcement Learning Approaches

Optimization consists in choosing, at each iteration, one (resp. several, in the parallel case) new point(s) at which the fitness function will be evaluated. Therefore, optimization can be considered as multistage optimization, i.e., optimization of one decision per time step. More precisely, optimization is usually formalized as follows:

- At each time step t, a point X_t has to be chosen;
- After each such choice, the fitness function y_t at this point is evaluated;
- After all the time steps, a reward (some function of the evaluated points and of the fitness) is evaluated.

If we are a little bit more general, we get the following framework:

- At each time step, a decision has to be chosen;
- After each such choice, we get some information;
- After all the time steps, a reward is given.

This is exactly the framework of reinforcement learning. This section is devoted to this possible solving of optimization problems by reinforcement learning algorithms.

A black-box optimization algorithm is a function which, given past observations (i.e., points with their fitness values), chooses a new point. Therefore, we can consider the optimization algorithm as a (possibly stochastic) function $Optimize$. This functions takes as input the time step of the optimization run and the past chosen points and their fitness values, and outputs a new search point. More precisely, consider the problem of choosing X_1, X_2,...,X_N, the points to be visited, by a function $Optimize$ as follows:

$$\begin{aligned} X_1 &= Optimize(1) \qquad (6.1) \\ y_1 &= f(X_1) \end{aligned}$$

$$\begin{aligned} X_2 &= Optimize(2, X_1, y_1) \\ y_2 &= f(X_2) \\ \ldots &= \ldots \\ X_n &= Optimize(n-1, X_1, y_1, \ldots, X_{n-1}, y_{n-1}) \end{aligned}$$

with the goal of minimizing

$$Loss = \mathbb{E}_f L(f, X_n) \tag{6.2}$$

for some loss function $L(f, X_n)$, which might be, e.g., $L(f, X_n) = f(X_n)$. Equation (6.2) makes sense only if a distribution of probability for f is available; this is a prior for the optimization. Given n and this prior, can we choose $Optimize$ minimizing Eq. (6.2) (which is a quality indicator for the optimization algorithm for the given prior knowledge).

This formulation is quite general, e.g., it covers both discrete and continuous optimization, but could be further extended. The algorithm might be stochastic as well (Eq. (6.2) might then contain an expectation operator w.r.t the source of randomness of the algorithm), or, for including gradient-based algorithms, take into account a gradient, with

$$\begin{aligned} y_i &= f(X_i) \\ g_i &= \nabla f(X_i) \quad \nabla f(X_i) \text{ is the gradient of } f \text{ at } X_i \\ X_{i+1} &= Optimize(i+1, X_1, y_1, g_1, \ldots, X_i, y_i, g_i), \end{aligned}$$

i.e., the newly visited point depends on the iteration index i, the fitness values $(y_j)_{j \le i}$, the gradients $(g_j)_{j \le i}$. In the simplest form above Eqs. (6.1) and (6.2), i.e., only fitness values are used by the optimization algorithm.

For clarity, let us consider an example which is simple, but which shows that noisy optimization (i.e., cases in which the fitness function is noisy) is possible in this framework. This example is as follows. Consider a simple distribution on f, namely, for x^* (optimum of the fitness function) randomly uniformly drawn in the domain $[0, 1]^d$,

$$f(x) = B(\min(1, ||x - x^*||)) \text{ where } x^* = \arg\min f, \tag{6.3}$$

i.e., $f(x)$ is a Bernoulli random variable with parameter $\min(1, ||x - x^*||)$, denoted by $B(\min(1, ||x - x^*|))$. That is, it is 1 with probability $\min(1, ||x - x^*||)$ and 0 otherwise, and the loss at x is the expectation, i.e. $Loss = \mathbb{E}_f \min(1, ||x_n - x^*||)$. Some authors [2, 10, 26] propose to consider and solve the problem defined by Eqs. (6.1) and (6.2) as a Markov decision process (MDP).

Unfortunately, the MDP defined by Eqs. (6.1) and (6.2) has a huge size: The number of time steps is n, and the dimension of the state space is $n \times (d + 1)$. With $n = 50$ and $d = 4$, this implies a dimension 250 for the MDP; this is quite big for usual algorithms. [2, 26] propose the use of the Upper Confidence Tree algorithm

```
*Optimize* function for choosing $X_{k+1}$ given the $X_j$ for $j \leq k$ and their fitness values.
Inputs:

• an integer $n$ (number of iterations of the optimization run); $k < n$.
• a loss function $L$;
• a distribution of fitness functions;
• $s_0 = (x_1, y_1, \ldots, x_k, y_k) = (X_1, y_1, \ldots, X_k, y_k)$ the set of current observations (i.e.
  the $X_i$'s are the previously visited points (i.e. in the domain and the $y_i$'s are their
  fitness values).

Initialize $S$ and $N$ to the zero function; initialize $P(s)$ to the empty set for all $s$.
while Time left $> 0$ do
    $s \leftarrow s_0$.    //beginning of a simulation
    Randomly draw $\hat{f}$ according to the prior distribution on $f$ conditionally to
    $x_1, y_1, \ldots, x_k, y_k$.
    $i \leftarrow k$
    while $i < n$ do
        if $\left(N(s) > \#P(s)^{\frac{1}{2}}\right)$    //progressive widening then
            $P(s) \leftarrow P(s) \cup \{$ one randomly drawn point in the search space$\}$.
        end if
        $x_{i+1} = \arg\min_{x \in P(s)} \hat{L}(x_1, y_1, \ldots, x_i, y_i, x)$, where
            $\hat{L}_s(x) = S(x)/N(x) - \sqrt{\log(N(s))/N(x)}$.
        $y_{i+1} \leftarrow \hat{f}(x_i)$.
        $i \leftarrow i + 1$
    end while
    Let $loss = L(\hat{f}, x_n)$.
    for all $s'$ prefix of $s$    // $s'$ might be of the form $(x_1, y_1, \ldots, x_j, y_j)$ or
    $(x_1, y_1, \ldots, x_j, y_j, x_{j+1})$ do
        $N(s') \leftarrow N(s') + 1$    //number of simulations through $s'$
        $S(s') \leftarrow S(s') + loss$    //sum of simulated losses
    end for
end while
Output: $X_{k+1} = \arg\max_x N(x_1, y_1, \ldots, x_k, y_k, x_k + 1)$, the next visited point.
```

Fig. 6.3 An *Optimize*() function which optimally solves the optimization problem (Eqs. (6.1) and (6.2)) for some probabilistic prior on the fitness function, in the case of a discrete set of y values (i.e., the fitness values might be in $\{0, 1\}$ as in Eq. (6.3), or any other finite set)

(UCT [16]), an algorithm which has been proved very efficient for many difficult problems, and became particularly famous for the application of some variants to the game of Go [4, 6, 18].

The resulting algorithm is summarized in Fig. 6.3 (a complete description is in [26]). This version works only if the fitness values are in a finite set (otherwise the writing is more complicated). Typically, a nice application is the noisy binary case, i.e., the framework given in Eq. (6.3). Algorithm 6.3 is theoretically appealing, as it has a mathematical proof of optimality, within a huge computational power, but unfortunately it is only good with huge computation time; it could only be applied for very limited n and d. Nonetheless, for an optimization process which is repeated often on randomly distributed objective functions, this might be interesting.

In the next section, we'll see the case in which f is drawn according to a gaussian process. We'll see that in that case and under some approximations, it's possible to design fast algorithms (in terms of convergence rate w.r.t the number of iterations) whilst keeping a more reasonable computational cost than with the algorithm above.

6.4 Gaussian Processes for Optimization

The function to be optimized is a real-valued function f defined on some compact search space $\mathbb{X}$. This section deals with OAs that rely on a Gaussian random process to model the objective function. This idea, initiated by J. Mockus under the name of *Bayesian optimization* (see [19, 20, 33, 42] and references therein), has received particular attention during the last decade in the field of *computer experiments*, where functions to be optimized are typically expensive to evaluate (see, e.g., [12, 14, 15, 36]).

6.4.1 From Deterministic to Bayesian Optimization

A deterministic optimization procedure is an application

$$\begin{aligned} \underline{X} : \mathcal{F} := \mathbb{R}^{\mathbb{X}} &\to \mathbb{X}^{\mathbb{N}}, \\ f \quad &\mapsto \underline{X}(f) \ := \ (X_1(f),\, X_2(f),\, \dots)\,, \end{aligned} \tag{6.4}$$

that maps a real-valued function defined on $\mathbb{X}$ to a sequence of points in $\mathbb{X}$, with the property that, for all $n \geq 1$, $X_{n+1}(f)$ depends only on the previous n evaluations $f(X_1(f)),\ \dots,\ f(X_n(f))$. If a probability measure P is chosen on $\mathcal{F}$, then the optimization strategy becomes a *random* sequence $\underline{X}$ in $\mathbb{X}$. Note that randomness only comes from the fact that the objective function f is considered as a random element in $\mathcal{F}$.

The Bayesian approach to optimization is based on two main ideas. The first idea is to capture prior information about the unknown function f by choosing P on $\mathcal{F}$, or equivalently, by choosing the probability distribution of the canonical stochastic process

$$\begin{aligned} \xi : \mathbb{X} \times \mathcal{F} &\to \mathbb{R}, \\ (x, f) &\mapsto \xi(x, f) := f(x). \end{aligned} \tag{6.5}$$

For practical reasons, only Gaussian process priors (which are, roughly speaking, the extension of multivariate Gaussian random variables to a continuous index for dimensions) have been considered in the literature. In this case, the prior is completely characterized by the first- and second-order moments of ξ, that is, the mean function $m(x) := \mathsf{E}[\xi(x)]$ and the covariance function $k(x, y) := \mathsf{E}[(\xi(x) - m(x))(\xi(y) - m(y))]$. The mean and covariance functions are chosen by the user to correspond to his/her prior knowledge on f. The covariance function can be chosen to match a regularity assumption [31, 35]. If no prior knowledge is available in advance, these functions are generally estimated from the data using, for instance, maximum likelihood estimation (see the discussion in Sect. 6.4.4.1).

The second idea of Bayesian optimization is to choose the evaluation points X_n according to a sampling criterion $\rho_n(x)$ that measures the interest (for the optimization problem) of an additional evaluation at $x \in \mathbb{X}$, given the previous evaluations $f(X_1(f)), \ldots, f(X_n(f))$. Thus, to choose the $(n+1)$th evaluation point, one has to solve a second optimization problem

$$X_{n+1} = \arg\max_{x \in \mathbb{X}} \rho_n(x), \tag{6.6}$$

which, unfortunately, is usually not analytically solvable.

In other words, Bayesian optimization replaces the optimization of the objective function by a series of optimizations of a sampling criterion. This is only interesting if the original optimization problem is so expensive that solving many problems of the form in Eq. (6.6) is a better idea; this is typically the case if the original problem involves a very expensive fitness function.

Of course, each optimization program defined by Eq. (6.6) must nonetheless be preferably cheap, which entails that the evaluation of ρ_n at a given point of $\mathbb{X}$ must also be cheap. To summarize, ρ_n should be a cheap-to-evaluate criterion that quantifies the interest of an additional evaluation—using what has been assumed on f through P, and learned from previous evaluations—and reflects our concern that evaluations of f are expensive.

In this setting, a key issue is how to update the prior P—or equivalently, the distribution of ξ—when new evaluation results are obtained; or in other words, how to take into account previous evaluations for the choice of a new one. Here, the *kriging predictor* plays a central role because it is a method that makes it possible to update the prior in an easy way. Section 6.4.3 presents three algorithms based on three different sampling criteria. The three algorithms make a direct use of the kriging predictor. For the sake of consistency, we shall discuss very briefly how to build the kriging predictor in the next section.

6.4.2 An Introduction to Kriging

Kriging [5, 37] is widely used in the domain of computer experiments since the seminal paper of [27] as a method to construct an approximation of an unknown function from scattered data. It can also be understood as a kernel regression method, such as splines [38], Radial Basis Functions [40], or Support Vector Regression [32]. Kriging is also known as the Best Linear Unbiased Predictor (BLUP) in statistics [31], and has been reinvented in the machine-learning community under the name of Gaussian Processes [24].

Let ξ be a Gaussian process with mean $m(x)$, $x \in \mathbb{X}$, and covariance function $k(x,y)$, $(x,y) \in \mathbb{X}^2$. Assume that $m(x)$ can be written as a linear parametric function $m(x) = b^\mathsf{T} p(x)$, where $p(x)$ is the q-dimensional vector of all monomials of degree less than or equal to $l \in \mathbb{N}$, and $b \in \mathbb{R}^q$ is a vector of fixed but *unknown* coefficients. The theory of kriging is concerned with the construction of the *best linear unbiased predictor* (BLUP) of ξ based on a finite set of pointwise observations of the process. For $x \in \mathbb{X}$ and $\underline{x}_n := (x_1, \dots, x_n) \in \mathbb{X}^n$, $n \geq 1$, denote by $\hat{\xi}(x; \underline{x}_n)$ a linear predictor of $\xi(x)$ based on $\xi(x_1), \dots, \xi(x_n)$. Such a linear predictor can be written as

$$\hat{\xi}(x; \underline{x}_n) = \lambda(x; \underline{x}_n)^\mathsf{T} \underline{\xi}_n \,, \tag{6.7}$$

with $\underline{\xi}_n = (\xi(x_1), \dots, \xi(x_n))^\mathsf{T}$ and $\lambda(x; \underline{x}_n)$ a vector of weights $\lambda_i(x; \underline{x}_n)$, $i = 1, \dots, n$. The BLUP is the linear projection of $\xi(x)$ onto $\mathrm{span}\{\xi(x_i), i \leq n\}$ orthogonally to the space of functions $\mathcal{P} := \{b^\mathsf{T} p(x); b \in \mathbb{R}^q\}$, and in such way that the norm of the prediction error is minimum. Then, the vector of kriging coefficients $\lambda(x; \underline{x}_n)$ is obtained as the solution of the linear system of equations [5]

$$\begin{pmatrix} k(\underline{x}_n, \underline{x}_n) & P^\mathsf{T} \\ P & 0 \end{pmatrix} \begin{pmatrix} \lambda(x; \underline{x}_n) \\ \alpha(x; \underline{x}_n) \end{pmatrix} = \begin{pmatrix} k(x, \underline{x}_n) \\ p(x) \end{pmatrix} , \tag{6.8}$$

where $k(\underline{x}_n, \underline{x}_n)$ is the $n \times n$ matrix of covariances $k(x_i, x_j)$, P is a $q \times n$ matrix with entries $x_j{}^i$ for $j = 1, \dots, n$ and multi-indexes $i = (i_1, \dots, i_d)$ such that $|i| := i_1 + \cdots + i_d \leq l$, $\alpha(x; \underline{x}_n)$ is a vector of Lagrange coefficients, $k(x, \underline{x}_n)$ is a vector of size n with entries $k(x, x_i)$ and $p(x)$ is a vector of size q with entries x^i, i such that $|i| \leq l$. It is one of the main advantages of kriging that its implementation only necessitates very basic numerical operations.

The variance of the kriging error, also called *kriging variance*, is given by

$$\sigma^2(x; \underline{x}_n) := var[\xi(x) - \hat{\xi}(x; \underline{x}_n)] = k(x,x) - \lambda(x; \underline{x}_n)^\mathsf{T} k(x, \underline{x}_n) - \alpha(x; \underline{x}_n)^\mathsf{T} p(x) \,. \tag{6.9}$$

The knowledge of this variance makes it possible to derive confidence intervals for the predictions. If a given region of $\mathbb{X}$ remains unexplored, the variance of the prediction error is generally high. From the viewpoint of optimization, it means that a global optimizer may be contained in that region. Note that the kriging predictor is an interpolator (the kriging variance is equal to zero at evaluation points).

6.4.3 Bayesian Sampling Criteria

In the following sections, three Bayesian sampling criteria to find the *global maxima* of f are presented. To simplify the presentation, we shall assume that f is modeled by a *zero-mean* Gaussian random process ξ with *known covariance function* $k(x, y)$. We shall assume moreover that the global optimum, denoted by $M = \sup_{x \in \mathbb{X}} f(x)$, is unique. The corresponding global optimum will be denoted by $X^\star$. We shall also use the notations $M_n := \xi(X_1) \vee \cdots \vee \xi(X_n)$ (the maximum of the evaluation results at step n) and $m_n := \xi(X_1) \wedge \cdots \wedge \xi(X_n)$ (the minimum of the evaluations).

6.4.3.1 P-Algorithm

One of the first sampling criteria proposed in the literature is based on the idea of iteratively maximizing the probability of excursion of ξ over a sequence of thresholds u_n, $n \geq 1$ [17]. At each step n, if $u_n = M_n$ for instance, then the idea is to select the next evaluation point that will have the highest probability to exceed the current maximum. The algorithm derived from this criterion is called the P-algorithm [14, 42]. Using our previous assumptions on ξ and the kriging predictor, the strategy can be written formally as

$$X_{n+1} = \arg\max_{x \in \mathbb{X}} \rho_n(x) := \mathsf{P}\{\xi(x) \geq u_n \mid \xi(X_1), \ \ldots, \ \xi(X_n)\}$$
$$= \begin{cases} 1 - \Phi\left(\frac{\hat{\xi}(x; \underline{X}_n) - u_n}{\sigma(x; \underline{X}_n)}\right) & \text{if } \sigma(x; \underline{X}_n) > 0, \\ 0 & \text{if } \sigma(x; \underline{X}_n) = 0, \end{cases} \tag{6.10}$$

where Φ is the normal cumulative distribution function. As mentioned in Sect. 6.4.1, the optimization problem (6.10) is again a global optimization problem, but the criterion ρ_n can be optimized with limited computational resources. The choice of u_n is a free parameter of the algorithm. Jones [14] suggests to use the empirical rule

$$u_n = \max_{x \in \mathbb{X}} \hat{\xi}(x; \underline{X}_n) + \alpha_n (M_n - m_n),$$

with α_n a positive number. Small values of α_n entail that the algorithm favors exploitation over exploration, which means that the algorithm favors local optimization over global optimization. The converse holds true for large values of α_n; that is, the algorithm tries to sample unexplored regions rather than promising regions. The analysis of the convergence of the algorithm has been proposed under restrictive hypotheses in [3].

6.4.3.2 Expected Improvement

The *expected improvement* (EI) algorithm (also called Efficient Global Optimization) is a popular OA proposed by J. Mockus in the 1970s and brought to the field of computer experiments by M. Schonlau [15, 28–30]. The EI algorithm chooses a new evaluation point X_{n+1} as the maximizer over $\mathbb{X}$ of the quantity

$$\rho_n(x) := \mathsf{E}\big[(\xi(x) - M_n) \vee 0 \mid \xi(X_1),\ \ldots,\ \xi(X_n)\big],$$
$$:= \begin{cases} s\,\Phi'\left(\frac{z}{s}\right) + z\,\Phi\left(\frac{z}{s}\right) & \text{if } s > 0, \\ \max(z, 0) & \text{if } s = 0. \end{cases} \tag{6.11}$$

with $z = \hat{\xi}(x; \underline{X}_n) - M_n$ and $s = \sigma(x; \underline{X}_n)$. The function $\rho_n(x)$, which is called the expected improvement at x, can be interpreted as the conditional mean excess of $\xi(x)$ above the current maximum M_n. There is no parameter to tune here, which is certainly an advantage with respect to the P-algorithm. Many applications of the EI algorithm are available in the engineering literature. The convergence of the EI algorithm has been analyzed in [34].

6.4.3.3 Informational Approach to Global Optimization

The *Informational Approach to Global Optimization* (IAGO) proposed recently in [36] provides a choice of an evaluation point that is one-step optimal in terms of reduction of the uncertainty on the maximizer location. It is based on two main ideas. The first idea is to estimate the probability distribution $\mathsf{P}_{X^\star|\underline{\xi}_n}$ of $X^\star$ conditioned on the observation vector $\underline{\xi}_n = (\xi(X_1)\ldots\xi(X_n))$. This probability distribution represents what has been learned (through evaluations) and assumed (through the Gaussian model) about $X^\star$. The progress made in finding a solution to the global optimization problem can be assessed by looking at how the probability distribution of $X^\star$ is spread over $\mathbb{X}$—in practice, only a *finite* subset $\mathbb{X}_d \subset \mathbb{X}$ is considered. In particular, the support of the distribution narrows as progress is being made toward reducing the uncertainty left on $X^\star$. This estimation can be carried out using the kriging predictor and *conditional simulations* of ξ over a finite grid (see [5, 36] for an insight into how conditional simulations are generated).

The second idea is to adopt an information-based search strategy, by sampling f where the largest uncertainty reduction on $X^\star$ is expected. To quantify uncertainty, the conditional Shannon entropy of the global optimum has been suggested. This results in a sampling strategy defined as

$$X_{n+1} = \arg\min_{x\in\mathbb{X}} \mathsf{E}\left[H(X^\star; \underline{\xi}_n, \xi(x)) \mid \underline{\xi}_n\right], \tag{6.12}$$

$$= \arg\min_{x\in\mathbb{X}} \int_{z\in\mathbb{R}} H(X^\star; \underline{\xi}_n, \xi(x) = z)\, p_{\xi(x)|\underline{\xi}_n}(z)dz, \tag{6.13}$$

where $p_{\xi(x)|\underline{\xi}_n}$ denotes the density of the candidate observation conditioned on $\underline{\xi}_n$ and where $H(X^\star; \underline{\xi}_n, \xi(x))$ stands for the entropy of $X^\star$ conditioned on $\underline{\xi}_n$

and the candidate observation $\xi(x)$, which can be written—using straightforward notations—as

$$H(X^\star; \underline{\xi}_n, \xi(x)) = -\sum_{y \in \mathbb{X}_d} P_{X^\star|\underline{\xi}_n,\xi(x)}(y) \log P_{X^\star|\underline{\xi}_n,\xi(x)}(y)\,.$$

The advantages of IAGO over EI have been discussed in [35]. However, it must be stated that the computational complexity of IAGO is significantly higher than that of EI (cf. [35]).

6.4.4 From the Sampling Criterion to the OA

In this section, we shall discuss very briefly how to insert these criteria into OA, and give some brief recommendations regarding their practical use.

6.4.4.1 Choosing a Model

The choice of P (or, equivalently, that of ξ) is a central and largely open question in the domain of Bayesian optimization. In the framework of kriging, the covariance function of ξ is generally chosen in a parametrized class, and its parameters are estimated from available data. The method used for this estimation (variogram fitting in geostatistics, cross-validation in machine learning or maximum likelihood in statistics) may not be adapted in a context of a small number of evaluations. Now, Bayesian optimization may perform very poorly if the covariance is inadequate (see, e.g., [14] for examples). Empirical results indicate, however, that it is possible to choose a fixed covariance that ensures a satisfactory behavior over a large class of functions [34].

6.4.4.2 Optimization of the Sampling Criterion

Optimizing a sampling criterion ρ_n to select a new evaluation point is an important practical difficulty of Bayesian optimization, as ρ_n may have many local optima. However, one has to keep in mind that the evaluation of the sampling criterion does not require any evaluation of the objective function f, and that there is no need for exact optimization since the sampling criterion is only used to determine the next evaluation point. In practice, working with approximate solutions for the optimizers of the sampling criterion has little influence on the final estimation of the optimum of f.

At the beginning of the optimization procedure, very little is known on the potential location of the optimizers. However, as the number of evaluations increases, it becomes obvious that certain areas of search space do not have any interest for an additional evaluation as they will probably contain no global optimizer. A possible strategy is then to perform the optimization of the sampling criterion over a *finite set of candidate points* and resample this set at each iteration to maintain a good representation of the support of the probability density of the optimizers of f over $\mathbb{X}$.

6.5 Conclusion

We have discussed the design of OA specialized on some prior. We have seen that this prior might be a set of (possibly randomized) fitness functions; then, an OA based on this prior might be:

- An OA with mathematically derived parameters; however, this is often quite difficult in real life, and many practitioners use parametrizations very far from the choice of theoreticians.
- An experimentally tuned optimizer, based on a "meta"-level; i.e., another optimizer is in charge of optimizing the optimizer, as in Sect. 6.2.
- A provably optimal optimizer, built on the distribution of fitness functions (or possible for a worst case on a distribution of fitness functions); this involves complicated Markov Decision Process solvers as in Sect. 6.3. Due to huge internal computational cost of such optimizers, they are for the moment essentially theoretical, in spite of some recent positive results.
- An optimization algorithm somehow similar to the above provably optimal algorithm, but for which the process is simplified by a sufficiently "nice" prior as Gaussian processes in Sect. 6.4, and an approximate criterion ρ_n. This approach (e.g., Efficient Global Optimization (EGO), Informational Approach to Global Optimization (IAGO)) is quite expensive, but nonetheless provides operational OA for some expensive optimization problems.

Further work remains to (i) promote better test beds (in particular emphasizing real-world problems and frameworks more related to real-world problems), and (ii) increase the speed of EGO, IAGO, and more theoretical algorithms discussed in this chapter, so that they can be tested and used more conveniently.

Acknowledgements O. Teytaud is grateful to NSC for funding NSC100-2811-E-024-001, to ANR for funding COSINUS program (project EXPLO-RA ANR-08-COSI-004), and to the European FP7 program (European Project Nr. FP7-ICT-247022).

References

1. A. Auger, H.G. Beyer, N. Hansen, S. Finck, R. Ros, M. Schoenauer, D. Whitley, Black-box optimization benchmarking, in *GECCO'09: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, Montreal, 2009

2. A. Auger, O. Teytaud, Continuous lunches are free plus the design of optimal optimization algorithms. Algorithmica **57**(1), 121–146 (2010)
3. J.M. Calvin, A one-dimensional optimization algorithm and its convergence rate under Wiener measure. J. Complex. **17**, 306–344 (2001)
4. G. Chaslot, M.H.M. Winands, J.W.H.M. Uiterwijk, H. van den Herik, B. Bouzy, Progressive strategies for Monte Carlo tree search, in *Proceedings of the 10th Joint Conference on Information Sciences (JCIS 2007)*, Salt Lake City, ed. by P. Wang et al. (World Scientific Publishing Co. Pvt. Ltd., 2007), pp. 655–661
5. J.P. Chilès, P. Delfiner, *Geostatistics: Modeling Spatial Uncertainty* (Wiley, New York, 1999)
6. R. Coulom, Efficient selectivity and backup operators in Monte Carlo tree search, in *Proceedings of the 5th International Conference on Computers and Games*, Turin, 2006, ed. by P. Ciancarini, H.J. van den Herik
7. S. Droste, T. Jansen, I. Wegener, Perhaps not a free lunch but at least a free appetizer, in *Proceedings of the First Genetic and Evolutionary Computation Conference (GECCO'99)*, San Francisco, 13–17, ed. by W. Banzhaf, J. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M. Jakiela, R.E. Smith (Morgan Kaufmann, 1999), pp. 833–839
8. A.E. Eiben, Principled approaches to tuning EA parameters, in *Proceedings of CEC (tutorial)*, Trondheim, 2009
9. R. Fourer, D.M. Gay, B.W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming* (Duxbury Press, Belmont, 2002)
10. S. Gelly, S. Ruette, O. Teytaud, Comparison-based algorithms are robust and randomized algorithms are anytime. Evol. Comput. J. (Special Issue on Bridging Theory and Practice) **15**(4), 26 (2007)
11. N.I.M. Gould, D. Orban, P.L. Toint, CUTEr and SifDec: a constrained and unconstrained testing environment, revisited. ACM Trans. Math. Softw. **29**(4), 373–394 (2003)
12. D. Huang, T. Allen, W. Notz, N. Zeng, Global optimization of stochastic black-box systems via sequential kriging meta-models. J. Glob. Optim. **34**, 441–466 (2006)
13. C. Igel, M. Toussaint, On classes of functions for which no free lunch results hold. Inf. Process. Lett. **86**, 317–321 (2003). See also Los Alamos Preprint cs.NE/0108011
14. D.R. Jones, A taxonomy of global optimization methods based on response surfaces. J. Glob. Optim. **21**, 345–383 (2001)
15. D.R. Jones, M. Schonlau, W.J. Welch, Efficient global optimization of expensive black-box functions. J. Glob. Optim. **13**, 455–492 (1998)
16. L. Kocsis, C. Szepesvari, Bandit-based Monte Carlo planning, in *ECML'06*, Berlin, 2006, pp. 282–293
17. H.J. Kushner, A new method of locating the maximum point of an arbitrary multipeak curve in the presence of noise. J. Basic Eng. **86**, 97–106 (1964)
18. C-S. Lee, M-H. Wang, G. Chaslot, J-B. Hoock, A. Rimmel, O. Teytaud, S-R. Tsai, S-C. Hsu, T-P. Hong, The computational intelligence of MoGo revealed in Taiwan's Computer Go tournaments, *IEEE Trans. Comput. Intell. AI Games* **1**(1), 73–89 (2009)
19. J. Mockus, *Bayesian Approach to Global Optimization: Theory and Applications* (Kluwer, Dordrecht/Boston/London, 1989)
20. J. Mockus, V. Tiesis, A. Zilinskas, The application of Bayesian methods for seeking the extremum, in *Towards Global Optimization*, vol. 2, ed. by L.C.W. Dixon, G.P. Szego (North-Holland, New York, 1978) pp. 117–129
21. V. Nannen, A.E. Eiben, Relevance estimation and value calibration of evolutionary algorithm parameters, in *International Joint Conference on Artificial Intelligence (IJCAI'07)*, Hyderabad, 2007, pp. 975–980
22. V. Nannen, A.E. Eiben, Variance reduction in meta-EDA, in *GECCO'07: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, London (UK) (ACM, New York, 2007) pp. 627–627
23. Pascal Challenges, http://pascallin2.ecs.soton.ac.uk/Challenges/, 2011
24. C.E. Rasmussen, C.K.I. Williams, *Gaussian Processes for Machine Learning* (MIT, Cambridge, 2006)

25. I. Rechenberg, *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien des biologischen Evolution* (Frommann-Holzboog Verlag, Stuttgart, 1973)
26. P. Rolet, M. Sebag, O. Teytaud, Optimal robust expensive optimization is tractable, in *GECCO'09: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, Montréal (ACM, 2009)
27. J. Sacks, W.J. Welch, T.J. Mitchell, H.P. Wynn, Design and analysis of computer experiments. Stat. Sci. **4**(4), 409–435 (1989)
28. M. Schonlau, Computer experiments and global optimization, Ph.D. thesis, University of Waterloo, Waterloo, 1997
29. M. Schonlau, W.J. Welch, Global optimization with nonparametric function fitting, in *Proceedings of the ASA, Section on Physical and Engineering Sciences*, Alexandria (American Statistical Association, 1996) pp. 183–186
30. M. Schonlau, W.J. Welch, D.R. Jones, A data analytic approach to Bayesian global optimization, in *Proceedings of the ASA, Section on Physical and Engineering Sciences*, Anaheim (American Statistical Association, 1997) pp. 186–191
31. M.L. Stein, *Interpolation of Spatial Data: Some Theory for Kriging* (Springer, New York, 1999)
32. I. Steinwart, A. Christmann, *Support Vector Machines* (Springer, New York, 2008)
33. A. Törn, A. Zilinskas, *Global Optimization* (Springer, Berlin, 1989)
34. J. Villemonteix, *Optimisation de fonctions coûteuses*, PhD thesis, Université Paris-Sud XI, Faculté des Sciences d'Orsay, 2008
35. J. Villemonteix, E. Vazquez, M. Sidorkiewicz, E. Walter, Global optimization of expensive-to-evaluate functions: an empirical comparison of two sampling criteria. J. Glob. Optim. **43**(2–3), 373–389 (2009)
36. J. Villemonteix, E. Vazquez, E. Walter, An informational approach to the global optimization of expensive-to-evaluate functions. J. Glob. Optim. **44**(4), 509–534 (2009). doi:10.1007/s10898-008-9354-2
37. H. Wackernagel, *Multivariate Geostatistics* (Springer, Berlin, 1995)
38. G. Wahba, in *Spline Models for Observational Data*. Volume 59 of CBMS-NSF Regional Conference Series in Applied Mathematics (SIAM, Philadelphia, 1990)
39. B. Weinberg, E.G. Talbi, NFL theorem is unusable on structured classes of problems, in *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, Portland (IEEE, 2004), pp. 220–226
40. H. Wendland, *Scattered Data Approximation*. Monographs on Applied and Computational Mathematics (Cambridge University Press, Cambridge, 2005)
41. D.H. Wolpert, W.G. Macready, No free lunch theorems for search, Technical Report, Santa Fe Institute, 1995
42. A. Zilinskas, A review of statistical models for global optimization. J. Glob. Optim. **2**, 145–153 (1992)

Chapter 7
The Bayesian Search Game

Marc Toussaint

Abstract The aim of this chapter is to draw links between (1) No Free Lunch (NFL) theorems which, interpreted inversely, lay the foundation of how to design search heuristics that exploit prior knowledge about the function, (2) partially observable Markov decision processes (POMDP) and their approach to the problem of sequentially and optimally choosing search points, and (3) the use of Gaussian processes as a representation of belief, i.e., knowledge about the problem. On the one hand, this joint discussion of NFL, POMDPs and Gaussian processes will give a broader view on the problem of search heuristics. On the other hand this will naturally introduce us to efficient global optimization algorithms that are well known in operations research and geology (Gutmann, J Glob Optim 19:201–227, 2001; Jones et al., J Glob Optim 13:455–492, 1998; Jones, J Glob Optim 21:345–383, 2001) and which, in our view, naturally arise from a discussion of NFL and POMDPs.

7.1 Introduction

We consider the problem of optimization, where an objective function $f : X \to \mathbb{R}$ is fixed but unknown and an algorithm has to find points in the domain X which maximize $f(x)$. In this paper we take the view that search is a problem of navigating through belief space. With "belief" we denote our current knowledge about the objective function represented in terms of a probability distribution over problems (functions). In principle, Bayes' rule tells us how to update this belief state when we explore a new search point and thereby gain new knowledge about the objective function. In that sense, repeatedly querying search points generates a trajectory

M. Toussaint (✉)
Machine Learning & Robotics Lab, Free University of Berlin, Arnimallee 7, 14195 Berlin, Germany
e-mail: marc.toussaint@fu-berlin.de

through belief space—and efficient search means to "navigate" through belief space in a goal-directed manner. For instance, navigating with the goal to gain information about the objective function, or with the goal to reach a belief that implies certainty about the location of the optimum of the objective function.

In this view, the problem of optimization can be framed as a partially observable Markov decision process problem just in the same way as standard navigation problems. The POMDP formulation naturally ties in with an alternative formulation of the No Free Lunch (NFL) theorem: In the next section we present such a formulation which is equivalent to the one in [5] but, instead of relying on classical notions like "closed under permutation", is formulated in terms of the structure of the function prior $P(f)$. In a sequential search process, this prior is updated to become a new posterior $P(f \mid \text{observations})$ after each exploration of a search point and observation of the corresponding function value. This posterior is the belief state in the corresponding POMDPs. Therefore, the POMDP framework directly implies the optimal policy in the case that NFL conditions do not hold.

Clearly, for most relevant cases the optimal search policy is infeasible to compute. However, when making strong assumptions about the prior belief—that is, our initial knowledge about the objective function—and approximating optimal planning with optimal 1- or 2-step look-ahead planning, then such algorithms become tractable. An example is search in continuous spaces when the prior belief is a Gaussian process. The resulting approximate optimal search algorithms are of high practical relevance and have a long history in operations research and geology (e.g., under the name of *kriging*) [2, 6, 7].

The material covered in this chapter is complementary to the survey on (approximately) optimal search algorithms in Chap. 6. In particular, Chap. 6 gives an explicit introduction to kriging, while the focus of this chapter is on the alternative formulation of NFL, how this ties in with the POMDP approach to optimal search policies, and a basic demonstration of a truly planning (2-step look-ahead) search policy in the case of Gaussian processes.

In the following section we present an alternative formulation of a general NFL result that is equivalent to the one presented in [5]. Section 7.3 briefly introduces the most relevant notions of POMDPs. Section 7.4 then draws the relation between POMDPs and optimization. We interpret optimization as a "Bayesian search game" and discuss the belief update when we acquire new observations during search. In Sect. 7.5 we define some simple heuristic policies to choose new search points based on the current belief, including one that would be optimal for a two-step horizon problem. Finally, in Sect. 7.6 we discuss the use of Gaussian processes as belief representation, as was done before in the context of kriging [2, 6, 7], and illustrate the resulting optimization algorithms on some examples. The reader may experience the idea of search using Gaussian processes by literally playing the Bayesian search game (competing with a Gaussian processes-based search policy), using the implementation given at the author's webpage.[1]

[1] http://userpage.fu-berlin.de/mtoussai/07-bsg/

7.2 Yet Another Formulation of NFL

Let us begin with a simple formulation of No Free Lunch (NFL) [16]. Roughly speaking, NFL theorems specify conditions under which "informed search"—that is, picking search points better than random—is impossible. These issues are intimately linked to the conditions under which generalization in learning theory is possible—which we discuss briefly below. The point in specifying such conditions is that (1) one should never try to write an efficient search algorithm when NFL conditions hold, and (2) the NFL theorems should give a hint on how to design a search algorithm when these conditions do not hold.

There are many alternative formulations of NFL theorems; a standard one for optimization is [16]. In [5] we presented a general formulation which specifies conditions on the probability distribution over the objective function. The formulation we present here is equivalent to the one in [5] but more naturally leads to the notion of beliefs, POMDPs and Bayesian search. The specific formulation and proof we give here are, to our knowledge, novel—but only a minor variant of the existing formulations; see [5] for a more extensive discussion of existing NFL formulations.

Let X be a finite or continuous search space. We call elements in X sites. Assume a search algorithm is applied on a function $f : X \to Y$ sampled from $P(f)$. We write f_x for the function value of f at site x. A non-revisiting algorithm iteratively samples a new site x_t and gets in return an observation $y_t = f_{x_t}$. We formalize an algorithm $\mathcal{A}$ as a search distribution $P(x_t \mid x_{0:t-1}, y_{0:t-1}; \mathcal{A})$ conditioned on previous samples and their observed values, and the initial search distribution $P(x_0; \mathcal{A})$, with zero probability of revisitation, $x_t \in x_{0:t-1} \Rightarrow P(x_t \mid x_{0:t-1}, y_{0:t-1}; \mathcal{A}) = 0$. All this defines a stochastic process of the search algorithm interacting with the objective function, as summarized by the joint distribution

$$\begin{aligned} &P(f, x_{0:T}, y_{0:T}; \mathcal{A}) \\ &\quad = P(f)\; P(y_0 \mid x_0, f)\; P(x_0; \mathcal{A}) \prod_{t=1}^{T} P(y_t \mid x_t, f)\; P(x_t \mid x_{0:t-1}, y_{0:t-1}; \mathcal{A})\,. \end{aligned} \tag{7.1}$$

Theorem 7.1. *In this setting, a basic NFL theorem reads*

$$\exists h : Y \to \mathbb{R} \;\; s.t.$$

$$\forall \textit{ finite subsets } \{x_1, .., x_K\} \subset X : \;\; P(f_{x_1}, .., f_{x_K}) = \prod_{k=1}^{K} h(f_{x_k}) \tag{7.2}$$

$$\Longleftrightarrow \qquad \forall_{\mathcal{A}}, \forall_T : \;\; P(y_{0:T}; \mathcal{A}) = \prod_{i=0}^{T} h(y_i) \quad \textit{(independent of } \mathcal{A}) \tag{7.3}$$

The condition (7.2) on the left simply says that $P(f)$ factorizes identically,[2] which means that every f_x is mutually independent from every other $f_{x'}$—nothing can be learnt about $f_{x'}$ from f_x. The "for-all-finite-subsets" formulation we used is typical for continuous X and in analogy to the definition of Gaussian processes (see below). Additionally, the condition (7.2) says that every marginal distribution $h(f_{x_k})$ (we called $h : Y \to \mathbb{R}$ *histogram* of function values in [5]) is identical, independent of the site x_k. In other terms, the function values are identically independently distributed (independently refers to different sites x). Hence, no algorithm can predict the observation at a new site based on previous samples better than with a constant marginal that ignores previous samples and the location of the site. The inevitable result is that the function values an algorithm observes are a random sequence independent of the algorithm itself.

Proof. We first show Eq. (7.2) $\Rightarrow$ Eq. (7.3): Up to some total time t we have the random variables $f, x_{0:t}, y_{0:t}$. Their joint is given by Eq. (7.1). Here, $P(y_t \mid x_t, f)$ is the probability of observing a value y_t when sampling at point x_t, given the function is f. This could account for noisy function evaluations, but for simplicity here we simply assume $P(y_t \mid x_t, f) = \delta_{y_t, f_{x_t}}$. Given this joint, we find

$$\begin{aligned}
&P(y_t \mid x_{0:t-1}, y_{0:t-1}; \mathcal{A}) \\
&= \sum_{x_t \in X} \Big[\sum_f P(y_t \mid x_t, f)\; P(f \mid x_{0:t-1}, y_{0:t-1}) \Big]\; P(x_t \mid x_{0:t-1}, y_{0:t-1}; \mathcal{A}) \\
&= \sum_{x_t \in X} P(f_{x_t} = y_t \mid x_{0:t-1}, y_{0:t-1})\; P(x_t \mid x_{0:t-1}, y_{0:t-1}; \mathcal{A}) \\
&= \sum_{x_t \in X} h(y_t)\; P(x_t \mid x_{0:t-1}, y_{0:t-1}; \mathcal{A}) = h(y_t)\;.
\end{aligned} \tag{7.4}$$

The last line used the fact that the algorithm is non-revisiting and that $P(f)$ factorized, such that $P(f_{x_t} = y_t \mid x_{0:t-1}, y_{0:t-1}) = P(f_{x_t} = y_t) = h(y_t)$. (For X continuous we need to replace summations by integrals.) This means that a newly sampled function value y_t is independent of the algorithm $\mathcal{A}$ and of the history $x_{0:t-1}, y_{0:t-1}$. By induction over $t = 0, 1, \ldots$ we get the right-hand side (RHS), Eq. (7.3).

We now show the inverse Eq. (7.2) $\Leftarrow$ Eq. (7.3): To show $\neg$(7.2)$\Rightarrow$ $\neg$(7.3) let $\{x_1, .., x_K\}$ for which $P(f_{x_1}, .., f_{x_K})$ does not identically factorize. We distinguish two cases: (i) In the case that the marginals are not identical (h depends on the site) it is clear that two algorithms that pick two different sites (with different marginals h) as the first search point will have a different $P(y_0)$—and the RHS (7.3) is violated. (ii) If all marginals $P(f_{x_1}) = h(f_{x_1})$ are the same but $P(f_{x_1}, .., f_{x_K})$ does not

[2]On true *subsets* $\subset X$, but not all subsets $\subseteq X$. This weaker condition ensures that also the $\Leftarrow$ holds; see proof for details.

factorize, then at least one conditional $P(f_{x_1} \mid f_{x_2}, .., f_{x_K}) \neq h(f_{x_1})$ is different from the marginal. Two algorithms that deterministically first pick $x_2, .., x_K$ and then, *depending* on the conditional $P(f_{x_1} \mid f_{x_2}, .., f_{x_K})$ will pick either x_1 or an outside point in $X \setminus \{x_1, .., x_K\}$ (here we need the real subset $\{x_1, .., x_K\} \subset X$ rather than the $\subseteq X$) will have a different $P(y_K)$ than random search—and the RHS (7.3) is violated.

To link to more traditional presentations: The left-hand side, LHS (7.2), is related to sets of functions which are closed under permutation. In particular, associating equal probability to functions in a set closed under permutation leads to independent and identically distributed function values at different points. The LHS (7.2) is equivalent to the so-called strong NFL conditions in [5]. Further, one usually assumes some criterion $\mathcal{C}$ that evaluates the quality of an algorithm $\mathcal{A}$ by mapping the sequence $y_{0:t}$ of observed values to a real number. Obviously, if $P(y_{0:t})$ is independent of the algorithm, then so is $\sum_{y_{0:t}} P(y_{0:t})\ C(y_{0:t})$. In traditional terms this means, averaged over all functions (in terms of $P(f)$), the quality of an algorithm is independent of the algorithm. For instance, every algorithm is as good as random search.

A note on continuous spaces: The LHS condition (7.2) is interesting in the case of continuous search spaces, which touches deeply into the notion of well-defined measures over functions in continuous space. Naively, the LHS condition (7.2) describes something like a Gaussian process with a zero covariance function, $C(x, x') = 0$ for any $x \neq x'$. At first sight there seems to be no problem in defining such a distribution over functions also in continuous space, in particular because the definition of a Gaussian process only makes reference to function value distributions over *finite* subsets of the domain. However, [1] make the point that this "zero-covariance" Gaussian process is actually not a proper Lebesgue measure over the space of function. This means any $P(f)$ which fulfils the LHS (7.2) is not a Lebesgue measure. Inversely, if we assume that $P(f)$ is a Lebesgue measure—and [1] imply that this is the only sensible definition of measure over functions in continuous space—then it follows that NFL does not hold in continuous domains.

A note on generalization in statistical learning theory: The NFL theorem, as we formulated it, is closely related to the issue of generalization: Can the algorithm generalize knowledge gained from sites $x_{1:T-1}$ to a *new* site? NFL says that this is not possible without assumptions on the underlying function. On the surface this seems to contradict the classical foundation of statistical learning theory, stating that generalization to "new" data is possible without making assumptions about the underlying function. The origin of this seeming contradiction is simply the use of the word "new data" in both contexts. The prototypical setup in statistical learning theory considers a joint distribution $P(X, Y) = P(Y|X)\ P(X)$ from which data $\{(x_i, y_i)\}_{i=0}^N$ was sampled i.i.d. In that context, a "new" data point x^* is one that is equally sampled from the same source $P(X)$ as the previous data—*without ruling out revisitation of the same site*. Statements on generalization roughly state that, in the limit of large N, generalization to new data is possible. If the domain X is finite, the limit of large N implies that new data points are likely to coincide with previously observed sites and the possibility of generalization is obvious. If the domain is continuous, the chance to revisit exactly the same site is zero and it seems that revisitation is not an issue and NFL holds—however, as we discussed in the previous section, w.r.t. standard Lebesgue measures over functions, NFL does not hold in continuous spaces.

7.3 Some Background on POMDPs

Our formulation of NFL states that, assuming a fully factored distribution over functions, any search algorithm will have the same expected performance. Inversely this implies that when we *assume* a non-factored distribution over functions—which we call function *prior*—then the algorithm has at least a chance to exploit previous observations $x_{0:t}, y_{0:t}$ to decide "intelligently" (better than random search) about the next sample x_{t+1}.

Partial observable Markov decision processes (POMDP) give us a clear description of how an optimal (fully Bayes-rational) algorithm would choose the next sample point based on previous observations. The point in referring to POMDPs will *not* be that we will in practice be able to design fully Bayes-optimal search algorithms—this is in any realistic case computationally infeasible. However, the POMDP framework provides us with two important aspects: First the notion of a *belief*, which can be shown to subsume all the information from the history of observations $x_{0:t}, y_{0:t}$ that is necessary to make optimal decisions. And second, the POMDP framework provides us with promising approximate decision heuristics, for instance, iteratively using the optimal two-step look-ahead strategy as an approximation to the optimal T-step look-ahead for a problem of horizon T, as is discussed in detail in Sect. 7.5.

We briefly introduce POMDPs and the notion of beliefs in POMDPs here. For more details see [9]. A POMDP is a stochastic model of the interaction of an agent with an environment where the agent does not fully observe the state s_t of the environment but only has access to ("partial") observations y_t. For every time step t the environment is in state s_t, the agent chooses an action a_{t+1}, the world transitions into a new state according to a conditional probability $P(s_{t+1} \mid a_{t+1}, s_t)$, and the agent gets a new observation according to $P(y_{t+1} \mid s_{t+1}, a_{t+1})$.

Since each single observation y_t gives only partial information about the state, it is in general suboptimal for the agent to use only y_t to decide on an action a_{t+1}. A better alternative for the agent would be to take the full history $(y_{0:t}, a_{0:t})$ as input to choose an action—since this provides all the information accessible to the agent at the time this, in principle, supports choosing optimal actions. However, it can be shown [9] that a sufficient alternative input to choose optimal actions is the posterior distribution $P(s_t \mid y_{0:t}, a_{0:t})$. This should not be a surprise: given the Markovian structure of the world itself, if the agent *would* have access to the state s_t then optimal policy would map s_t directly to a_{t+1}. If, as in POMDPs, the agent

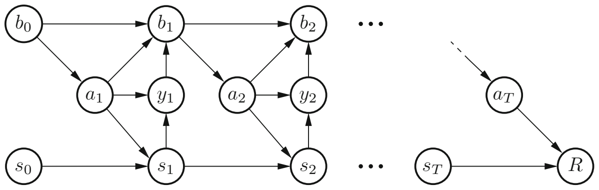


Fig. 7.1 Dynamic Bayesian network for the stochastic process of a (belief-based) agent interacting within a POMDP—for simplify in the case of finite horizon and final reward only

does not have access to s_t, then the state posterior $P(s_t \mid y_{0:t}, a_{0:t})$ provides all the information about s_t accessible to the agent, i.e., that can be inferred from previous observations and actions.

The state posterior is also called the *belief* $b_t = P(s_t \mid y_{0:t}, a_{0:t})$. To summarize, Fig. 7.1 illustrates the stochastic process of a (belief-based) agent interacting within a POMDP as a dynamic Bayesian network. The environment is described by the state transition probabilities $P(s_{t+1} \mid a_{t+1}, s_t)$, the observation probabilities $P(y_t \mid s_t, a_t)$, and the initial state distribution $P(s_0)$. The agent is described (in the belief-based case[3]) by the policy $\pi : b_t \mapsto a_{t+1}$ that maps the current belief state to the action. In each step, after executing action a_{t+1} and observing y_{t+1}, the agent updates the belief using Bayes' rule:

$$\begin{aligned} b_{t+1}(s_{t+1}) &= P(s_{t+1} \mid y_{0:t+1}, a_{0:t+1}) \\ &\propto P(y_{t+1} \mid s_{t+1}, y_{0:t}, a_{0:t})\ P(s_{t+1} \mid y_{0:t}, a_{0:t}) \\ &= P(y_{t+1} \mid s_{t+1}, a_t) \Big[\sum_{s_t} P(s_{t+1}, s_t \mid y_{0:t}, a_{0:t}) \Big] \\ &= P(y_{t+1} \mid s_{t+1}, a_t) \sum_{s_t} P(s_{t+1} \mid s_t, a_t)\ b_t(s_t) \end{aligned} \tag{7.5}$$

This equation is called *belief update*. The *prior belief* $b_0(s_0)$ is initialized with the initial state distribution $P(s_0)$.

7.4 From NFL to Beliefs and the Bayesian Search Game

Table 7.1 summarizes how one can draw a relation between the problem of optimization and POMDPs. The action a_t of the agent/algorithm correspond to the next site x_t that the algorithm explores. The state s_t of the environment corresponds to the unknown underlying function f—a difference here is that in POMDPs the

[3]Alternatives to represent agent policies are, for instance, finite state controllers [11].

Table 7.1 Translation of the search game as a partially observable Markov decision process

POMDP	Bayesian search game
World state s_t	Objective function f
Action a_t	Choice of search point x_t
Observation y_t	Function value $y_t = f(x_{t-1})$
Belief state $b_t = P(s_t \mid y_{0:t}, a_{0:t})$	Belief $b_t = P(f \mid y_{0:t}, x_{0:t})$

environment state is manipulated by actions, whereas in search exploring a site x_t does not change the function f of the environment. But as in a POMDP, the environment state f is not fully observable. Only a partial observation $P(y_t \mid f, x_t)$ is accessible to the agent/algorithm depending on the site it explores.

As in POMDPs, the belief $b_t(f)$ captures all information about the state f that is accessible to the algorithm. The $P(f)$ defined in the previous section provides the prior belief $b_0(f) := P(f)$; in Eq. (7.4) we also referred to the posterior belief at time t,

$$b_t(f) := P(f \mid x_{0:t}, y_{0:t}) \ . \tag{7.6}$$

NFL says that if the prior belief factorizes in identical marginals, then there is no way to derive a smart sampling heuristic from this belief. The reason is that in the NFL case the belief cannot be updated in a useful way. Why is this? Given a new observation y_t at x_t we can update the belief in the sense that now we explicitly know the function value at x_t—but we cannot update the belief about function values at yet unexplored sites because the NFL conditions do not allow us to generalize to unexplored sites. Hence the belief over yet unexplored sites always remains i.i.d. with marginals $h(y)$.

Inversely, the belief is a generic and exhaustive way to capture all of what we can possibly know about the underlying function given the observations made so far. In particular, when NFL condition (7.2) does *not* hold, then an observation y_t at some site x_t tells us something about the function at other sites. The belief state is an exact description of this information about yet unexplored sites.

The stochastic search processes of a belief-based algorithm, which pick new search points based on a policy $\pi_t : b_{t-1} \mapsto x_t$, can be depicted as the dynamic Bayesian network (DBN) as in Fig. 7.2. This process can be viewed as a (single-player) game: The game starts with the player picking a specific prior belief b_0 over the space of functions, and with the environment choosing a specific function f from some function prior $P(f)$. For simplification, we assume that the player is informed on $P(f)$ such that his prior belief coincides with the function prior, $b_0(f) = P(f)$. This initialization of the game corresponds to the first two nodes on the left in the DBN.

In the first time step, it will use the policy $\pi_t : b_{t-1} \mapsto x_t$ to pick a first site at time $x_1 = \pi_1(b_0)$. The environment responds by returning the function evaluation

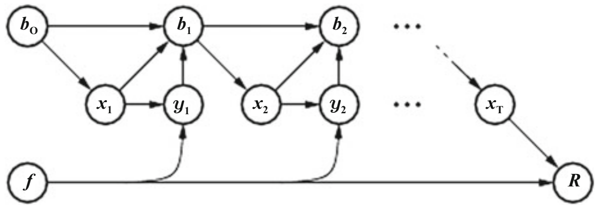


Fig. 7.2 The dynamic Bayesian network describing the search game. f is a function sampled from $P(f)$. b_t is a belief over functions that the player maintains; it is initialized deterministically to $b_0 = P(f)$. $x_t \sim \pi_t(b_t)$ is the player's sample action at time t, and $y_t = f(x_t)$ the evaluation feedback

$y_1 = f(x_1)$. The player updates its belief as in Eq. (7.5). Since in our case the function f is not influenced by the action x_t the belief update simplifies to

$$\begin{aligned} b_t(f) &= P(f \mid x_{0:t}, y_{0:t}) \\ &\propto P(y_t \mid f, x_{0:t}, y_{0:t-1})\; P(f \mid x_{0:t}, y_{0:t-1}) \\ &= P(y_t \mid x_t, f)\; b_{t-1}(f) \end{aligned} \tag{7.7}$$

The game continues like that until, at some final deadline T, a reward R is emitted depending only on the last sample.

Drawing the connection to POMDPs does not directly lead to new efficient solution methods. However, some simple facts from POMDPs also help us understand the problem of search better: (1) We know that the optimal policy in POMDPs is a deterministic function from beliefs to actions. This notion of optimality in POMDPs is very general and implies optimal solutions to the so-called exploration-exploitation problem [12] or strategies to gain information for later payoff. Clearly, such strategies are also relevant in the context of search. (2) A POMDP can be reformulated as a Markov decision process (MDP) with world state $\tilde{s}_t = (s_t, b_t)$—that is, when we think of the tuple (s_t, b_t) as the new (embedding) world state. This also implies that optimal policies can be found by computing a value function $V(s_t, b_t)$ over this embedding space. Note that this value function is a function over the space of distributions—and thereby of extremely high complexity. Point-based value iteration methods follow this approach by exploiting a sparse structure of the value function [9].

7.5 Belief-Based Search Policies

In this section we consider some basic heuristic policies to choose the next search point based on the current belief. For simplicity we consider a finite horizon T where the reward is exactly the function value $f(x_T) \in \mathbb{R}$ of the last site. The objective is then: Find a policy $\pi_t \;:\; b_t \mapsto x_{t+1}$ (different for each t) that maximizes the expectation of $f(x_T)$.

The problem the player is faced with is obviously a problem of planning ahead, i.e., taking samples that allow him to learn as much as possible about f (shaping its belief favorably) such that at the deadline T he is as well informed as possible to pick the final sample. But what are *computationally feasible* policies in practice? Let us define some basic policies here:

The k-step look-ahead policy: $\pi^k \;:\; b_t \mapsto x_{t+1}$ is defined as the optimal policy for picking x_{t+1} based on b_t, assuming that the horizon $T = t + k$ is k steps ahead. For large k, computing this policy is infeasible. For $k = 1$ or $k = 2$ approximations may be feasible.

The greedy policy: $\pi^{k=1}$ is the one-step lookahead policy which picks the point x_{t+1} that maximizes the predictive mean $\hat{f}_t(x) = \int_f f(x)\; b_t(f)\; df$, that is, the mean function given the current belief b_t,

$$\pi^{k=1}(b_t) = \text{argmax}_x \hat{f}_t(x)\;. \tag{7.8}$$

Thereby, the greedy policy is the optimal policy for the final pick of x_T based on b_{T-1}.

The two-step look-ahead: This policy $\pi^{k=2}$ is

$$\pi^{k=2}(b_t) = \text{argmax}_{x_{t+1}} \int_{y_{t+1}} \max_{x_{t+2}} \hat{f}_{t+1}(x_{t+2})\; P(y_{t+1} \,|\, x_{t+1}, b_t) \tag{7.9}$$

$$\hat{f}_{t+1} = \int_f f(x) b(f; y_{t+1}, x_{t+1}, b_t) df \tag{7.10}$$

where $b(f; y_{t+1}, x_{t+1}, b_t)$ is the belief when updating b_t with the new observations according to Eq. (7.7). The term $\max_{x_{t+2}} \hat{f}_{t+1}(x_{t+2})$ is the expected reward when the greedy policy is applied in the next step. The integral over y_{t+1} accounts for all possible outcomes (and corresponding belief updates) for the sample x_{t+1}. In that sense, the two-step look-ahead policy can imply explorative strategies: One might want to pick x_{t+1} such that the outcome y_{t+1} contains crucial information for the belief update such that the final (greedy) pick has maximal expected reward.

A simple **exploration** policy π^{explore} is to always pick the site x_{t+1} that maximizes the predictive variance $\hat{\sigma}_t(x)^2 = \int_f [f(x) - \hat{f}_t(x)]^2 b_t(f) df$ of the belief. This strategy aims at learning as much as possible about the function, but neglects that we are interested in *high* function values and should thus learn as much as possible about regions where we hope to find high function values.

A simple **exploit-explore** policy π^{EE} is to pick the site x_{t+1} that maximizes $g_t(x) = \hat{f}_t(x) + \alpha\sigma_t(x)$, that is, a combination of the predictive mean and variance. $g_t(x)$ can be interpreted as an optimistic function value estimate: The value could potentially be α standard deviations above the current mean estimation.

Another heuristic combining exploration and exploitation is the **expected improvement** policy π^{EI}. Let $Y_t = \max\{y_{1:T}\}$ be the maximum value observed so far. We can compute for each site x the expected improvement $q_t(x) = \int_f f(x)\delta_{f(x)>Y_t} b_t(f)df$, where δ is the indicator function. This expected improvement computes a mean value, as with $\hat{f}_t$, but only over function values greater than Y_t.

7.6 Experiments with Gaussian Processes as Belief Representation

The belief update in Eq. (7.7) is a simple equation, but for a concrete algorithm it requires to represent a distribution over function space and be able to multiply the *likelihood* term $P(y_t \mid x_t, f)$ to the belief to become a new belief. What is a family of distributions over functions which we can be represented in computers and which is conjugate (that is, if the old belief is an element of this family, then the updated belief is also an element of this family)?

Gaussian processes [13] are such a family of distributions over continuous functions. They can be defined as follows. Let $f \sim GP(\mu, C)$ be a random function sampled from a Gaussian process with mean function $\mu(x)$ and covariance function $C(x, x')$. Then, for any finite set of points $\{x_1, \dots, x_N\}$, the vector $(f(x_1), \dots, f(x_N))$ is distributed joint Gaussian with mean vector $(\mu(x_1), \dots, \mu(x_N))$ and covariance matrix $C(x_i, x_j)$, $i, j = 1 \dots N$. Since this definition describes the behavior of random functions on finite subsets it fits nicely with our formulation of NFL.

The covariance function $C(x, x')$ is typically decaying with the distance $|x - x'|$ such that points close to each other are strongly correlated. This leads to smooth functions. Often $C(x, x')$ is chosen squared exponential $C(x, x') = \nu^2 \exp\{-(x - x')^2/2\nu^2\} + \delta_{x=x'}\varrho^2$ with correlation bandwidth ν (and observation standard deviation ϱ). Figure 7.3 displays a number of functions sampled independently from a GP prior with constant mean $\mu(x) = 0$ and bandwidth $\nu = \frac{1}{2}$. This should illustrate what it means to assume such a prior: We believe a priori that functions typically look like those in Fig. 7.3, in particular w.r.t. the type of smoothness. (GPs are related to cubic splines, see [13].)

It is a common approach to use GPs as a representation of the belief b for search and optimization problems; in geology this method is also called *kriging* [2, 6, 7]. One often assumes that a single function evaluation is expensive (e.g., drilling a hole to get a geological probe) and therefore extensive computational cost to evaluate a policy is acceptable.

To demonstrate the use of Gaussian processes to represent beliefs we implemented a Bayesian search game, which can be downloaded from the author's

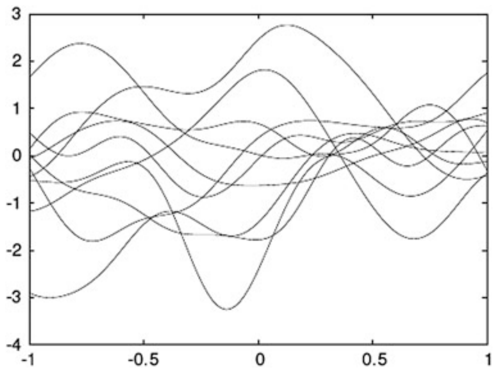


Fig. 7.3 Ten sample functions from a Gaussian process prior with bandwidth $\nu = \frac{1}{2}$

Table 7.2 Performance for different policies for finding the optimum of a function sampled from a GP prior with bandwidth $\nu = \frac{1}{2}$, constrained to the search interval $[-1, 1]$. We measure the loss as the difference between the last sampled value $f(x_T)$ and the true optimum of the function. The algorithm is only allowed to take $T = 10$ (respectively, $T = 5$) samples. Mean and standard deviation are given for 10,000 random functions

Policy	Final loss for $T = 10$	Avg loss ($T = 10$)	Final loss for $T = 5$	Avg loss ($T = 5$)
$\pi^{k=1}$	0.632 ± 0.006	0.764 ± 0.006	0.648 ± 0.006	0.891 ± 0.006
$\pi^{\text{EE}}, \alpha = 1$	0.051 ± 0.002	**0.492 ± 0.003**	**0.254 ± 0.004**	**0.834 ± 0.005**
$\pi^{\text{EE}}, \alpha = 2$	0.0039 ± 0.0004	0.687 ± 0.002	**0.256 ± 0.004**	0.970 ± 0.004
$\pi^{\text{EE}}, \alpha = 4$	0.0026 ± 0.0001	0.952 ± 0.003	0.296 ± 0.004	1.079 ± 0.004
π^{EI}	**0.0015 ± 0.0001**	0.926 ± 0.003	0.299 ± 0.004	1.063 ± 0.005
π^{explore}	**0.0015 ± 0.0001**	0.926 ± 0.003	0.303 ± 0.004	1.069 ± 0.005

webpage.[4] Here we report on some quantitative experiments. We implemented the policies $\pi^{k=1}$, $\pi^{k=2}$, π^{EE}, π^{EI}, π^{explore} simply by evaluating the respective integrals over a grid. This becomes expensive already for $k = 2$.

We performed some experiments with Gaussian process beliefs to illustrate and evaluate the different policies defined in the previous section. The objective is to find the optimum of a function sampled from a Gaussian process with bandwidth $\nu = \frac{1}{2}$. The search is constrained to the interval $[-1, 1]$.

Table 7.2 displays the results when we allow the algorithms to take only $T = 10$ or $T = 5$ samples to find the optimum. The objective is the final loss: the difference between the last sampled value $f(x_T)$ and the true optimum of the function. We also report on the average loss during the T samples. Although this is not the objective it indicates whether the algorithm tends to sample good points also in intermediate steps.

[4]http://userpage.fu-berlin.de/mtoussai/07-bsg/

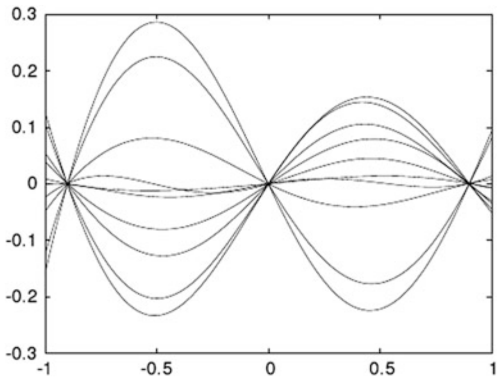


Fig. 7.4 Ten sample functions from a Gaussian process prior with bandwidth $\nu = 1$ conditioned on $f(x) = 0$ for $x = -0.9, 0, 0.9$

Table 7.3 Performance for different policies for finding the optimum of a function sampled from a GP prior illustrated in Fig. 7.4. The algorithm is only allowed to take $T = 2$ samples. Mean and standard deviation are given for 10,000 random functions

Policy	Final loss for $T = 2$
$\pi^{k=1}$	0.0144 ± 0.0003
$\pi^{\text{EE}}, \alpha = 1$	0.0116 ± 0.0002
$\pi^{\text{EE}}, \alpha = 2$	0.0116 ± 0.0002
$\pi^{\text{EE}}, \alpha = 4$	0.0116 ± 0.0002
π^{EI}	0.0116 ± 0.0002
π^{explore}	0.0116 ± 0.0002
$\pi^{k=2}$	**0.0095 ± 0.0002**

For $T = 10$ we find that the expected improvement policy π^{EI} and the simple exploration policy π^{explore} perform best. Both of them are rather exploratory, which is evident also from the high average loss. In contrast, π^{EE} with $\alpha = 1$ is less exploratory, focuses on a (local) optimum earlier, leading to higher final loss but lower average loss. For comparison we also tested for only $T = 5$, where the greedier π^{EE} with $\alpha = 1$ performs slightly better than the other policies.

Finally, we also want to demonstrate the effect of two-step look-ahead planning. It is not easy to find a problem class for which this policy performs better than the others. Here is a slightly contrived example: We sampled random functions from a GP prior with large bandwidth $\nu = 1$ (very smooth functions) which were additionally conditioned on $f(x) = 0$ at the sites $x = -0.9, 0, 0.9$. Figure 7.4 displays 10 random samples from this prior.

Table 7.3 displays the results for all policies with only $T = 2$—that is, the algorithm only has one sample to learn as much as possible about the function before placing the final sample, which decides on the final loss. First, we find that all algorithms π^{EE}, π^{EI}, π^{explore} have the same performance. This is because they all first sample the site $x = -0.45$ or $x = 0.45$, which have maximal entropy, and then sample the last point using the greedy policy. Hence, they are all equivalent for $T = 2$.

The two-step look-ahead policy behaves differently: It first samples a point very near by $x = 0$ (approximately $x = 0.05$). The observation at this point implicitly allows the algorithm to infer the *slope* of the true function around $x = 0$. This implies a "better informed" GP posterior of the function, which has more certainty about the function on both sides rather than only on one side of $x = 0$. As a consequence, the final (greedy) pick of x_T is better than with the other algorithms.

This rather contrived example demonstrates, on the one hand, the intricate implications of lookahead strategies—how they pick points based on how their knowledge for future picks is improved. On the other hand, the minimal differences in performance and given that we had to construct such complicated scenarios to demonstrate the advantage of a two-step look-ahead strategy argues against such strategies. Note that $\pi^{k=2}$ is computationally orders of magnitude slower than the other policies.

7.7 Discussion

In this chapter we presented a discussion of three seemingly unconnected topics: No Free Lunch, POMDPs, and Gaussian processes. However, we hope it became clear that these topics are closely related.

NFL & Gaussian processes: In our formulation, the NFL condition (7.2) is that the function prior identically factorizes on any finite subset $\{x_1, .., x_K\} \subset X$. Only if this condition is violated can we hope for an efficient search algorithm. Violation of this constraint implies that function values on finite subsets are dependent—a Gaussian process by definition describes exactly this correlation of values on finite subsets. Therefore, in our view, a Gaussian process is a very natural and simple model of the violation of NFL conditions. At this point one should note that, although Gaussian processes are typically formulated for continuous X with continuous covariance function, they can of course also be applied on discrete spaces, e.g., with a covariance function depending on the Hamming distance or other similarity measures.

NFL & POMDPs: The reason we discussed POMDPs in the context of NFL is that the POMDP framework explicitly states what the optimal search algorithm *would* be. In particular, the POMDP framework clarifies that the notion of a belief is a sufficient representation of all the knowledge gained from previous explorations, in the sense that the optimal algorithm can be viewed as a policy mapping from the belief to a new search point. Generally, we do not want to over-stress the discussion of truly optimal search algorithms. The POMDP framework formulated here leads to optimal search (given the assumption of the prior belief). Hutter [4] has discussed universal optimality, where the role of the prior is replaced by a complexity measure over algorithms (Solomonoff complexity). In both cases the computational complexity of evaluating the optimal policy is exponential and the key is to have good approximate policies. However, the notion of the belief leads naturally to the existing literature on optimization using heuristics like the expected improvement policy.

Let us also mention estimation of distribution algorithms (EDAs) [8]. It has been argued before that EDAs implicitly learn about the problem by shaping the search distribution [14]. From our perspective, EDAs (and also genetic algorithms) try to perform two tasks at once with the search distribution: They use it to accumulate information about the problem (representing where optima might be), and they use it to describe the next sample point. The belief framework suggests to disentangle these two issues: The belief is used to represent all knowledge and a separate policy maps it to a new samples. From a Bayesian perspective the benefit is that there is no loss in information in the belief update.

Finally, let us discuss related literature. Gutmann [2], Jones [7], and Jones et al. [6] discuss *global* optimization using response surfaces (also called surrogates, or kriging). Our Gaussian process search algorithm is an instance of such global response surface modelling. However, this work has not made the connection to POMDPs, NFL and look-ahead planning. Only the maximizing immediate measures (figures of merit) like the *expected improvement* has been discussed in this context.

Another branch of research focuses on *local* models of the fitness function [3, 10, 15]. These methods are very effective when many samples can be taken (where a global model would become infeasible). However, look-ahead heuristic or a well-defined Bayesian belief update has not been discussed in this context.

Acknowledgements This research was supported by the German Research Foundation (DFG), Emmy Noether fellowship TO 409/1-3.

References

1. A. Auger, O. Teytaud, Continuous lunches are free plus the design of optimal optimization algorithms. Algorithmica **57**(1), 121–146 (2010)
2. H. Gutmann, A radial basis function method for global optimization. J. Glob. Optim. **19**, 201–227 (2001)
3. N. Hansen, A. Ostermeier, Completely derandomized self-adaption in evolutionary strategies. Evol. Comput. **9**, 159–195 (2001)
4. M. Hutter, Towards a universal theory of artificial intelligence based on algorithmic probability and sequential decision theory. arXiv: cs.AI/0012011 (2000)
5. C. Igel, M. Toussaint, A no-free-lunch theorem for non-uniform distributions of target functions. J. Math. Model. Algorithms **3**, 313–322 (2004)
6. D. Jones, M. Schonlau, W. Welch, Efficient global optimization of expensive black-box functions. J. Glob. Optim. **13**, 455–492 (1998)

7. D.R. Jones, A taxonomy of global optimization methods based on response surfaces. J. Glob. Optim. **21**, 345–383 (2001)
8. M. Pelikan, D.E. Goldberg, F. Lobo, A survey of optimization by building and using probabilistic models. Technical Report IlliGAL-99018, Illinois Genetic Algorithms Laboratory, 1999
9. J. Pineau, G. Gordon, S. Thrun, Anytime point-based approximations for large POMDPs. J. Artif. Intell. Res. **27**, 335–380 (2006)
10. J. Poland, Explicit local models: towards optimal optimization algorithms. Technical Report No. IDSIA-09-04, 2004
11. P. Poupart, C. Boutilier, Bounded finite state controllers, in *Advances in Neural Information Processing Systems 16 (NIPS 2003)*, Vancouver, vol. 16 (MIT Press, 2004)
12. P. Poupart, N. Vlassis, J. Hoey, K. Regan, An analytic solution to discrete Bayesian reinforcement learning, in *Proceeding of the 23rd International Conference on Machine Learning (ICML 2006)*, Pittsburgh, 2006, pp. 697–704
13. C.E. Rasmussen, C. Williams, *Gaussian Processes for Machine Learning* (MIT Press, Cambridge, 2006)
14. M. Toussaint, Compact representations as a search strategy: compression EDAs. Theor. Comput. Sci. **361**, 57–71 (2006)
15. H. Ulmer, F. Streichert, A. Zell, Optimization by Gaussian processes assisted evolution strategies, in *International Conference on Operations Research (OR 2003)* (Springer, Heidelberg, 2003) pp. 435–442
16. D.H. Wolpert, W.G. Macready, No free lunch theorems for optimization. IEEE Trans. Evol. Comput. **1**(1), 67–82 (1997)

Chapter 8
Principled Design of Continuous Stochastic Search: From Theory to Practice

Nikolaus Hansen and Anne Auger

Abstract We derive a stochastic search procedure for parameter optimization from two first principles: (1) imposing the least prior assumptions, namely by maximum entropy sampling, unbiasedness and invariance; (2) exploiting all available information under the constraints imposed by (1). We additionally require that two of the most basic functions can be solved reasonably fast. Given these principles, two principal heuristics are used: reinforcing of good solutions and good steps (increasing their likelihood) and rendering successive steps orthogonal. The resulting search algorithm is the *covariance matrix adaptation evolution strategy*, CMA-ES, that coincides to a great extent to a natural gradient descent. The invariance properties of the CMA-ES are formalized, as are its maximum likelihood and stationarity properties. A small parameter study for a specific heuristic—deduced from the principles of reinforcing good steps and exploiting all information—is presented, namely for the cumulation of an evolution or search path. Experiments on two noisy functions are provided.

8.1 Introduction: Top-Down Versus Bottom-Up

Let $f : \mathbb{R}^n \to \mathbb{R}, \boldsymbol{x} \mapsto f(\boldsymbol{x})$ be an *objective* or *cost* (or fitness) function to be minimized, where, in practice, the typical search space dimension n obeys $3 < n < 300$. When properties of f are unknown a priori, an iterative search algorithm can proceed in evaluating solutions on f and so gather information for finding better solutions over time (black-box search or optimization). Good solutions have, by definition, a small f-value, and evaluations of f are considered as the *cost of search* (note the double entendre of the word cost for f). The objective is, in practice,

N. Hansen (✉) · A. Auger
INRIA Saclay – Île-de-France, Orsay, France
e-mail: Nikolaus.Hansen@inria.fr; anne.auger@inria.fr

```
Given: a cost function $f$, a parametrized family of distributions $P(\theta)$, and $\lambda \in \mathbb{N}$
Initialize: $k \leftarrow 0$, set $\theta_k$
Repeat while not happy
        Sample: $x_1, \ldots, x_\lambda \sim P(\theta_k)$ i.i.d.
        Update: $\theta_{k+1} = \mathit{Update}(\theta_k, x_1, \ldots, x_\lambda, f(x_1), \ldots, f(x_\lambda))$
        $k \leftarrow k + 1$
```

Fig. 8.1 Stochastic search template

to find a good solution with the least number of function evaluations and, more rigorously, to generate a sequence $\boldsymbol{x}_k$, $k = 1, 2, 3, \ldots$, such that $f(\boldsymbol{x}_k)$ converges fast to the *essential infimum* of f, denoted f^*. The essential infimum f^* is the largest real number such that the set of better search points $\{\boldsymbol{x} \in \mathbb{R}^n : f(\boldsymbol{x}) < f^*\}$ has zero volume.

In order to search in continuous spaces with even moderate dimension, some structure in the cost function needs to be exploited. For *evolution strategies*, the principle structure is believed to be *neighborhood*. Strong causality [33]—the principle that small actuator changes have generally only small effects—and fitness-distance correlation [31]—a statistical perspective of the same concept—are two ways to describe the structure that evolution strategies are based upon. In contrast to Chaps. 4, 6, and 7 of this volume, in this chapter we do not introduce an a priori assumption on the problem class we want to address, that is, we do not assume any structure in the cost function a priori. However, we use two ideas that might imply the exploitation of neighborhood: We assume that the variances of the sample distribution exist, and we encourage consecutive iteration steps to become, under a variable metric, orthogonal (via step-size control). Empirically, the latter rather reduces the locality of the algorithm: The step-sizes that achieve orthogonality are usually large in their stationary condition. We conjecture therefore that either the mere existence of variances and/or the "any-time" approach that aims to improve in each iteration, rather than only in a final step, implies already the exploitation of a neighborhood structure in our context.

In order to solve the above-introduced search problem on f, we take a principled *stochastic* (or *randomized*) approach. We first **sample** points from a distribution over the search space with density $p(.|\theta)$, we **evaluate** the points on f and finally **update** the parameters θ of the distribution. This is done iteratively and defines a search procedure on θ as depicted in Fig. 8.1. Indeed, the update of θ remains the one and only crucial element—besides the choice of p (and λ) in the first place. Consequently, this chapter is entirely devoted to the question of how to update θ.

Before we proceed, we note that under some mild assumptions on p, and for any increasing transformation $g : \mathbb{R} \to \mathbb{R}$ (in particular also for the identity), the minimum of the function

$$\theta \mapsto E(g(f(\boldsymbol{x}))|\theta) \tag{8.1}$$

coincides with the minimum of f (the expectation E is taken under the sample distribution p, given parameters θ). The **optimal distribution** is entirely concentrated in the arg min of f. In black-box search, we do not want (and are not able) to impose strong regularity conditions on the unknown function f. However, we have entire control over p. This seems an excellent justification for a *randomized* approach to the original black-box search problem. We sketch two approaches to solve (8.1).[1]

8.1.1 The Top-Down Way

We might chose p being "sufficiently smooth" and conduct a gradient descent,

$$\theta_{k+1} = \theta_k - \eta \nabla_\theta E(f(\boldsymbol{x})|\theta) \qquad \text{with } \eta > 0 \ . \tag{8.2}$$

We are facing two problems with Eq. (8.2). On the one hand, we need to compute $\nabla_\theta E(f(\boldsymbol{x})|\theta)$. On the other hand, the gradient ∇_θ strongly depends on the specifically chosen parameterization in θ. The unique solution to the second problem is the *natural gradient*. The idea to use the natural gradient in evolution strategies was coined in [40] and elegantly pursued in [11]. The natural gradient is unique, invariant under reparametrization and in accordance with the Kullback-Leibler (KL) divergence or relative entropy, the informational difference measure between distributions. We can reformulate Eq. (8.2) using the natural gradient, denoted $\tilde{\nabla}$, in a unique way as

$$\theta_{k+1} = \theta_k - \eta \tilde{\nabla} E(f(\boldsymbol{x})|\theta) \ . \tag{8.3}$$

We can express the natural gradient in terms of the vanilla gradient ∇_θ, using the Fisher information matrix, as $\tilde{\nabla} = F_\theta^{-1}\nabla_\theta$. Using the log-likelihood trick, $\nabla_\theta p = (p/p)\nabla_\theta p = p\nabla_\theta \log p$ we can finally, under mild assumption on p, re-arrange Eq. (8.3) into

$$\theta_{k+1} = \theta_k - \eta E(\underbrace{f(\boldsymbol{x})}_{\text{expensive}} \overbrace{F_\theta^{-1}\nabla_\theta \log p(\boldsymbol{x}|\theta)}^{\text{“controlled”}}) \ . \tag{8.4}$$

In practice, the expectation in Eq. (8.4) can be approximated/replaced by taking the average over a (potentially small) number of samples, $\boldsymbol{x}_i$, where computing $f(\boldsymbol{x}_i)$ is assumed to be the costly part. We will also choose p such that we can conveniently sample from the distribution and that the computation (or approximation) of $F_\theta^{-1}\nabla_\theta \log p$ is feasible. The top-down way of Eqs. (8.3) and (8.4) is an amazingly clean and principled approach to stochastic black-box optimization.

[1]That is, to find a sequence θ_k, $k = 1, 2, 3, \ldots$, such that $\lim_{k\to\infty} E(f(\boldsymbol{x}|\theta_k)) = f^*$.

8.1.2 The Bottom-Up Way

In this chapter, we choose a rather orthogonal approach to derive a principled stochastic search algorithm in the $\mathbb{R}^n$. We take a scrutinizing step-by-step road to construct the algorithm based on a few fundamental principles—namely maximal entropy, unbiasedness, maintaining invariance, and, under these constraints, exploiting all available information and solving simple functions reasonably fast.

Surprisingly, the resulting algorithm arrives at (8.3) and (8.4): Eqs. (8.12) and (8.51) implement Eq. (8.3) in the manifold of multivariate normal distributions under some monotonic transformation of f [1, 5] (let $\eta = 1$, $c_1 = c_\epsilon = 0$, $c_\mu = \sigma_k = 1$). The monotonic transformation is driven by an invariance principle. In both ways, top-down and bottom-up, the same, well-recognized stochastic search algorithm *covariance matrix adaptation evolution strategy* (CMA-ES) emerges. Our scrutinizing approach, however, reveals additional aspects that are consistently useful in practice: Cumulation via an evolution path, step-size control and different learning rates η for different parts of θ. These aspects are either well hidden by Eq. (8.4)[2] or can hardly be derived at all (cumulation). On the downside, the bottom up way is clearly less appealing.

The following sections will introduce and motivate the CMA-ES step-by-step. The CMA-ES samples new solutions from a multivariate normal distribution and updates the parameters of the distribution, namely the mean (incumbent solution), the covariance matrix and additionally a step-size in each iteration, utilizing the f-ranking of the sampled solutions. We formalize the different notions of invariance as well as the maximum likelihood and stationarity properties of the algorithm. A condensed final transcription of the algorithm is provided in the appendix. For a discussion under different perspectives, the reader is referred to [12, 15, 22].

8.2 Sampling with Maximum Entropy

We start by sampling λ (new) candidate solutions $\boldsymbol{x}_i \in \mathbb{R}^n$, obeying a multivariate normal (search) distribution

$$\boldsymbol{x}_i \sim \boldsymbol{m}_k + \sigma_k \times \mathcal{N}_i(\boldsymbol{0}, \boldsymbol{C}_k) \qquad \text{for } i = 1, \ldots, \lambda, \tag{8.5}$$

where $k = 0, 1, 2, \ldots,$ is the time or iteration index, and $\boldsymbol{m}_k \in \mathbb{R}^n$, $\sigma_k > 0$, and $\mathcal{N}(\boldsymbol{0}, \boldsymbol{C})$ denotes a multivariate normal distribution with zero mean and covariance matrix $\boldsymbol{C}$, $\sim$ denotes equality in distribution. For convenience, we will sometimes omit the iteration index k.

[2]Different learning rates might be related to some parameters in the distribution being orthogonal.

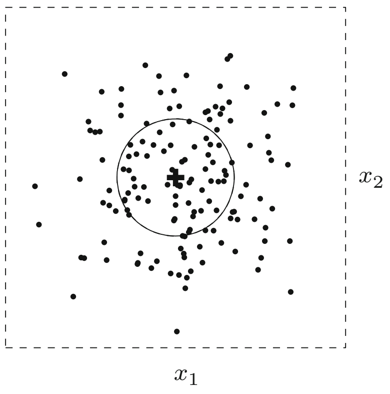


Fig. 8.2 One hundred and fifty samples from a multivariate (standard) normal distribution in 2-D. Both coordinates are i.i.d. according to a standard normal distribution. The *circle* depicts the one-σ equal density line, the *center of the circle* is the mean and modal value at zero. In general, lines of equal density (level sets) are ellipsoids. The probability to sample a point outside the *dashed box* is close to $1 - (1 - 2 \times 0.0015)^2 \approx 1/170$

New solutions obey a multivariate normal distribution with expectation $\boldsymbol{m}$ and covariance matrix $\sigma^2 \times \boldsymbol{C}$. Sets of equal density—that is, lines or surfaces in 2-D or 3-D, respectively—are ellipsoids centered about the mean and modal value $\boldsymbol{m}$. Figure 8.2 shows 150 sampled points from a standard (2-variate) normal distribution, $\mathcal{N}(\mathbf{0}, \mathbf{I})$.

Given mean, variances and covariances of a distribution, the chosen multivariate normal distribution has **maximum entropy** and—without any further knowledge—suggests itself for randomized search. We explain Eq. (8.5) in more detail.

- The distribution mean value, $\boldsymbol{m}$, is the **incumbent** solution of the algorithm: It is the current estimate for the global optimum provided by the search procedure. The distribution is point symmetrical about the incumbent. The incumbent $\boldsymbol{m}$ is (usually) not evaluated on f. However, it should be evaluated as final solution in the last iteration.
- New solutions are obtained by disturbing $\boldsymbol{m}$ with the *mutation distribution*

$$\mathcal{N}\big(\mathbf{0}, \sigma^2\boldsymbol{C}\big) \equiv \sigma \times \mathcal{N}(\mathbf{0}, \boldsymbol{C}) \ , \tag{8.6}$$

where the equivalence holds by definition of $\mathcal{N}(.,.)$. The parameter $\sigma > 0$ is a *step-size* or scale parameter and exists for notational convenience only. The *covariance matrix* $\boldsymbol{C}$ has $\frac{n^2+n}{2}$ degrees of freedom and represents a full quadratic model.

The covariance matrix determines the *shape* of the distribution, where level-sets of the density are hyper-ellipsoids (refer to [12, 15] for more details). On convex quadratic cost functions, $\boldsymbol{C}$ will closely align with the inverse Hessian of the cost function f (up to a scalar factor). The matrix $\boldsymbol{C}$ defines a *variable neighborhood* metric. The above-said suggests that using the maximum entropy distribution with finite variances implies the notion, and underlines the importance of *neighborhood*.

The initial incumbent $\boldsymbol{m}_0$ needs to be provided by the user. The algorithm has no preference for any specific value and its operations are *invariant* to the value of $\boldsymbol{m}_0$ (see translation invariance in Sect. 8.4).

Equation (8.5) implements the principle of **stationarity** or **unbiasedness**, because the expected value of Eq. (8.6) is zero. Improvements are not a priori made *by construction*, but only after sampling *by selection*. In this way, the **least additional assumptions** are built into the search procedure.

The number of candidate solutions sampled in Eq. (8.5) cannot be entirely derived from first principles. For small $\lambda \not\gg n$ the search process will be comparatively local and the algorithm can converge quickly. Only if previously sampled search points are considered, λ could be chosen to its minimal value of one—in particular if the best so-far evaluated candidate solution is always retained. We tend to disregard previous samples entirely (see below). In this case, a selection must take place between $\lambda \geq 2$ new candidate solutions. Because the mutation distribution is unbiased, newly sampled solutions tend to be worse than the previous best solution, and in practice $\lambda \geq 5$ is advisable.[3]

On the other hand, for large $\lambda \gg n$, the search becomes more global and the probability to approach the desired, global optimum on multimodal functions is usually larger. On the downside, more function evaluations are necessary to closely approach an optimum even on simple functions.

Consequently, a comparatively successful overall strategy runs the algorithm first with a small population size, e.g., the default $\lambda = 4 + \lfloor 3 \ln n \rfloor$, and afterwards conducts independent restarts with increasing population sizes (IPOP) [6].

After we have established the sampling procedure using a parameterized distribution, we need to determine the *distribution parameters* which are essential to conduct efficient search. All parameters depend explicitly or implicitly on the past and therefore are described in their update equations.

8.3 Exploiting the Objective Function

The pairs $(\boldsymbol{x}_i, f(\boldsymbol{x}_i))_{i=1,\ldots,\lambda}$ provide the information for choosing a new and better incumbent solution $\boldsymbol{m}_{k+1}$ as well as the new distribution covariance matrix $\sigma^2\boldsymbol{C}$. Two principles are applied.

[3]In the (μ, λ)-ES, only the μ best samples are selected for the next iteration. Given $\mu = 1$, a very general optimality condition for λ states that the currently second best solution must resemble the f-value of the previous best solution [24]. Consequently, on any linear function, $\lambda = 2$ and $\lambda = 3$ are optimal [24, 36]. On the sphere function Eq. (8.22), $\lambda = 5$ is optimal [33]. On the latter, $\lambda \approx 3.7\mu$ can also be shown optimal for $\mu \geq 2$ and equal recombination weights [9], compare (8.12). For $\lambda < 5$, the original strategy parameter setting for CMA-ES has been rectified in [10], but only mirrored sampling leads to satisfactory performance in this case [10].

8.3.1 Old Information Is Disregarded

There are a few reasons to believe that old information can or should be disregarded.

(a) The given $(n^2 + 3n)/2$ distribution parameters, $\boldsymbol{m}$ and $\sigma^2 \times \boldsymbol{C}$, should already capture all necessary previous information. Two additional state variables, the search paths $\boldsymbol{p}^\sigma, \boldsymbol{p}^\mathrm{c} \in \mathbb{R}^n$, will provide another $2n$ parameters. Theoretical results suggests that only slight improvements can be made by storing and using (all) previously sampled candidate solutions [38, 39], given rank-based selection.
(b) Convergence renders previously sampled solutions rather meaningless, because they are too far away from the currently focused region of interest.
(c) Disregarding old solutions helps to avoid getting trapped in local optima.
(d) An elitist approach can be destructive in the presence of noise, because a supersolution can stall any further updates. Under uncertainties, any information must be used with great caution.

8.3.2 Ranking of the Better Half Is Exploited

Only the *ranking of the better half* of the new candidate solutions is exploited. Function *values* are discarded as well as the ranking of the worse half of the newly sampled points. Specifically, the function f enters the algorithm only via the indices $i : \lambda$ for $i = 1, \ldots, \mu$, in that (serving as definition for $i : \lambda$)

$$f(\boldsymbol{x}_{1:\lambda}) \le f(\boldsymbol{x}_{2:\lambda}) \le \cdots \le f(\boldsymbol{x}_{\lambda:\lambda}) \tag{8.7}$$

is satisfied. We choose $\mu = \lfloor \lambda/2 \rfloor$, because

(a) On a linear function in expectation the better *half* of the new solutions improve over $\boldsymbol{m}_k$ and for the same reason
(b) On the quadratic sphere function only the better half of the new solutions can improve the performance, using positive recombination weights (see Eq. (8.12) below). For the remaining solutions, $\boldsymbol{x}_{i:\lambda} - \boldsymbol{m}_k$ needs to enter with a negative prefactor [3].

We feel that using worse points to make predictions for the location of better points might make a too strong assumption on the regularity of f in general. Indeed, optimization would be a much easier task if outstandingly bad points would allow generally valid implications on the location of good points, because bad points are generally easy to obtain.

On the highly symmetrical, isotropic sphere model, using the worse half of the points with the same importance than the better half of the points for calculating the new incumbent can render the convergence two times faster [2, 3]. In experiments with CMA-ES, we find the factor to be somewhat smaller and obtain very similar results also on the isotropic, highly multimodal Rastrigin function. On most anisotropic functions we observe performance degradations and also failures in rare cases and in cases with noise. The picture, though, is more encouraging for a covariance matrix update with negative samples, as discussed below.

Because only the f-ranked solution points (rather than the f-values) are used, we denote the f-ranking also as (rank-based) *selection*. The exploitation of available information is quite conservative, reducing the possible ways of deception. As an additional advantage, function values do not need to be available (for example, when optimizing a game-playing algorithm, a passably accurate selection and ranking of the μ best current players suffices to proceed to the next iteration). This leads to a strong robustness property of the algorithm: Invariance to order-preserving transformations, see next section. The downside of using only the f-ranking is that the possible convergence speed cannot be faster than linear [7, 28, 38].

8.4 Invariance

We begin with a general definition of invariance of a search algorithm $\mathcal{A}$. In short, invariance means that $\mathcal{A}$ does not change its behavior under exchange of f with an equivalent function $h \in \mathcal{H}(f)$, in general conditionally to change of the initial conditions.

Definition 8.1 (Invariance). Let $\mathcal{H}$ be a mapping from the set of all functions into its power set, $\mathcal{H} : \{\mathbb{R}^n \to \mathbb{R}\} \to 2^{\{\mathbb{R}^n \to \mathbb{R}\}}$, $f \mapsto \mathcal{H}(f)$. Let S be the state space of the search algorithm, $s \in S$ and $\mathcal{A}_f : S \to S$ an iteration step of the algorithm under objective function f. The algorithm $\mathcal{A}$ **is invariant under** $\mathcal{H}$ (in other words: Invariant under the exchange of f with elements of $\mathcal{H}(f)$) if for all $f \in \{\mathbb{R}^n \to \mathbb{R}\}$, there exists for all $h \in \mathcal{H}(f)$ a bijective state space transformation $T_{f\to h} : S \to S$ such that for all states $s \in S$

$$\mathcal{A}_h \circ T_{f\to h}(s) = T_{f\to h} \circ \mathcal{A}_f(s) \ , \tag{8.8}$$

or equivalently

$$\mathcal{A}_h(s) = T_{f\to h} \circ \mathcal{A}_f \circ T^{-1}_{f\to h}(s) \ . \tag{8.9}$$

If $T_{f\to h}$ is the identity for all $h \in \mathcal{H}(f)$, the algorithm is *unconditionally invariant* under $\mathcal{H}$. For randomized algorithms, the equalities hold almost surely, given appropriately coupled random number realizations, otherwise in distribution. The set of functions $\mathcal{H}(f)$ is an invariance set of f for algorithm $\mathcal{A}$.

The simplest example where unconditional invariance trivially holds is $\mathcal{H} : f \mapsto \{f\}$. Any algorithm is unconditionally invariant under the "exchange" of f with f.

$$\begin{tikzcd}
s_k \arrow[r, "\mathcal{A}_f"] \arrow[d, shift right, "T_{f\to h}"'] & s_{k+1} \arrow[d, "T_{f\to h}"'] \\
T_{f\to h}(s_k) \arrow[r, "\mathcal{A}_h"] \arrow[u, shift right, "T^{-1}_{f\to h}"'] & T_{f\to h}(s_{k+1})
\end{tikzcd}$$

Fig. 8.3 Commutative diagram for invariance. *Vertical arrows* depict an invertible transformation (encoding) T of the state variables. *Horizontal arrows* depict one time step of algorithm $\mathcal{A}$, using the respective function and state variables. The two possible paths between a state at time k and a state at time $k+1$ are equivalent in all (four) cases. The two paths from *upper left* to *lower right* are reflected in Eq. (8.8). For $f = h$ the diagram becomes trivial with $T_{f\to h}$ as the identity. One interpretation of the diagram is that given $T^{-1}_{f\to h}$, any function h can be optimized like f

The idea of invariance is depicted in the commutative diagram in Fig. 8.3. The two possible paths from the upper left to the lower right are reflected in Eq. (8.8).

Equation (8.9) implies (trivially) for all $k \in \mathbb{N}$ that

$$\mathcal{A}_h^k(s) = T_{f\to h} \circ \mathcal{A}_f^k \circ T^{-1}_{f\to h}(s) \quad , \tag{8.10}$$

where $\mathcal{A}^k(s)$ denotes k iteration steps of the algorithm starting from s. Equation (8.10) reveals that for all $h \in \mathcal{H}(f)$, the algorithm $\mathcal{A}$ optimizes the function h with initial state s just like the function f with initial state $T^{-1}_{f\to h}(s)$. In the lucky scenario, $T_{f\to h}$ is the identity and $\mathcal{A}$ behaves identically on f and h. Otherwise, first s must be moved to $T^{-1}_{f\to h}(s)$, such that *after an adaptation phase* any function h is optimized just like the function f. This is particularly attractive if f is the easiest function in the invariance class. The adaptation time naturally depends on the distance between s and $T^{-1}_{f\to h}(s)$.

We give the first example of unconditional invariance to order-preserving transformations of f.

Proposition 8.1 (Invariance to order-preserving transformations). *For all strictly increasing functions $g : \mathbb{R} \to \mathbb{R}$ and for all $f : \mathbb{R}^n \to \mathbb{R}$, the CMA-ES behaves identically on the objective function $\boldsymbol{x} \mapsto f(\boldsymbol{x})$ and the objective function $\boldsymbol{x} \mapsto g(f(\boldsymbol{x}))$. In other words, CMA-ES is unconditionally invariant under*

$$\mathcal{H}_{\text{monotonic}} : f \mapsto \{g \circ f \mid g \textit{ is strictly increasing}\} \quad . \tag{8.11}$$

Additionally, for each $f : \mathbb{R}^n \to \mathbb{R}$, the set of functions $\mathcal{H}_{\text{monotonic}}(f)$—the orbit of f—is an equivalence class of functions with indistinguishable search trace.

Proof idea. Only the f-ranking of solutions is used in CMA-ES, and g does not change this ranking. We define the equivalence relation as $f \sim h$ iff $\exists g$ strictly increasing such that $f = g \circ h$. Then, reflexivity, symmetry and transitivity for the equivalence relation $\sim$ can be shown elementarily, recognizing that the identity and g^{-1} and compositions of strictly increasing functions are strictly increasing. □

The CMA-ES depends only on the sub-level sets $\{\boldsymbol{x} \mid f(\boldsymbol{x}) \leq \alpha\}$ for $\alpha \in \mathbb{R}$. The monotonous transformation g does not change the sub-level sets, that is $\{\boldsymbol{x} \mid g(f(\boldsymbol{x})) \leq g(\alpha)\} = \{\boldsymbol{x} \mid f(\boldsymbol{x}) \leq \alpha\}$.

8.5 Update of the Incumbent

Given the restricted usage of information from the evaluations of f, the incumbent is generally updated with a weighted mean of mutation steps:

$$\boldsymbol{m}_{k+1} = \boldsymbol{m}_k + c_\mathrm{m} \sum_{i=1}^{\mu} w_i \, (\boldsymbol{x}_{i:\lambda} - \boldsymbol{m}_k) \tag{8.12}$$

with

$$\sum_{i=1}^{\mu} |w_i| = 1, \quad w_1 \geq w_2 \cdots \geq w_\mu, \quad 0 < c_\mathrm{m} \leq 1 \; . \tag{8.13}$$

The question of how to choose optimal weight values w_i is pursued in [3], and the default values in Table 8.2 of the Appendix approximate the optimal positive values on the infinite dimensional sphere model. As discussed above, we add the constraints

$$w_\mu > 0 \quad \text{and} \quad \mu \leq \lambda/2 \; , \tag{8.14}$$

while the formulation with Eq. (8.12) also covers more general settings. Usually, we set the learning rate $c_\mathrm{m} = 1$ and the computation of the new incumbent simplifies to

$$\boldsymbol{m}_{k+1} = \sum_{i=1}^{\mu} w_i \, \boldsymbol{x}_{i:\lambda} \; . \tag{8.15}$$

A learning rate of one seems to be the largest sensible setting. A value larger than one should only be advantageous if σ_k is too small, and implies that the step-size heuristic should be improved. Very small σ_k together with $c_\mathrm{m} \gg 1$ resemble a classical gradient descent scenario.

The amount of utilized information can be quantified via the *variance effective selection mass*, or *effective* μ

$$\mu_\mathrm{eff} = \left(\sum_{i=1}^{\mu} w_i^2 \right)^{-1} \; , \tag{8.16}$$

where we can easily derive the tight bounds $1 < \mu_{\mathrm{eff}} \leq \mu$. Usually, a weight setting with $\mu_{\mathrm{eff}} \approx \lambda/4$ is appropriate. Given μ_{eff}, the specific choice of the weights is comparatively uncritical. The presented way to update the incumbent using a weighted mean of all μ selected points gives raise for the name$(\mu/\mu_{\mathrm{W}}, \lambda)$-CMA-ES.

Proposition 8.2 (Random ranking and stationarity of the incumbent). *Under (pure) random ranking, $\boldsymbol{m}_k$ follows an unbiased random walk*

$$\boldsymbol{m}_{k+1} \sim \boldsymbol{m}_k + \frac{\sigma_k}{\sqrt{\mu_{\mathrm{eff}}}} \mathcal{N}(\mathbf{0}, \boldsymbol{C}_k) \tag{8.17}$$

and consequently

$$E(\boldsymbol{m}_{k+1} | \boldsymbol{m}_k) = \boldsymbol{m}_k \ . \tag{8.18}$$

Pure random ranking means that the index values $i : \lambda \in \{1, \ldots, \lambda\}$ do not depend on $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_\lambda$, for all $i = 1, \ldots, \lambda$, for example, when $f(\boldsymbol{x})$ is a random variable with a density and does not depend on $\boldsymbol{x}$, or when $i : \lambda$ is set to i.

Proof idea. Equation (8.17) follows from Eqs. (8.5), (8.12), (8.16), and (8.18) follows because $E\mathcal{N}(\mathbf{0}, \boldsymbol{C}) = \mathbf{0}$ by definition. □

The proposition affirms that only selection (f-ranking) can induce a biased movement of the incumbent $\boldsymbol{m}$.

Proposition 8.3 (Maximum likelihood estimate of the mean). *Given $\boldsymbol{x}_{1:\lambda}, \ldots, \boldsymbol{x}_{\mu:\lambda}$, the incumbent $\boldsymbol{m}_{k+1}$ maximizes, independent of the positive definite matrix $\boldsymbol{C}$, the weighted likelihood*

$$\boldsymbol{m}_{k+1} = \underset{\boldsymbol{m} \in \mathbb{R}^n}{\arg\max} \prod_{i=1}^{\mu} p_{\mathcal{N}}^{w_i} (\boldsymbol{x}_{i:\lambda} \,|\, \boldsymbol{m}) \,, \tag{8.19}$$

where $p_{\mathcal{N}}^{w_i}(\boldsymbol{x} \,|\, \boldsymbol{m}) = (p_{\mathcal{N}}(\boldsymbol{x} \,|\, \boldsymbol{m}))^{w_i}$ and $p_{\mathcal{N}}(\boldsymbol{x} \,|\, \boldsymbol{m})$ denotes the density of $\mathcal{N}(\boldsymbol{m}, \boldsymbol{C})$ at point $\boldsymbol{x}$, or equivalently the weighted log-likelihood

$$\boldsymbol{m}_{k+1} = \underset{\boldsymbol{m} \in \mathbb{R}^n}{\arg\max} \sum_{i=1}^{\mu} w_i \times \log p_{\mathcal{N}} (\boldsymbol{x}_{i:\lambda} \,|\, \boldsymbol{m}) \,, \tag{8.20}$$

Proof idea. We exploit the one-dimensional normal density and the fact that the multivariate normal distribution, after a coordinate system rotation, can be decomposed into n independent marginal distributions. □

Finally, we find translation invariance, a property that every continuous search algorithm should enjoy.

Proposition 8.4 (Translation invariance). *The CMA-ES is translation invariant, that is, invariant under*

$$\mathcal{H}_{\text{trans}} : f \mapsto \{h_{\boldsymbol{a}} : \boldsymbol{x} \mapsto f(\boldsymbol{x} - \boldsymbol{a}) \mid \boldsymbol{a} \in \mathbb{R}^n\} \ , \tag{8.21}$$

with the bijective state transformation, $T_{f \to h_a}$, that maps $\boldsymbol{m}$ to $\boldsymbol{m} + \boldsymbol{a}$ (cf. Fig. 8.3). In other words, the trace of $\boldsymbol{m}_k + \boldsymbol{a}$ is the same for all functions $h_{\boldsymbol{a}} \in \mathcal{H}_{\text{trans}}$.

Proof idea. Consider Fig. 8.3: An iteration step with state $(\boldsymbol{m}_k, \sigma_k, \boldsymbol{C}_k, \dots)$ using cost function $\boldsymbol{x} \mapsto f(\boldsymbol{x})$ in the upper path is equivalent with an iteration step with state $(\boldsymbol{m}_k + \boldsymbol{a}, \sigma_k, \boldsymbol{C}_k, \dots)$ using cost function $h_{\boldsymbol{a}} : \boldsymbol{x} \mapsto f(\boldsymbol{x} - \boldsymbol{a})$ in the lower path. □

Translation invariance, meaning also that $\boldsymbol{m}_k - \boldsymbol{m}_0$ does not depend on $\boldsymbol{m}_0$, is a rather indispensable property for a search algorithm. Nevertheless, because $\boldsymbol{m}_k$ depends on $\boldsymbol{m}_0$, a reasonable proposition for $\boldsymbol{m}_0$, depending on f, is advisable.

8.6 Step-Size Control

Step-size control aims to make a search algorithm adaptive to the overall scale of search. Step-size control allows for fast convergence to an optimum and serves to satisfy the following basic demands on a search algorithm:

1. Solving linear functions, like $f(\boldsymbol{x}) = x_1$. On linear functions we desire a geometrical increase of the f-gain $f(\boldsymbol{m}_k) - f(\boldsymbol{m}_{k+1})$ with increasing k.
2. Solving the simplest convex-quadratic function, the sphere function

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} (x_i - x_i^*)^2 = \|\boldsymbol{x} - \boldsymbol{x}^*\|^2 \ , \tag{8.22}$$

quickly. We desire

$$\frac{\|\boldsymbol{m}_k - \boldsymbol{x}^*\|}{\|\boldsymbol{m}_0 - \boldsymbol{x}^*\|} \approx \exp\left(-c\frac{k}{n}\right) \ , \tag{8.23}$$

such that $c \not\ll 0.02 \min(n, \lambda)$, because $c \approx 0.25\,\lambda$ is the optimal value which can be achieved with optimal step-size, and optimal positive weights for $\lambda \not\gg n$ ($c \approx 0.5\,\lambda$ can be achieved using also negative weights for $\boldsymbol{x}_{i:\lambda} - \boldsymbol{m}_k$ in Eq. (8.12), see [3]). The optimal step-size changes when approaching the optimum.

Additionally, step-size control will provide scale invariance, as explicated below.

Unfortunately, step-size control can hardly be derived from first principles and therefore relies on some internal model or some heuristics. Line-search is one such heuristic that decides on the realized step length *after* the direction of the step is given. Surprisingly, a line-search can gain very little over a fixed (optimal) step length given in each iteration [27]. Recent theoretical results even seem to indicate that in the limit for $n \to \infty$ the optimal progress rate cannot be improved at all by a cost-free ray search on a half-line (given positive weights) or by a line search otherwise (Jebalia M, personal communication). A few further heuristics for step-size control are well-recognized:

1. Controlling the success rate of new candidate solutions, compared to the best solution seen so far (one-fifth success rule) [33, 35].
2. Sampling different candidate solutions with different step-sizes (self-adaptation) [33, 36]. Selected solutions also retain their step-size.
3. Testing different step-sizes by conducting additional test steps in direction $\boldsymbol{m}_{k+1} - \boldsymbol{m}_k$, resembling a rudimentary line-search (two-point adaptation) [18, 34].
4. Controlling the length of the search path, taken over a number of iterations (*cumulative step-size adaptation*, CSA, or *path-length control*) [32].

In our context, the last two approaches find reasonable values for σ in simple test cases (like ridge topologies).

We use cumulative step-size adaptation here. The underlying design principle is to achieve perpendicularity of successive steps. Perpendicularity is measured using an *evolution path* and a *variable metric*.

Conceptually, an **evolution path**, or search path, of length j is the vector

$$\boldsymbol{m}_k - \boldsymbol{m}_{k-j} \quad , \tag{8.24}$$

that is, the total displacement of the mean during j iterations. For technical convenience, and in order to satisfy the stationary condition Eq. (8.26), we compute the search path, $\boldsymbol{p}^\sigma$, in an iterative momentum equation with the initial path $\boldsymbol{p}_0^\sigma = \boldsymbol{0}$ as

$$\boldsymbol{p}_{k+1}^\sigma = (1 - c_\sigma)\boldsymbol{p}_k^\sigma + \sqrt{c_\sigma(2 - c_\sigma)\mu_{\text{eff}}}\, \boldsymbol{C}_k{}^{-\frac{1}{2}} \frac{\boldsymbol{m}_{k+1} - \boldsymbol{m}_k}{\sigma_k} \; . \tag{8.25}$$

The factor $1 - c_\sigma > 0$ is the decay weight, and $1/c_\sigma \approx n/3$ is the backward time horizon. After $1/c_\sigma$ iterations about $1 - \exp(-1) \approx 63\,\%$ of the information has been replaced; $\boldsymbol{C}_k{}^{-\frac{1}{2}}$ is the positive symmetric square root[4] of $\boldsymbol{C}_k{}^{-1}$. The remaining factors are, without further degree of freedom, chosen to guarantee the stationarity,

$$\boldsymbol{p}_k^\sigma \sim \mathcal{N}(\boldsymbol{0}, \mathbf{I}) \quad \text{for } k = 1, 2, 3, \ldots \; , \tag{8.26}$$

given $\boldsymbol{p}_0^\sigma \sim \mathcal{N}(\boldsymbol{0}, \mathbf{I})$ and pure random ranking of $\boldsymbol{x}_{i:\lambda}$ in all preceding time steps.

[4]The positive symmetric square root satisfies $\boldsymbol{C}_k{}^{-\frac{1}{2}}\boldsymbol{C}_k{}^{-\frac{1}{2}} = \boldsymbol{C}_k{}^{-1}$, has only positive eigenvalues and is unique.

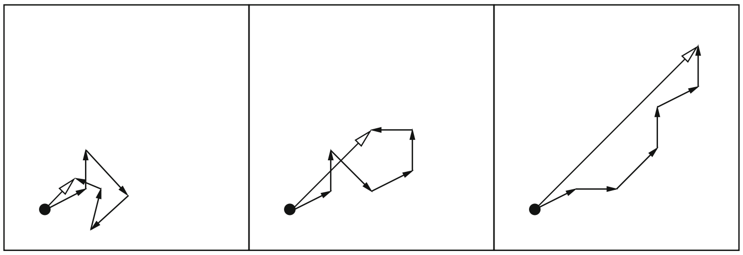


Fig. 8.4 Schematic depiction of three evolution paths in the search space (each with six successive steps of $\boldsymbol{m}_k$). *Left*: Single steps cancel each other out and the evolution path is short. *Middle*: Steps are "on average orthogonal". *Right*: Steps are positively correlated and the evolution path is long. The length of the path is a good indicator for optimality of the step-size

The length of the evolution path is used to update the step-size σ either following [29]

$$\sigma_{k+1} = \sigma_k \times \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|\boldsymbol{p}^\sigma_{k+1}\|^2 - n}{2n}\right)\right) \tag{8.27}$$

or via

$$\sigma_{k+1} = \sigma_k \times \exp\left(\frac{c_\sigma}{d_\sigma}\left(\frac{\|\boldsymbol{p}^\sigma_{k+1}\|}{\mathsf{E}\|\mathcal{N}(\mathbf{0},\mathbf{I})\|} - 1\right)\right) , \tag{8.28}$$

where $d_\sigma \approx 1$. The step-size increases/decreases iff $\|\boldsymbol{p}^\sigma_{k+1}\|^2$ or $\|\boldsymbol{p}^\sigma_{k+1}\|$ is larger/smaller than its expected value. Equation (8.27) is more appealing and easier to analyze, but Eq. (8.28) might have an advantage in practice. In practice, also an upper bound to the argument of exp is sometimes useful.

Figure 8.4 depicts the idea of the step-size control schematically.

- If steps are positively correlated, the evolution path tends to be long (*right picture*). A similar trajectory could be covered by fewer but longer steps and the step-size is increased.
- If steps are negatively correlated they tend to cancel each other out and the evolution path is short (*left picture*). Shorter steps seem more appropriate and the step-size is decreased.
- If the f-ranking does not affect the length of the evolution path, the step-size is unbiased (*middle picture*).

We note two major postulates related to step-size control and two major design principles of the step-size update.

Postulate 1 (Conjugate steps). *Successive iteration steps should be approximately $\boldsymbol{C}^{-1}$-conjugate, that is, orthogonal with respect to the inner product (and metric) defined by $\boldsymbol{C}^{-1}$.*

As a consequence of this postulate, we have used perpendicularity as optimality criterion for step-size control.

If steps are uncorrelated, like under random selection, they indeed become approximately $\boldsymbol{C}^{-1}$-conjugate, that is, $(\boldsymbol{m}_{k+1} - \boldsymbol{m}_k)^{\mathrm{T}} \boldsymbol{C}^{-1} \boldsymbol{m}_k - \boldsymbol{m}_{k-1} \approx 0$, see [15]. This means the steps are orthogonal with respect to the inner product defined by $\boldsymbol{C}^{-1}$ and therefore orthogonal in the coordinate system defined by $\boldsymbol{C}$. In this coordinate system, the coordinate axes, where the independent sampling takes place, are eigenvectors of $\boldsymbol{C}$. Seemingly uncorrelated steps are the desired case and are achieved by using $\boldsymbol{C}^{-1/2}$ in Eq. (8.25).

In order to better understand the following assertions, we rewrite the step-size update in Eq. (8.28), only using an *additive* update term,

$$\log \sigma_{k+1} = \log \sigma_k + \frac{c_\sigma}{d_\sigma} \left(\frac{\|\boldsymbol{p}^\sigma_{k+1}\|}{\mathsf{E}\|\mathcal{N}(\mathbf{0}, \mathbf{I})\|} - 1 \right) . \tag{8.29}$$

First, in accordance with our stationary design principle, we establish a stationarity condition on the step-size.

Proposition 8.5 (Stationarity of step-size). *Given pure random ranking and* $\boldsymbol{p}^\sigma_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$*, the quantity* $\log \sigma_k$ *performs an unbiased random walk (see Eq.* (8.29)*). Consequently, the step-size obeys the stationarity condition*

$$E(\log \sigma_{k+1} | \sigma_k) = \log \sigma_k \ . \tag{8.30}$$

Proof idea. We analyze the update Eqs. (8.29) and (8.25). □

Postulate 2 (Behavior on linear functions [14]). *On a linear function, the dispersion of new candidate solutions should increase geometrically fast in the iteration sequence, that is, linearly on the log scale. Given* σ_k^β *as dispersion measure with* $\beta > 0$*, we can set w.l.o.g.* $\beta = 1$ *and demand for some* $\alpha > 0$

$$E(\log \sigma_{k+1} | \sigma_k) \geq \log \sigma_k + \alpha \ . \tag{8.31}$$

The CMA-ES satisfies the postulate for some k_0 and all $k \geq k_0$, because on a linear function the expected length of the evolution path increases monotonically. We reckon that $k_0 \propto 1/c_\sigma$. Finally, we investigate the more abstract conception of scale invariance as depicted in Fig. 8.5.

Proposition 8.6 (Scale invariance). *The CMA-ES is invariant under*

$$\mathcal{H}_{\mathrm{scale}} : f \mapsto \{h_\alpha : \boldsymbol{x} \mapsto f(\boldsymbol{x}/\alpha) \mid \alpha > 0\} \tag{8.32}$$

with the associated bijective state space transformation

$$T : (\boldsymbol{m}, \sigma, \boldsymbol{C}, \boldsymbol{p}^\sigma, \boldsymbol{p}^{\mathrm{c}}) \mapsto (\alpha \boldsymbol{m}, \alpha\sigma, \boldsymbol{C}, \boldsymbol{p}^\sigma, \boldsymbol{p}^{\mathrm{c}}) \ .$$

$$\begin{tikzcd}
(\boldsymbol{m}_k, \sigma_k, \boldsymbol{C}_k, \dots) \arrow[r, "k \to k+1 \text{ using } f(\boldsymbol{x})"] \arrow[d, shift right, "T(\alpha)"'] & (\boldsymbol{m}_{k+1}, \sigma_{k+1}, \boldsymbol{C}_{k+1}, \dots) \arrow[d, shift left, "T(\alpha)"] \\
(\alpha\boldsymbol{m}_k, \alpha\sigma_k, \boldsymbol{C}_k, \dots) \arrow[r, "k \to k+1 \text{ using } f(\boldsymbol{x}/\alpha)"] \arrow[u, shift right, "T^{-1}(\alpha)"'] & (\alpha\boldsymbol{m}_{k+1}, \alpha\sigma_{k+1}, \boldsymbol{C}_{k+1}, \dots) \arrow[u, shift left, "T^{-1}(\alpha)"]
\end{tikzcd}$$

Fig. 8.5 Commutative diagram for scale invariance. *Vertical arrows* depict an invertible transformation (encoding) T of all state variables of CMA-ES with $T(\alpha) : (\boldsymbol{m}, \sigma, \boldsymbol{C}, \boldsymbol{p}^\sigma, \boldsymbol{p}^{\mathrm{c}}) \mapsto (\alpha\boldsymbol{m}, \alpha\sigma, \boldsymbol{C}, \boldsymbol{p}^\sigma, \boldsymbol{p}^{\mathrm{c}})$. *Horizontal arrows* depict one time step of CMA-ES, applied to the respective tuple of state variables. The two possible paths between a state at time k and a state at time $k+1$ are equivalent in all (four) cases. For $\alpha = 1$ the diagram becomes trivial. The diagram suggests that CMA-ES is invariant under the choice of $\alpha > 0$ in the sense that, given T and T^{-1} were available, any function $\boldsymbol{x} \mapsto f(\alpha\boldsymbol{x})$ is (at least) as easy to optimization as f

That means for all states $(\boldsymbol{m}_k, \sigma_k, \boldsymbol{C}_k, \boldsymbol{p}_k^\sigma, \boldsymbol{p}_k^{\mathrm{c}})$

$$\textit{CMA-ES}_h(T(\boldsymbol{m}_k, \sigma_k, \boldsymbol{C}_k, \boldsymbol{p}_k^\sigma, \boldsymbol{p}_k^{\mathrm{c}})) = T(\textit{CMA-ES}_f(\underbrace{\boldsymbol{m}_k, \sigma_k, \boldsymbol{C}_k, \boldsymbol{p}_k^\sigma, \boldsymbol{p}_k^{\mathrm{c}}}_{= \atop T^{-1}(T(\boldsymbol{m}_k, \sigma_k, \boldsymbol{C}_k, \boldsymbol{p}_k^\sigma, \boldsymbol{p}_k^{\mathrm{c}}))})) \ , \tag{8.33}$$

see Fig. 8.5. Furthermore, for any given $f : \mathbb{R}^n \to \mathbb{R}$, *the set of functions* $\mathcal{H}_{\text{scale}}(f)$*—the orbit of* f*—is an equivalence class.*

Proof idea. We investigate the update equations of the state variables comparing the two possible paths from the lower left to the lower right in Fig. 8.5. The equivalence relation property can be shown elementarily (cf. Proposition 8.1) or using the property that the set $\{\alpha > 0\}$ is a transformation group over the set $\{h : \mathbb{R}^n \to \mathbb{R}\}$ and therefore induces the equivalence classes $\mathcal{H}_{\text{scale}}(f)$ (see also Proposition 8.9). □

Invariance allows us to draw the commutative diagram of Fig. 8.5. Scale invariance can be interpreted in several ways:

- The choice of scale α is irrelevant for the algorithm, that is, the algorithm has no intrinsic (built-in) notion of scale.
- The transformation T in Fig. 8.5 is a change of coordinate system (here: change of scale) and the update equations are independent of the actually chosen coordinate system; that is, they could be formulated in an algebraic way.
- For functions in the equivalence class $\mathcal{H}_{\text{scale}}(f)$, the trace of the algorithm $(\alpha\boldsymbol{m}_k, \alpha\sigma_k, \boldsymbol{C}_k, \boldsymbol{p}_k^\sigma, \boldsymbol{p}_k^{\mathrm{c}})$ will be identical for all $k = 0, 1, 2, \dots$, given that $\boldsymbol{m}_0$ and σ_0 are chosen appropriately, for example, $\sigma_0 = 1/\alpha$ and $\boldsymbol{m}_0 = \sigma_0 \times \boldsymbol{a}$. Then the trace for $k = 0$ equals $(\alpha\boldsymbol{m}_0, \alpha\sigma_0, \boldsymbol{C}_0, \dots) = (\boldsymbol{a}, 1, \boldsymbol{C}_0, \dots)$, and the trace does not depend on α for any $k \geq 0$.

- From the last point follows that the step-size control has a distinct role in scale invariance. In practice, when α is unknown, adaptation of the step-size that achieves $\sigma_k \propto 1/\alpha$ can render the algorithm virtually independent of α.

Scale invariance and step-size control also facilitate the possibility of **linear convergence** in k to the optimum $\boldsymbol{x}^*$, in that

$$\lim_{k\to\infty} \sqrt[k]{\frac{\|\boldsymbol{m}_k - \boldsymbol{x}^*\|}{\|\boldsymbol{m}_0 - \boldsymbol{x}^*\|}} = \exp\left(-\frac{c}{n}\right) \tag{8.34}$$

exists with $c > 0$ or equivalently,

$$\begin{aligned}\lim_{k\to\infty} \frac{1}{k} \log \|\boldsymbol{m}_k - \boldsymbol{x}^*\| &= \lim_{k\to\infty} \frac{1}{k} \log \frac{\|\boldsymbol{m}_k - \boldsymbol{x}^*\|}{\|\boldsymbol{m}_0 - \boldsymbol{x}^*\|} \\ &= \lim_{k\to\infty} \frac{1}{k} \sum_{k=1}^{t} \log \frac{\|\boldsymbol{m}_k - \boldsymbol{x}^*\|}{\|\boldsymbol{m}_{k-1} - \boldsymbol{x}^*\|} \\ &= -\frac{c}{n} \end{aligned} \tag{8.35}$$

and similarly

$$E\left(\log \frac{\|\boldsymbol{m}_{k+1} - \boldsymbol{x}^*\|}{\|\boldsymbol{m}_k - \boldsymbol{x}^*\|}\right) \to -\frac{c}{n} \quad \text{for } k \to \infty\ . \tag{8.36}$$

Hence, c denotes a convergence rate and for $c > 0$ the algorithm converges "log-linearly" (in other words, geometrically fast) to the optimum.

In the beginning of this section we stated two basic demands on a search algorithm that step-size control is meant to address, namely solving linear functions and the sphere function appropriately fast. We now pursue, with a single experiment, whether the demands are satisfied.

Figure 8.6 shows a run on the objective function $f : \mathbb{R}^n \to \mathbb{R}, \boldsymbol{x} \mapsto \|\boldsymbol{x}\|$, with $n = 20$, $\lambda = 12$ (the default value, see Table 8.2) and with $\sigma_0 = 10^{-9}$ chosen far too small given that $\boldsymbol{m}_0 = \mathbf{1}$. The outcome when repeating this experiment always looks very similar. We discuss the demands in turn.

1. During the first 170 iterations the algorithm virtually "observes" the linear function $\boldsymbol{x} \mapsto \sum_{i=1}^{20} x_i$ at point $\mathbf{1} \in \mathbb{R}^{20}$. We see during this phase that σ increases geometrically fast (linearly on the log scale). From this observation, and the invariance properties of the algorithm (also rotation invariance, see below), we can safely imply that the demand for linear functions is satisfied.
2. After the adaptation of σ after about 180 iterations, linear convergence to the optimum can be observed. We compute the convergence rate between iteration

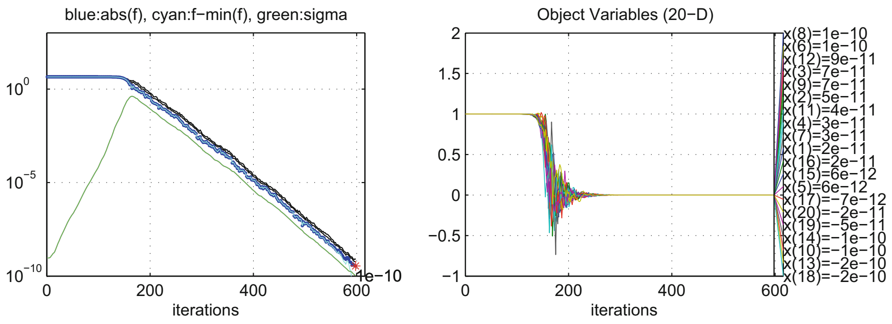


Fig. 8.6 A run of CSA-ES (Eqs. (8.5), (8.15), (8.25) and (8.28)) on the objective function $f : \mathbb{R}^{20} \to \mathbb{R}, \boldsymbol{x} \mapsto \|\boldsymbol{x}\|$, as a member of the equivalence class of functions $\boldsymbol{x} \mapsto g(\|\alpha\,\boldsymbol{x} - \boldsymbol{x}^*\|)$ with identical behavior, given $\sigma_0 \propto 1/\alpha$ and $\boldsymbol{m}_0 = \sigma_0 \times (\text{const} + \boldsymbol{x}^*)$. Here, $\boldsymbol{m}_0 = \mathbf{1}$ and the initial step-size $\sigma_0 = 10^{-9}$ is chosen far too small. *Left*: $f(\boldsymbol{m}_k)$ (*thick blue graph*) and σ_k versus iteration number k in a semi-log plot. *Right*: All components of $\boldsymbol{m}_k$ versus k

180 and 600 from the graph. Starting with $\frac{\|\boldsymbol{m}_k\|}{\|\boldsymbol{m}_0\|} \approx \exp\left(-c\frac{k}{n}\right)$ from Eq. (8.23) we replace $\boldsymbol{m}_0$ with $\boldsymbol{m}_{180}$ and compute

$$\frac{\|\boldsymbol{m}_{k=600}\|}{\|\boldsymbol{m}_{k=180}\|} \approx \frac{10^{-9.5}}{10^0} \approx \exp\left(-c\,\frac{600-180}{20}\right) . \tag{8.37}$$

Solving for c yields $c \approx 1.0$ and with $\min(n, \lambda) = \lambda = 12$ we get $c \approx 1.0 \not\ll 0.24 = 0.02 \min(n, \lambda)$. Our demand on the convergence rate c is more than satisfied. The same can be observed when covariance matrix adaptation is applied additionally (not shown).

The demand on the convergence (8.23) can be rewritten in that

$$\log \|\boldsymbol{m}_k - \boldsymbol{x}^*\| \approx -c\frac{k}{n} + \text{const} . \tag{8.38}$$

The k in the RHS numerator implies *linear convergence* in the number of iterations. The n in the denominator implies **linear scale-up**: The number of iterations to reduce the distance to the optimum by a given factor increases linearly with the dimension n. Linear *convergence* can also be achieved with covariance matrix adaptation. Given $\lambda \not\gg n$, linear *scale-up* cannot be achieved with covariance matrix adaptation alone, because a reliable setting for the learning rate for the covariance matrix is $o(1/n)$. However, step-size control is reliable and achieves linear scale-up given the step-size damping parameter $d_\sigma = O(1)$ in Eq. (8.28). Scale-up experiments are inevitable to support this claim and have been done, for example, in [22].

8.7 Covariance Matrix Adaptation

In the remainder we exploit the f-ranked (i.e., selected and ordered) set $(\boldsymbol{x}_{1:\lambda}, \ldots, \boldsymbol{x}_{\mu:\lambda})$ to update the covariance matrix $\boldsymbol{C}$. First, we note that the covariance matrix represents *variation* parameters. Consequently, an apparent principle is to encourage, or *reinforce variations that have been successful*—just like successful candidate solutions are reinforced in the update of $\boldsymbol{m}$ in Eq. (8.15). Based on the current set of f-ranked points, the successful variations are (by definition)

$$\boldsymbol{x}_{i:\lambda} - \boldsymbol{m}_k \quad \text{for } i = 1, \ldots, \mu \, . \tag{8.39}$$

Remark that "successful variation" does not imply $f(\boldsymbol{x}_{i:\lambda}) < f(\boldsymbol{m}_k)$, which is neither necessary nor important nor even desirable in general. Even the demand $f(\boldsymbol{x}_{1:\lambda}) < f(\boldsymbol{m}_k)$ would often result in a far too small a step-size.

8.7.1 The Rank-μ Matrix

From the successful variations in (8.39) we form a covariance matrix

$$\boldsymbol{C}^{\mu}_{k+1} = \sum_{i=1}^{\mu} w_i \, \frac{\boldsymbol{x}_{i:\lambda} - \boldsymbol{m}_k}{\sigma_k} \times \frac{\left(\boldsymbol{x}_{i:\lambda} - \boldsymbol{m}_k\right)^{\mathrm{T}}}{\sigma_k} \, . \tag{8.40}$$

Equation (8.40) is analogous to Eq. (8.15) where successful solution points are used to form the new incumbent. We can easily derive the condition

$$E(\boldsymbol{C}^{\mu}_{k+1} | \boldsymbol{C}_k) = \boldsymbol{C}_k \tag{8.41}$$

under pure random ranking, thus explaining the factors $1/\sigma_k$ in (8.40).

Assuming the weights w_i as given, the matrix $\boldsymbol{C}^{\mu}_{k+1}$ maximizes the (weighted) likelihood of the f-ranked steps.

Proposition 8.7 (Maximum likelihood estimate of $\boldsymbol{C}$). *Given $\mu \geq n$, the matrix $\boldsymbol{C}^{\mu}_{k+1}$ maximizes the weighted log-likelihood*

$$\boldsymbol{C}^{\mu}_{k+1} = \underset{\boldsymbol{C} \text{ pos def}}{\arg\max} \sum_{i=1}^{\mu} w_i \times \log p_{\mathcal{N}} \left(\frac{\boldsymbol{x}_{i:\lambda} - \boldsymbol{m}_k}{\sigma_k} \,\middle|\, \boldsymbol{C} \right) , \tag{8.42}$$

where $p_{\mathcal{N}}(\boldsymbol{x} \,|\, \boldsymbol{C})$ denotes the density of $\mathcal{N}(\boldsymbol{0}, \boldsymbol{C})$ at point $\boldsymbol{x}$, and therefore the RHS of Eq. (8.42) reads more explicitly

$$\underset{\boldsymbol{C} \text{ pos def}}{\arg\max} \left(-\frac{1}{2} \log \det(\alpha \boldsymbol{C}) - \frac{1}{2{\sigma_k}^2} \sum_{i=1}^{\mu} w_i (\boldsymbol{x}_{i:\lambda} - \boldsymbol{m}_k)^{\mathrm{T}} \boldsymbol{C}^{-1} (\boldsymbol{x}_{i:\lambda} - \boldsymbol{m}_k) \right) \tag{8.43}$$

where $\alpha = 2\pi{\sigma_k}^2$ is irrelevant for the result.

Proof idea. The proof is nontrivial but works similarly to the classical non-weighted case. □

In contrast to the computation of $\boldsymbol{m}$ in Eq. (8.12), we are not aware of a derivation for optimality of certain weight values in Eq. (8.40). Future results might reveal that different weights and/or even a different value for μ are desirable for Eqs. (8.12) and (8.40). Before we turn finally to the covariance matrix update, we scrutinize the computation of $\boldsymbol{C}^{\mu}_{k+1}$.

8.7.1.1 What Is Missing?

In Sect. 8.3 we argued to use only the μ best solutions from the last iteration to update distribution parameters. For a covariance matrix update, disregarding the worst solutions might be too conservative, and a negative update of the covariance matrix with the μ worst solutions is proposed in [29]. This idea is not accommodated in this chapter, but has been recently exploited with consistently good results [4,23]. An inherent inconsistency with negative updates though is that long steps tend to be worse merely because they are long (and not because they represent a bad direction) meanwhile, unfortunately, long steps also lead to stronger updates.

At first sight we might believe to have covered all variation information given by $\boldsymbol{x}_{i:\lambda} - \boldsymbol{m}_k$ in the covariance matrix $\boldsymbol{C}^{\mu}_{k+1}$. On closer inspection we find that the outer product in Eq. (8.40) removes the sign: Using $-(\boldsymbol{x}_{i:\lambda} - \boldsymbol{m})$ instead of $\boldsymbol{x}_{i:\lambda} - \boldsymbol{m}$ in Eq. (8.40) yields the same $\boldsymbol{C}^{\mu}_{k+1}$. One possibility to recover the sign information is to favor the direction $\boldsymbol{x}_{i:\lambda} - \boldsymbol{m}$ over $-(\boldsymbol{x}_{i:\lambda} - \boldsymbol{m}) = \boldsymbol{m}_k - \boldsymbol{x}_{i:\lambda}$ in some way. This seems difficult to accomplish without affecting either the distribution mean (interfering with Proposition 8.3) or the maximum entropy property. Therefore, we choose a different way to recover the sign information.

8.7.2 *Another Evolution Path*

We recover the sign information in a classical and rather heuristic way, which turns out to be nevertheless quite effective. We consider an evolution path $\boldsymbol{x} - \boldsymbol{m}_{k-j}$ for $j > 0$, where $\boldsymbol{x}$ might be $\boldsymbol{m}_{k+1}$ or any $\boldsymbol{x}_{i:\lambda}$. We decompose the path into the recent step and the old path

$$\boldsymbol{x} - \boldsymbol{m}_{k-j} = \boldsymbol{x} - \boldsymbol{m}_k + \boldsymbol{m}_k - \boldsymbol{m}_{k-j} \ . \tag{8.44}$$

Switching the sign of the last step means using the vector $\boldsymbol{m}_k - \boldsymbol{x}$ instead of $\boldsymbol{x} - \boldsymbol{m}_k$, and we get in this case

$$\begin{aligned} \boldsymbol{m}_k - \boldsymbol{x} + \boldsymbol{m}_k - \boldsymbol{m}_{k-j} &= 2(\boldsymbol{m}_k - \boldsymbol{x}) + \boldsymbol{x} - \boldsymbol{m}_{k-j} \\ &= \boldsymbol{x} - \boldsymbol{m}_{k-j} - 2(\boldsymbol{x} - \boldsymbol{m}_k) \ . \end{aligned} \tag{8.45}$$

Comparing the last line with the LHS of Eq. (8.44), we see that now the sign of the recent step matters. Only in the trivial cases, if either $\boldsymbol{x} = \boldsymbol{m}_k$ (zero step) or $\boldsymbol{m}_k = \boldsymbol{m}_{k-j}$ (previous zero path) the outer products of Eqs. (8.44) and (8.45) are identical. Because we will compute the evolution path over a considerable number of iterations j, the specific choice for $\boldsymbol{x}$ should become rather irrelevant and we will use $\boldsymbol{m}_{k+1}$ in the following.

In practice, we compute the evolution path, analogous to Eq. (8.25). We set $\boldsymbol{p}_0^{\mathrm{c}} = \boldsymbol{0}$ and use the momentum equation

$$\boldsymbol{p}_{k+1}^{\mathrm{c}} = (1 - c_{\mathrm{c}})\boldsymbol{p}_k^{\mathrm{c}} + h_\sigma \sqrt{c_{\mathrm{c}}(2 - c_{\mathrm{c}})\mu_{\mathrm{eff}}}\, \frac{\boldsymbol{m}_{k+1} - \boldsymbol{m}_k}{\sigma_k} \,, \tag{8.46}$$

where $h_\sigma = 1$ if $\|\boldsymbol{p}_{k+1}^\sigma\|^2 < \left(1 - (1 - c_\sigma)^{2(k+1)}\right)\left(2 + \frac{2}{n+1}\right) n$ and zero otherwise; h_σ stalls the update whenever $\|\boldsymbol{p}_{k+1}^\sigma\|$ is large. The implementation of h_σ supports the judgment of pursuing a heuristic rather than a first principle here, and is driven by two considerations.

1. Given a fast increase of the step-size (induced by the fact that $\|\boldsymbol{p}_{k+1}^\sigma\|$ is large), the change to the "visible" landscape will be fast, and the adaptation of the covariance matrix to the current landscape seems inappropriate, in particular, because
2. The covariance matrix update using $\boldsymbol{p}^{\mathrm{c}}$ is asymmetric: A large variance in a single direction can be *introduced* fast (while $\|\boldsymbol{p}_{k+1}^{\mathrm{c}}\|$ is large), but the large variance can only be *removed* on a significantly longer time scale. For this reason in particular, an unjustified update should be avoided.

While in Eq. (8.46), again, $1 - c_{\mathrm{c}}$ is the decay factor and $1/c_{\mathrm{c}} \approx (n + 4)/4$, the remaining constants are determined by the stationarity condition

$$\boldsymbol{p}_{k+1}^{\mathrm{c}} \sim \boldsymbol{p}_k^{\mathrm{c}} \,, \tag{8.47}$$

given $\boldsymbol{p}_k^{\mathrm{c}} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{C}_k)$ and pure random ranking and $h_\sigma \equiv 1$.

The evolution path $\boldsymbol{p}^{\mathrm{c}}$ heavily exploits the sign information. Let us consider, for a given $\boldsymbol{y} \in \mathbb{R}^n$, two hypothetical situations with $\boldsymbol{m}_{k+1} - \boldsymbol{m}_k = \alpha^k \boldsymbol{y}$, for $k = 0, 1, 2, \ldots$. We find that for $k \to \infty$

$$\boldsymbol{p}_k^{\mathrm{c}} \to \sqrt{\frac{2 - c_{\mathrm{c}}}{c_{\mathrm{c}}}}\boldsymbol{y} \approx \sqrt{\frac{n+2}{2}}\,\boldsymbol{y} \;\text{ if } \alpha^k = 1 \tag{8.48}$$

$$\boldsymbol{p}_k^{\mathrm{c}} \to (-1)^{k-1}\sqrt{\frac{c_{\mathrm{c}}}{2 - c_{\mathrm{c}}}}\boldsymbol{y} \approx (-1)^{k-1}\sqrt{\frac{2}{n+2}}\,\boldsymbol{y} \;\text{ if } \alpha^k = (-1)^k \tag{8.49}$$

Both equations follow from solving the stationarity condition $x = (1-c_\mathrm{c}) \times (\pm x) + \sqrt{c_\mathrm{c}(2-c_\mathrm{c})}$ for x. Combining both equations, we get the ratio between maximal and minimal possible length of $\boldsymbol{p}^\mathrm{c}$, given the input vectors have constant length, as

$$\frac{2-c_\mathrm{c}}{c_\mathrm{c}} \approx \frac{n+2}{2} \,. \tag{8.50}$$

Additionally to the matrix $\boldsymbol{C}^{\mu}_{k+1}$, we use the rank-one matrix $\boldsymbol{p}^\mathrm{c}_{k+1}\boldsymbol{p}^{\mathrm{c}\,\mathrm{T}}_{k+1}$ to introduce the missing sign information into the covariance matrix. The update is specified below in Eq. (8.51). The update implements the principal heuristic of reinforcing successful variations for variations observed over several iterations.

Evaluation of the Cumulation Heuristic

We evaluate the effect of the evolution path for covariance matrix adaptation. Figure 8.7 shows running length measurements of the $(\mu/\mu_\mathrm{W}, \lambda)$-CMA-ES depending on the choice of c_c on the cigar function (see legend). The graphs in the left plot are typical example data to identify a good parameter setting. Ten values for c_c^{-1} between 1 and $10\,n$ are shown for each dimension. Larger values are not regarded as sensible. The setting $c_\mathrm{c} = 1$ means that the heuristic is switched off. Improvements over the setting $c_\mathrm{c} = 1$ can be observed in particular for larger dimensions, where, up to $n = 100$, the function can be solved up to ten times faster. For $c_\mathrm{c}^{-1} = n$ the performance is for all dimensions close to optimal.

The right plot shows the running lengths for four different parameter settings versus dimension. For $n = 3$ the smallest speed-up of about 25 % is observed for all variants with $c_\mathrm{c}^{-1} > 1$. The speed-up grows to a factor of roughly 2, 4, and 10 for dimensions 10, 30, and 100, respectively, and always exceeds a factor of $\sqrt{n}/2$. For $c_\mathrm{c} = 1$ (heuristic off) the scaling with the dimension is $\approx n^{1.7}$. For $c_\mathrm{c}^{-1} = \sqrt{n}$ the scaling becomes $\approx n^{1.1}$ and about linear for $c_\mathrm{c}^{-1} \geq n/3$. These findings hold for any function, where the predominant task is to acquire the orientation of a constant number of "long axes", in other words to find a few insensitive directions, where yet a large distance needs to be traversed. The assertion in [37] that $c_\mathrm{c}^{-1} \propto n$ is needed to get a significant scaling improvement turns out to be wrong. For larger population sizes λ, where the rank-μ update becomes more effective, the positive effect reduces and almost vanishes with $\lambda = 10n$.

The same experiment has been conducted on other (unimodal) functions. While on many functions the cumulation heuristic is less effective and yields only a rather n-independent and small speed-up (e.g., on the Rosenbrock function somewhat below a factor of two), we have not seen an example yet where it compromises the performance remarkably. Hence the default choice has become $c_\mathrm{c}^{-1} \approx n/4$ (see Table 8.2 in the Appendix), because (a) the update for the covariance matrix will have a time constant of $c_1^{-1} \approx n^2/2$ and we feel that $c_1^{-1}/c_\mathrm{c}^{-1}$ should not be smaller than n, and (b) in our additional experiments the value $c_\mathrm{c}^{-1} = n$ is indeed sometimes worse than smaller values.

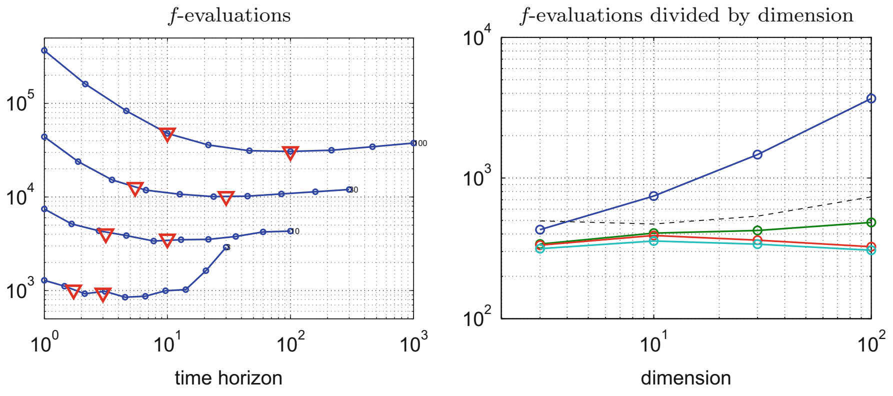


Fig. 8.7 Number of function evaluations to reach $f(\boldsymbol{x}) < 10^{-6}$ on $f(\boldsymbol{x}) = x_1^2 + 10^6 \sum_{i=2}^{n} x_i^2$ with $\boldsymbol{m}_0 = \boldsymbol{1}$ and $\sigma_0 = 1$. For a (backward) time horizon of $c_c^{-1} = 1$, the cumulation heuristic is, by definition, switched off. *Left figure*: Number of function evaluations, where each point represents a single run, plotted versus the backward time horizon of the evolution path, c_c^{-1}, for $n = [3; 10; 30; 100]$ (from bottom to top). *Triangles* show averages for $c_c^{-1} = \sqrt{n}$ and n, also shown on the right. *Right figure*: Average number of function evaluations divided by n, from $[10; 3; 2; 1] = \lfloor 10/\lfloor\sqrt{n}\rfloor\rfloor$ runs, plotted versus n for (from top to bottom) $c_c^{-1} = 1$; $\sqrt{n}$; $\frac{n+3}{3}$; n. Compared to $c_c = 1$, the speed-up exceeds in all cases a factor of $\sqrt{n}/2$ (*dashed line*)

8.7.3 The Covariance Matrix Update

The final covariance matrix update combines a rank-one update using $\boldsymbol{p}^c\boldsymbol{p}^{c\mathrm{T}}$ and a rank-μ update using $\boldsymbol{C}_{k+1}^{\mu}$,

$$\boldsymbol{C}_{k+1} = (1 - c_1 - c_\mu + c_\epsilon)\,\boldsymbol{C}_k + c_1 \boldsymbol{p}_{k+1}^{c} \boldsymbol{p}_{k+1}^{c\ \mathrm{T}} + c_\mu \boldsymbol{C}_{k+1}^{\mu}\ , \tag{8.51}$$

where $\boldsymbol{p}^c$ and $\boldsymbol{C}_{k+1}^{\mu}$ are defined in Eqs. (8.46) and (8.40), respectively, and $c_\epsilon = (1 - h_\sigma^2)\, c_1 c_c (2 - c_c)$ is of minor relevance and makes up for the loss of variance in case of $h_\sigma = 0$. The constants $c_1 \approx 2/n^2$ and $c_\mu \approx \mu_{\mathrm{eff}}/n^2$ for $\mu_{\mathrm{eff}} < n^2$ are learning rates satisfying $c_1 + c_\mu \leq 1$. The approximate values reflect the rank of the input matrix or the number of input samples, divided by the degrees of freedom of the covariance matrix. The remaining degrees of freedom are covered by the old covariance matrix $\boldsymbol{C}_k$. Again, the equation is governed by a stationarity condition.

Proposition 8.8 (Stationarity of covariance matrix $\boldsymbol{C}$). *Given pure random ranking and $\boldsymbol{p}_k^c \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{C}_k)$ and $h_\sigma = 1$, we have*

$$E(\boldsymbol{C}_{k+1}|\boldsymbol{C}_k) = \boldsymbol{C}_k\ . \tag{8.52}$$

Proof idea. Compute the expected value of Eq. (8.51). □

Finally, we can state general linear invariance for CMA-ES, analogous to scale invariance in Proposition 8.6 and Fig. 8.5.

Proposition 8.9 (Invariance under general linear transformations). *The CMA-ES is invariant under full rank linear transformations of the search space, that is, for each $f : \mathbb{R}^n \to \mathbb{R}$ invariant under*

$$\mathcal{H}_{\mathrm{GL}} : f \mapsto \{ f \circ \boldsymbol{B}^{-1} : \boldsymbol{x} \mapsto f(\boldsymbol{B}^{-1}\boldsymbol{x}) \mid \boldsymbol{B} \textit{ is a full rank } n \times n \textit{ matrix} \} \, . \qquad (8.53)$$

The respective bijective state space transformation reads

$$T_{\boldsymbol{B}} : (\boldsymbol{m}, \sigma, \boldsymbol{C}, \boldsymbol{p}^{\sigma}, \boldsymbol{p}^{\mathrm{c}}) \mapsto (\boldsymbol{Bm}, \sigma, \boldsymbol{BCB}^{\mathrm{T}}, \boldsymbol{p}^{\sigma}, \boldsymbol{Bp}^{\mathrm{c}}) \, . \qquad (8.54)$$

Furthermore, for each f, the set $\mathcal{H}_{\mathrm{GL}}(f)$ is an equivalence class with identical algorithm trace $T_{\boldsymbol{B}}(\boldsymbol{m}_k, \sigma_k, \boldsymbol{C}_k, \boldsymbol{p}^{\sigma}_k, \boldsymbol{p}^{\mathrm{c}}_k)$ for a state s and the initial state $(\boldsymbol{m}_0, \sigma_0, \boldsymbol{C}_0, \boldsymbol{p}^{\sigma}_0, \boldsymbol{p}^{\mathrm{c}}_0) = T_{\boldsymbol{B}}^{-1}(s)$.

Proof idea. Straightforward computation of the updated tuple: The equivalence relation property can be shown elementarily (cf. Proposition 8.1) or by recognizing that the set of full rank matrices is a transformation group over the set $\{f : \mathbb{R}^n \to \mathbb{R}\}$ with group action $(\boldsymbol{B}, f) \mapsto f \circ \boldsymbol{B}^{-1}$ and therefore induces the equivalence classes $\mathcal{H}_{\mathrm{GL}}(f)$ as orbits of f under the group action. □

A commutative diagram, analogous to Fig. 8.5, applies with $T_{\boldsymbol{B}}$ in place of $T(\alpha)$ and using $f(\boldsymbol{B}^{-1}\boldsymbol{x})$ in the lower path. The transformation $\boldsymbol{B}$ can be interpreted as a change of basis and therefore CMA-ES is invariant under linear coordinate system transformations. All further considerations made for scale invariance likewise hold for invariance under general linear transformations.

Because an appropriate (initial) choice of $\boldsymbol{B}$ is usually not available in practice, general linear invariance must be complemented with adaptivity of $\boldsymbol{C}$ to make it useful in practice and eventually *adapt* a linear encoding [17].

Corollary 8.1 (Adaptive linear encoding and variable metric [17]). *The covariance matrix adaptation implements an adaptive linear problem encoding, that is, in other words, an adaptive change of basis, or a change of coordinate system, or a variable metric for an evolution strategy.*

Proof idea (The proof can be found in [16]). General linear invariance achieves identical performance on $f(\boldsymbol{B}^{-1}\boldsymbol{x})$ under respective initial conditions. Here, $\boldsymbol{B}$ is the linear problem encoding used within the algorithm. Changing (or adapting) $\boldsymbol{C}$ without changing $\boldsymbol{m}$ turns out to be equivalent with changing the encoding (or representation) $\boldsymbol{B}$ in a particular way without changing $\boldsymbol{B}^{-1}\boldsymbol{m}$ (see also [13, 16]). Also, for each possible encoding we find a respective covariance matrix $\boldsymbol{BB}^{\mathrm{T}}$. □

While adaptation of $\boldsymbol{C}$ is essential to implement general linear invariance, rotation invariance does not necessarily depend on an adaptation of $\boldsymbol{C}$: rotation invariance is already achieved for $\boldsymbol{C} \equiv \mathbf{I}$, because $\boldsymbol{B}\mathbf{I}\boldsymbol{B}^{\mathrm{T}} = \mathbf{I}$ when $\boldsymbol{B}$ is a rotation matrix, cf. Eq. (8.54). Nevertheless, it is important to note that covariance matrix adaptation *preserves* rotation invariance.

Corollary 8.2 (Rotation invariance). *The CMA-ES is invariant under search space rotations.*

Proof idea. Rotation invariance follows from Proposition 8.9 when restricted to the orthogonal group with $\boldsymbol{B}\boldsymbol{B}^{\mathrm{T}} = \mathbf{I}$ (for *any* initial state). □

8.8 An Experiment on Two Noisy Functions

We advocate testing new search algorithms always on pure random, on linear and on various (nonseparable) quadratic functions with various initializations. For the $(\mu/\mu_{\mathrm{W}}, \lambda)$-CMA-ES this has been done elsewhere with the expected results: Parameters are unbiased on pure random functions, the step-size σ grows geometrically fast on linear functions, and on convex quadratic functions the level sets of the search distribution align with the level sets of the cost function, in that $\boldsymbol{C}^{-1}$ aligns to the Hessian up to a scalar factor and small stochastic fluctuations [15, 22].

Here, we show results on the well-known Rosenbrock function

$$f(\boldsymbol{x}) = \sum_{i=1}^{n-1} 100\,(x_i^2 - x_{i+1})^2 + (x_i - 1)^2 \;,$$

where the possible achievement is less obvious. In order to "unsmoothen" the landscape, a noise term is added: Each function value is multiplied with

$$\exp\left(\frac{\alpha_N}{2\,n} \times (G + C/10)\right) + \frac{\alpha_N}{2\,n} \times (G + C/10) \;, \tag{8.55}$$

where G and C are standard Gauss (normal) and standard Cauchy distributed random numbers, respectively. All four random numbers in (8.55) are sampled independently each time f is evaluated. The term is a mixture between the common normal noise $1 + G$, which we believe has a principal "design flaw" [30], and the log-normal noise $\exp(G)$ which is alone comparatively easy to solve, each mixed with a heavy tail distribution which cannot be alleviated through averaging. We believe that this adds several difficulties on top of each other.

We show results for two noise levels, $\alpha_N = 0.01$ and $\alpha_N = 1$. A section through the 5-D and the 20-D landscape for $\alpha_N = 1$ is shown in Fig. 8.8. The lower dimensional landscape appears more disturbed but is not more difficult to optimize.

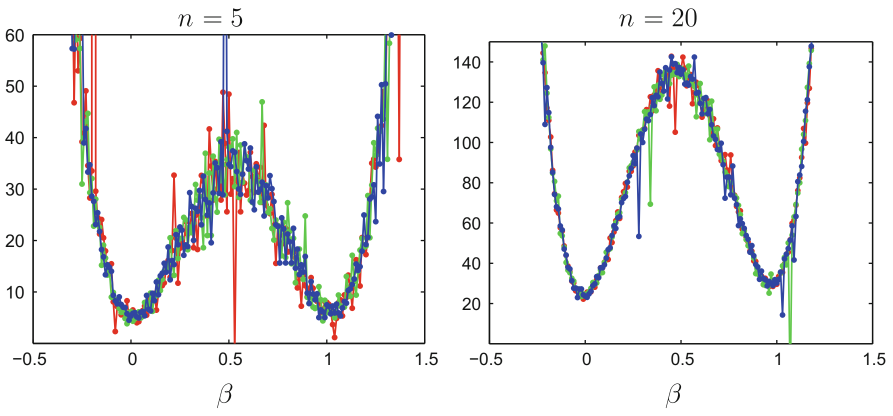


Fig. 8.8 Both figures show three sections of the Rosenbrock function for $\alpha_N = 1$ and argument $\boldsymbol{x} = \beta \times \mathbf{1} + \frac{1}{20}\mathcal{N}(\mathbf{0}, \mathbf{I})$. All graphs show 201 points for $\beta \in [-0.5, 1.5]$ and a single realization of $\mathcal{N}(\mathbf{0}, \mathbf{I})$ in each subfigure. The *left basin* about zero is initially highly attractive (cf., for example, Fig. 8.9, *upper right*) but is not nearby a local or global optimum. The basin around $\beta = 1$ is close to the global optimum at $\mathbf{1}$ and is monotonically (nonvisibly) connected to the left basin

Figure 8.9 shows the output from a typical run for $\alpha_N = 0.01$ of the $(\mu/\mu_W, \lambda)$-CMA-ES with $\boldsymbol{m}_0 = -\mathbf{1}$ and $\sigma_0 = 1$ (correctly presuming that in all variables $m_i \pm 3\sigma_0$ embrace the optimum at $\mathbf{1}$). The calling sequence in Matlab was[5]

```
opts.evalparallel = 'on'; % only one feval() call per iteration
cmaes('frosennoisy', -ones(20,1), 1, opts);  % run CMA-ES
plotcmaesdat;                % plot figures using output files
```

The default population size for $n = 20$ is $\lambda = 12$. An error of 10^{-9}, very close to the global optimum, is reached after about 20,000 function evaluations (without covariance matrix adaptation it takes about 250,000 function evaluations to reach 10^{-2}). The effect of the noise is hardly visible in the performance. In some cases, the optimization only finds the local optimum of the function close to $(-1, 1, \ldots, 1)^{\mathrm{T}}$; in some cases the noise leads to a failure to approach any optimum (see also below).

The main challenge on the Rosenbrock function is to follow a winding ridge, in the figure between evaluation 1,000 and 15,000. The ridge seems not particularly narrow: The observed axis ratio is about twenty, corresponding to a condition number of 400. But the ridge constantly changes its orientation (witnessed by the lower-right subfigure). Many stochastic search algorithms are not able to follow this ridge and get stuck with a function value larger than one.

[5]Source code is available at http://www.lri.fr/~hansen/cmaes_inmatlab.html and will be accessible at http://cma.gforge.inria.fr/ in the future. In our experiment, version 3.40.beta was used with Matlab.

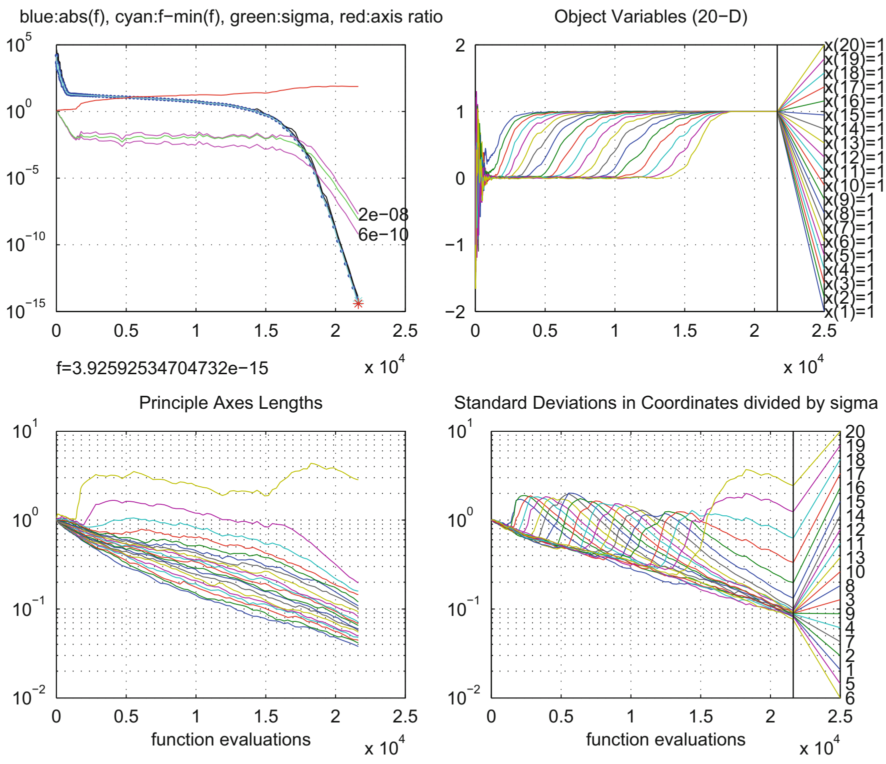


Fig. 8.9 A typical run of the $(\mu/\mu_{\mathrm{W}}, \lambda)$-CMA-ES on the Rosenbrock function ($n = 20$) with a small disturbance of the function value ($\alpha_N = 0.01$). All values are plotted against number of objective function evaluations. *Upper left*: Iteration-wise best function value (*thick blue graph*), median and worst function value (*black graphs*, mainly hidden), square root of the condition number of $\boldsymbol{C}_k$ (*increasing red graph*), smallest and largest coordinate-wise standard deviation of the distribution $\mathcal{N}(\mathbf{0}, \sigma_k^2 \boldsymbol{C}_k)$ with final values annotated (*magenta*), and σ_k following closely the largest standard deviation (*light green*). *Lower left*: Square roots of eigenvalues of $\boldsymbol{C}_k$, sorted. *Upper right*: Incumbent solution $\boldsymbol{m}_k$. *Lower right*: Square roots of diagonal elements of $\boldsymbol{C}_k$

In Fig. 8.10, the noise term is set to $\alpha_N = 1$, generating a highly rugged landscape (Fig. 8.8) and making it even harder to follow the winding ridge. Most search algorithms will fail to solve this function.[6] Now, two additional heuristics are examined.

[6]There is a simple way to smooth the landscape: A single evaluation can be replaced by the median (not the mean) of a number of evaluations. Only a few evaluations reduce the dispersion considerably, but about 1,000 evaluations are necessary to render the landscape similarly smooth as with $\alpha_N = 0.01$. Together with (μ/μ_{W})-CMA-ES, single evaluations, as in Fig. 8.10, need overall the least number of function evaluations (comprising restarts).

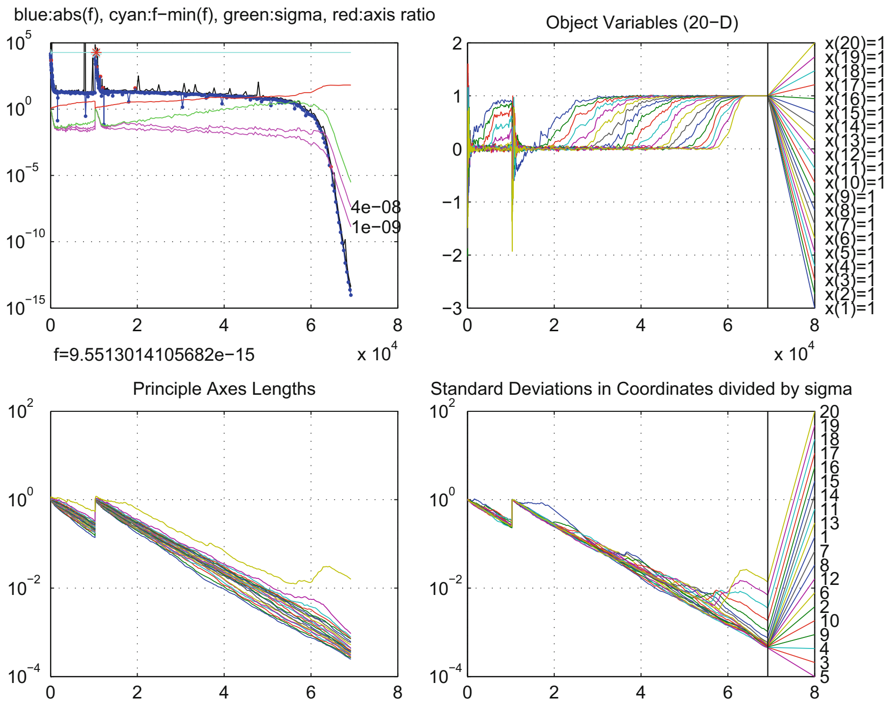


Fig. 8.10 A typical run of the IPOP-UH-CMA-ES on the noisy Rosenbrock function ($n = 20$, $\alpha_N = 1$), a (μ/μ_{W})-CMA-ES with uncertainly handling restarted with increasing population size. The highly rugged lines, partly beyond 10^5, in the *upper left* depict the worst measured function value (out of λ). One restart was necessary to converge close to the global optimum. See also Fig. 8.9 for more explanations

First, restarting the algorithm with increasing population size (IPOP, [6]). The population size is doubled for each restart. A larger population size λ is more robust to rugged landscapes, mainly because the sample variance can be larger (for $\mu_{\mathrm{eff}} < n$, the optimal step-size on the sphere function is proportional to μ_{eff} [2]). Restarting with increasing population sizes is a very effective heuristic when a good termination criterion is available.

Second, applying an uncertainty-handling (UH, [25]). The uncertainty-handling reevaluates a few solutions and measures their resulting rank changes [25]. If the rank changes exceed a threshold, an action is taken. Here, σ is increased. This prevents from getting stuck, when the noise disturbs the selection too severely, but it can also lead to divergence. This is of lesser relevance, because in this case the original algorithm would most likely have been stuck anyway. Again, a good termination criterion is essential.

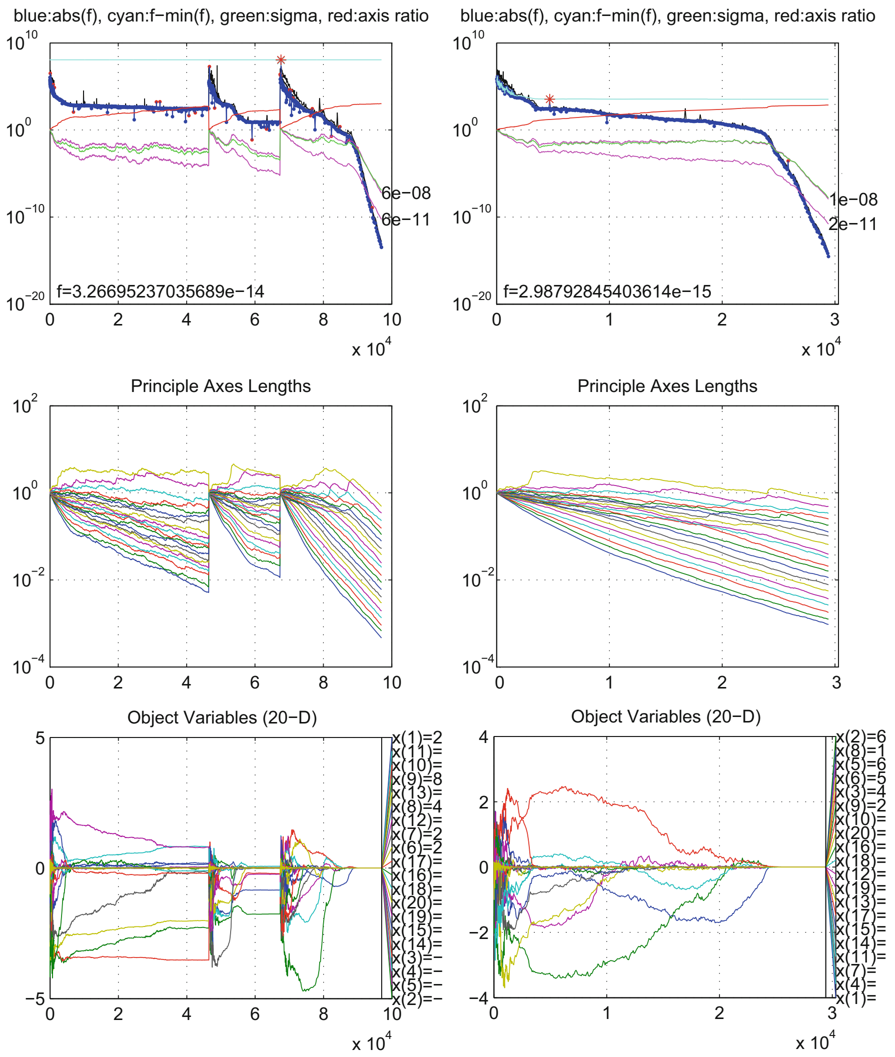


Fig. 8.11 Two typical runs of the IPOP-CMA-ES (*left*) and UH-CMA-ES (*right*, with uncertainty-handling) on the noisy ellipsoid function ($n = 20$, $\alpha_N = 1$). With $\alpha_N = 0$ the ellipsoid is solved in about 22,000 function evaluations. In the *lower left* we can well observe that the algorithm gets stuck "in the middle of nowhere" during the first two launches. See also Fig. 8.9 for more explanations

Remark that in both cases, for restarts and with the uncertainty handling, another possible action is to increase the number of function evaluations used for each individual in replacing a single value with a median.

For running IPOP-UH-CMA-ES, the following sequence is added before calling `cmaes`.

```
opts.restarts = 1;               % maximum number of restarts
opts.StopOnStagnation = 'yes';   % terminate long runs
opts.noise.on = 'yes';           % activate uncertainty-handling
```

Each restart uses the same initial conditions, here $\boldsymbol{m}_0 = -\mathbf{1}$ and $\sigma_0 = 1$ from above. For $\alpha_N = 0.01$ (Fig. 8.9) the uncertainty-handing increases the running length by about 15 %, simply due to the reevaluations (not shown). For $\alpha_N = 1$ in Fig. 8.10, it shortens the running length by a factor of about ten by reducing the number of necessary restarts. Typical for noisy functions, the restart was invoked due to stagnation of the run [20]. When repeating this experiment, in about 75 % one restart is needed to finally converge to the global optimum with $\lambda = 24$. Without uncertainty-handling it takes usually five to six restarts and a final population size of $\lambda \geq 384$. Without covariance matrix adaptation it takes about 70 times longer to reach a similar precision as in Fig. 8.10.

Experiments with the well-known Ellipsoid function,

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} 10^{6\frac{i-1}{n-1}} x_i^2$$

with the same noisy multiplier and $\alpha_N = 1$ are shown in Fig. 8.11 for IPOP-CMA-ES (left) and UH-CMA-ES (right). The function is less difficult and can be solved with a population size $\lambda = 48$ using the IPOP approach and with the default population size of 12 with UH-CMA-ES.

8.9 Summary

Designing a search algorithm is intricate. We recapitulate the principled design ideas for deriving the CMA-ES algorithm.

- Using a minimal amount of **prior assumptions** on the cost function f in order to achieve maximal robustness and minimal susceptibility to deceptiveness.
  - Generating candidate solutions by sampling a **maximum entropy** distribution adds the least amount of unwarranted information. This implies the stochastic nature of the algorithm and that no *construction* of potentially better points is undertaken. This also implies an internal quadratic model—at least when the distribution has finite variances—and stresses the importance of neighborhood. Consequently, a variable neighborhood suggests itself.
  - **Unbiasedness** of all algorithm components, given the objective function is random and independent of its argument. This principle suggests that only the current state and the selection information should bias the behavior of

  the algorithm. Adding another bias would add additional prior assumptions. We have deliberately violated this principle for uncertainty-handling as used in one experiment, where the step-size is increased under highly perturbed selection.
  - Only the *ranking* of the most recently sampled candidate solutions is used as feed-back from the objective function. This implies an attractive invariance property of the algorithm.

  Exploiting more specific information on f effectively, for example, smoothness, convexity or (partial) separability, will lead to different and more specific design decisions, with a potential advantage on smooth, convex or separable functions, respectively.
- Introducing and maintaining **invariance** properties. Even invariance is related to avoiding prior assumptions as it implies not exploiting the specific structure of the objective function f (for example, separability). We can differentiate two main cases.
  - Unconditional invariance properties do not depend on the initial conditions of the algorithm and strengthen any empirical performance observation. They allow us to unconditionally generalize empirical observations to the equivalence class of functions induced by the invariance property.
  - Invariance properties that depend on state variables of the algorithm (like σ_k for scale invariance in Fig. 8.5) must be complemented with adaptivity. They are particularly attractive, if adaptivity can drive the algorithm quickly into the most desirable state. This behavior can be empirically observed for CMA-ES on the equivalence class of convex-quadratic functions. Step-size control drives step-size σ_k close to its optimal value, and adaptation of the covariance matrix reduces these functions to the sphere model.
- Exploiting all **available information** effectively. The available information and its exploitation are highly restricted by the first two demands. Using a *deterministic* ranking and *different* weights for updating $\boldsymbol{m}$ and $\boldsymbol{C}$ are due to this design principle. Also the evolution paths in Eq. (8.46) and in Eq. (8.51) are governed by exploiting otherwise unused sign information. Using the evolution paths does not violate any of the above demands, but allows us to additionally exploit dependency information between successive time steps of the algorithm.
- Solving the two most basic continuous domain functions reasonably fast. Solving the linear function and the sphere function reasonably fast implies the introduction of step-size control. These two functions are quite opposed: The latter requires convergence, the former requires divergence of the algorithm.

Finally, two **heuristic concepts** are applied in CMA-ES.

- Reinforcement of the better solutions and the better steps (variations) when updating mean and variances, respectively. This seems a rather unavoidable heuristic given a conservative use of information from f. This heuristic bears the maximum likelihood principle.

Table 8.1 Summary of the update equations for the state variables in the $(\mu/\mu_{\mathrm{w}}, \lambda)$-CMA-ES with iteration index $k = 0, 1, 2, \ldots$. The chosen ordering of equations allows us to remove the iteration index in all variables but $\boldsymbol{m}_k$. Unexplained parameters and constants are given in Table 8.2

Given $k \in \mathbb{N} \cup \{0\}$, $\boldsymbol{m}_k \in \mathbb{R}^n$, $\sigma_k \in \mathbb{R}_+$, $\boldsymbol{C}_k \in \mathbb{R}^{n\times n}$ positive definite, $\boldsymbol{p}^{\sigma}{}_k \in \mathbb{R}^n$, and $\boldsymbol{p}^{\mathrm{c}}{}_k \in \mathbb{R}^n$

$$\boldsymbol{x}_i \sim \boldsymbol{m}_k + \sigma_k \times \mathcal{N}(\boldsymbol{0}, \boldsymbol{C}_k) \quad \text{is normally distributed for } i = 1, \ldots, \lambda \tag{8.56}$$

$$\boldsymbol{m}_{k+1} = \boldsymbol{m}_k + c_{\mathrm{m}} \sum_{i=1}^{\mu} w_i\, (\boldsymbol{x}_{i:\lambda} - \boldsymbol{m}_k) \quad \text{where } f(\boldsymbol{x}_{1:\lambda}) \le \cdots \le f(\boldsymbol{x}_{\mu:\lambda}) \le f(\boldsymbol{x}_{\mu+1:\lambda}) \ldots \tag{8.57}$$

$$\boldsymbol{p}^{\sigma}_{k+1} = (1 - c_\sigma)\boldsymbol{p}^{\sigma}{}_k + \sqrt{c_\sigma(2 - c_\sigma)\mu_{\mathrm{eff}}}\, \boldsymbol{C}_k{}^{-\frac{1}{2}}\, \frac{\boldsymbol{m}_{k+1} - \boldsymbol{m}_k}{c_{\mathrm{m}}\, \sigma_k} \tag{8.58}$$

$$\boldsymbol{p}^{\mathrm{c}}_{k+1} = (1 - c_{\mathrm{c}})\boldsymbol{p}^{\mathrm{c}}{}_k + h_\sigma \sqrt{c_{\mathrm{c}}(2 - c_{\mathrm{c}})\mu_{\mathrm{eff}}}\, \frac{\boldsymbol{m}_{k+1} - \boldsymbol{m}_k}{c_{\mathrm{m}}\, \sigma_k} \tag{8.59}$$

$$\begin{aligned}\boldsymbol{C}_{k+1} &= (1 - c_1 + (1 - h_\sigma{}^2)\, c_1 c_{\mathrm{c}} (2 - c_{\mathrm{c}}))\, \boldsymbol{C}_k \\ &\quad + c_1 \boldsymbol{p}^{\mathrm{c}}_{k+1} \boldsymbol{p}^{\mathrm{c}}_{k+1}{}^{\mathrm{T}} + c_\mu \sum_{i=1}^{\mu} w_i \left(\frac{\boldsymbol{x}_{i:\lambda} - \boldsymbol{m}_k}{\sigma_k} \times \frac{(\boldsymbol{x}_{i:\lambda} - \boldsymbol{m}_k)^{\mathrm{T}}}{\sigma_k} - \boldsymbol{C}_k \right)\end{aligned} \tag{8.60}$$

$$\sigma_{k+1} = \sigma_k \times \exp\left(1 \wedge \frac{c_\sigma}{d_\sigma} \left(\frac{\|\boldsymbol{p}^{\sigma}_{k+1}\|}{\mathsf{E}\|\mathcal{N}(\boldsymbol{0}, \mathbf{I})\|} - 1 \right)\right) \tag{8.61}$$

- Orthogonality of successive steps. This heuristic is a rather common conception in continuous domain search.

Pure random search, where the sample distribution remains constant in the iteration sequence, follows most of the above design principles and has some attractive robustness features. However, pure random search neither accumulates information from the past in order to modify the search distribution, nor changes and adapts internal state variables. Adaptivity of state variables, however, detaches the algorithm from the initial conditions and lets (additional) invariance properties come to life. Only invariance to increasing f-value transformations (Proposition 8.1) is independent of state variables of the search algorithm. We draw the somewhat surprising conclusion that the abstract notion of invariance leads, by advising the introduction of adaptivity, when carefully implemented, to a vastly improved practical performance.

Despite its generic, principled design, the **practical performance** of CMA-ES turns out to be surprisingly competitive, or even superior, also in comparatively specific problem classes. This holds in particular when more than $100n$ function evaluations are necessary to find a satisfactory solution [26]—even, for example, on smooth unimodal nonquadratic functions [8], or on highly multimodal functions [21] and on noisy or highly rugged functions [20]. In contrast, much better search heuristics are available given (nearly) convex-quadratic problems or (partially) separable multimodal problems.

Table 8.2 Default parameter values of the (μ/μ_{W})-CMA-ES, where by definition $\sum_{i=1}^{\mu}|w_i| = 1$ and $\mu_{\mathrm{eff}}^{-1} = \sum_{i=1}^{\mu} w_i^2$

$\alpha_{cov} = 2 \wedge \lambda/3$	could be chosen < 2, e.g. $\alpha_{\mathrm{cov}} = 0.5$ for noisy problems
$\lambda = 4 + \lfloor 3 \ln n \rfloor$	population size, see also [6, 19]
$\mu = \left\lfloor \frac{\lambda}{2} \right\rfloor$	parent number
$w_i = \frac{\ln\left(\frac{\lambda+1}{2}\right) - \ln i}{\sum_{j=1}^{\mu}\left(\ln\left(\frac{\lambda+1}{2}\right) - \ln j\right)}$	recombination weights for $i = 1, \ldots, \mu$
$c_{\mathrm{m}} = 1$	learning rate for the mean with $\kappa = \frac{1}{c_{\mathrm{m}}} \geq 1$
$c_\sigma = \frac{\mu_{\mathrm{eff}} + 2}{n + \mu_{\mathrm{eff}} + 5}$	cumulation constant for step-size ($1/c_\sigma$—respective time constant)
$d_\sigma = 1 + c_\sigma + 2\,\max\left(0, \sqrt{\frac{\mu_{\mathrm{eff}} - 1}{n + 1}} - 1\right)$	step-size damping, is usually close to one
$c_{\mathrm{c}} = \frac{4 + \mu_{\mathrm{eff}}/n}{n + 4 + 2\mu_{\mathrm{eff}}/n}$	cumulation constant for $\boldsymbol{p}^{\mathrm{c}}$
$c_1 = \frac{\alpha_{\mathrm{cov}}}{(n + 1.3)^2 + \mu_{\mathrm{eff}}}$	cov. matrix learning rate for the rank one update using $\boldsymbol{p}^{\mathrm{c}}$
$c_\mu = \min\left(1 - c_1, \alpha_{\mathrm{cov}} \frac{\mu_{\mathrm{eff}} - 2 + 1/\mu_{\mathrm{eff}}}{(ns + 2)^2 + \alpha_{\mathrm{cov}}\mu_{\mathrm{eff}}/2}\right)$	learning rate for rank-μ update

Acknowledgements The authors would like to express their gratitude to Marc Schoenauer for his kind and consistent support.

Appendix

The $(\mu/\mu_{\mathrm{W}}, \lambda)$-CMA-ES, as described in this chapter, is summarized in Table 8.1. We have $\boldsymbol{p}_{k=0}^{\sigma} = \boldsymbol{p}_{k=0}^{\mathrm{c}} = \mathbf{0}$, $\boldsymbol{C}_{k=0} = \mathbf{I}$, while $\boldsymbol{m}_{k=0} \in \mathbb{R}^n$ and $\sigma_{k=0} > 0$ are user defined. Additionally, $\boldsymbol{x}_{i:\lambda}$ is the i-th best of the solutions $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_\lambda$,

$$h_\sigma = \begin{cases} 1 & \text{if } \frac{\|\boldsymbol{p}_{k+1}^{\sigma}\|^2}{1-(1-c_\sigma)^{2(k+1)}} < \left(2 + \frac{4}{n+1}\right) n \\ 0 & \text{otherwise} \end{cases},$$

for $\mathsf{E}\|\mathcal{N}(\mathbf{0}, \mathbf{I})\| = \sqrt{2}\,\Gamma(\frac{n+1}{2})/\,\Gamma(\frac{n}{2}) \approx \sqrt{n - 1/2}$ we use the better approximation $\sqrt{n}\left(1 - \frac{1}{4n} + \frac{1}{21n^2}\right)$, and $\boldsymbol{C}_k{}^{-\frac{1}{2}}$ is symmetric with positive eigenvalues and satisfies

${\boldsymbol{C}_k}^{-\frac{1}{2}}{\boldsymbol{C}_k}^{-\frac{1}{2}} = (\boldsymbol{C}_k)^{-1}$. The binary $\wedge$ operator depicts the minimum of two values with low operator precedence. The default parameter values are shown in Table 8.2.

References

1. Y. Akimoto, Y. Nagata, I. Ono, S. Kobayashi, Bidirectional relation between CMA evolution strategies and natural evolution strategies, in *Proceedings of the Parallel Problem Solving from Nature – PPSN XI, Part I*, Kraków, ed. by R. Schaefer, C. Cotta, J. Kolodziej, G. Rudolph. Lecture Notes in Computer Science, vol. 6238 (Springer, 2010), pp. 154–163
2. D. Arnold, Optimal weighted recombination, in *Foundations on Genetic Algorithms FOGA 2005*, Aizu-Wakamatsu City. Lecture Notes in Computer Science, vol. 3469 (Springer, 2005), pp. 215–237
3. D. Arnold, Weighted multirecombination evolution strategies. Theor. Comput. Sci. **361**(1), 18–37 (2006)
4. D.V. Arnold, N. Hansen, Active covariance matrix adaptation for the (1+1)-CMA-ES, in *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2010*, Portland, 2010, pp. 385–392
5. L. Arnold, A. Auger, N. Hansen, Y. Ollivier, Information-geometric optimization algorithms: a unifying picture via invariance principles. arXiv:1106.3708 (Arxiv preprint) (2011)
6. A. Auger, N. Hansen, A restart CMA evolution strategy with increasing population size, in *The 2005 IEEE International Congress on Evolutionary Computation (CEC 2005)*, Edinburgh, ed. by B. McKay et al., vol. 2, 2005, pp. 1769–1776
7. A.Auger, N. Hansen, Reconsidering the progress rate theory for evolution strategies in finite dimensions, in *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation GECCO*, Seattle (ACM, 2006), pp. 445–452
8. A. Auger, N. Hansen, J. Zerpa, R. Ros, M. Schoenauer, Experimental comparisons of derivative free optimization algorithms, in *8th International Symposion on Experimental Algorithms SEA 2009*, Dortmund. Lecture Notes in Computer Science, vol. 5526 (Springer, 2009), pp. 3–15
9. H.G. Beyer, *The Theory of Evolution Strategies*. Natural Computing Series (Springer, Heidelberg, 2001)
10. D. Brockhoff, A. Auger, N. Hansen, D.V. Arnold, T. Hohm, Mirrored sampling and sequential selection for evolution strategies, in *Parallel Problem Solving from Nature (PPSN XI)*, Kraków, ed. by R. Schaefer et al. LNCS, vol. 6238 (Springer, 2010) pp. 11–20
11. T. Glasmachers, T. Schaul, Y. Sun, D. Wierstra, J. Schmidhuber, Exponential natural evolution strategies, in *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2010*, Portland, ed. by M. Pelikan, J. Branke (ACM, 2010) pp. 393–400
12. N. Hansen, The CMA evolution strategy: a tutorial, http://www.lri.fr/~hansen/cmatutorial.pdf
13. N. Hansen, Invariance, self-adaptation and correlated mutations in evolution strategies, in *Proceedings of PPSN VI, Parallel Problem Solving from Nature*, Paris, ed. by M. Schoenauer, K. Deb, G. Rudolph, X. Yao, E. Lutton, J. Merelo, H.P. Schwefel (Springer, 2000), pp. 355–364
14. N. Hansen, An analysis of mutative σ-self-adaptation on linear fitness functions. Evol. Comput. **14**(3), 255–275 (2006)
15. N. Hansen, The CMA evolution strategy: a comparing review, in *Towards a New Evolutionary Computation. Advances on Estimation of Distribution Algorithms*, Hefei, ed. by J. Lozano, P. Larranaga, I. Inza, E. Bengoetxea (Springer, 2006), pp. 75–102
16. N. Hansen, Adaptive encoding for optimization. Research Report RR-6518, INRIA (2008), http://hal.inria.fr/inria-00275983/en/
17. N. Hansen, Adaptive encoding: how to render search coordinate system invariant, in *Parallel Problem Solving from Nature (PPSN X)*, Dortmund, ed. by G. Rudolph et al. LNCS, 2008, pp. 205–214

18. N. Hansen, CMA-ES with two-point step-size adaptation. Technical Report, RR-6527, INRIA (2008), http://hal.inria.fr/inria-00276854/en/
19. N. Hansen, Benchmarking a BI-population CMA-ES on the BBOB-2009 function testbed, in *Workshop Proceedings of the GECCO Genetic and Evolutionary Computation Conference*, Montreal (ACM, 2009), pp. 2389–2395
20. N. Hansen, Benchmarking a BI-population CMA-ES on the BBOB-2009 noisy testbed, in *Workshop Proceedings of the GECCO Genetic and Evolutionary Computation Conference*, Montreal (ACM, 2009), pp. 2397–2402
21. N. Hansen, S. Kern, Evaluating the CMA evolution strategy on multimodal test functions, in *Parallel Problem Solving from Nature PPSN VIII*, Birmingham, ed. by X. Yao, et al. Lecture Notes in Computer Science, vol. 3242 (Springer, 2004), pp. 282–291
22. N. Hansen, A. Ostermeier, Completely derandomized self-adaptation in evolution strategies. Evol. Comput. **9**(2), 159–195 (2001)
23. N. Hansen, R. Ros, Benchmarking a weighted negative covariance matrix update on the BBOB-2010 noiseless testbed, in *Genetic and Evolutionary Computation Conference, GECCO 2010, Companion Material*, Portland, 2010, pp. 1673–1680
24. N. Hansen, A. Gawelczyk, A. Ostermeier, Sizing the population with respect to the local progress in (1, λ)-evolution strategies—a theoretical analysis, in *IEEE International Conference on Evolutionary Computation*, Perth, vol. 1, 1995, pp. 80–85
25. N. Hansen, S. Niederberger, L. Guzzella, P. Koumoutsakos, A method for handling uncertainty in evolutionary optimization with an application to feedback control of combustion. IEEE Trans. Evol. Comput. **13**(1), 180–197 (2009)
26. N. Hansen, A. Auger, R. Ros, S. Finck, P. Pošík, Comparing results of 31 algorithms from the black-box optimization benchmarking BBOB-2009, in *Workshop Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2010)*, Portland (ACM, 2010), pp. 1689–1696
27. J. Jägersküpper, Lower bounds for hit-and-run direct search, in *Stochastic Algorithms: Foundations and Applications – SAGA 2007*, Zurich, ed. by Yao, Xin et al. LNCS, vol. 4665 (Springer, Berlin/Heidelberg, 2007) pp. 118–129
28. J. Jägersküpper, Lower bounds for randomized direct search with isotropic sampling. Oper. Res. Lett. **36**(3), 327–332 (2008)
29. G. Jastrebski, D. Arnold, Improving evolution strategies through active covariance matrix adaptation, in *The 2006 IEEE International Congress on Evolutionary Computation (CEC 2006)*, Vancouver, 2006, pp. 2814–2821
30. M. Jebalia, A. Auger, N. Hansen, Log linear convergence and divergence of the scale-invariant (1+1)-ES in noisy environments. Algorithmica **59**(3), 425–460 (2011)
31. T. Jones, S. Forrest, Fitness distance correlation as a measure of problem difficulty for genetic algorithms, in *Proceedings of the 6th International Conference on Genetic Algorithms, ICGA*, Pittsburgh, ed. by L.J. Eshelman (Morgan Kaufmann, 1995), pp. 184–192
32. A. Ostermeier, A. Gawelczyk, N. Hansen, Step-size adaptation based on non-local use of selection information, in *Parallel Problem Solving from Nature PPSN IV*, Jerusalem, ed. by Y. Davidor et al. Lecture Notes in Computer Science, vol. 866 (Springer, 1994), pp. 189–198
33. I. Rechenberg, *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution* (Frommann-Holzboog, Stuttgart, 1973)
34. R. Salomon, J.L. van Hemmen, Accelerating backpropagation through dynamic self-adaptation. Neural Netw. **9**(4), 589–601 (1996)
35. M. Schumer, K. Steiglitz, Adaptive step size random search. IEEE Trans. Autom. Control **13**(3), 270–276 (1968)
36. H.P. Schwefel, *Numerical Optimization of Computer Models* (Wiley, New York, 1981)
37. T. Suttorp, N. Hansen, C. Igel, Efficient covariance matrix update for variable metric evolution strategies. Mach. Learn. **75**(2), 167–197 (2009)
38. O. Teytaud, H. Fournier, Lower bounds for evolution strategies using VC-dimension, in *Parallel Problem Solving from Nature PPSN X*, Dortmund. Lecture Notes in Computer Science, vol. 5199 (Springer, 2008) pp. 102–111

39. O. Teytaud, S. Gelly, General lower bounds for evolutionary algorithms, in *Parallel Problem Solving from Nature PPSN IX*, Reykjavik. Lecture Notes in Computer Science, vol. 4193 (Springer, 2006), pp. 21–31
40. D. Wierstra, T. Schaul, J. Peters, J. Schmidhuber, Natural evolution strategies, in *IEEE Congress on Evolutionary Computation*, Hong Kong (IEEE, 2008), pp. 3381–3387

Chapter 9
Parsimony Pressure Made Easy: Solving the Problem of Bloat in GP

Riccardo Poli and Nicholas Freitag McPhee

Abstract The parsimony pressure method is perhaps the simplest and most frequently used method to control bloat in genetic programming (GP). In this chapter we first reconsider the size evolution equation for genetic programming developed in Poli and McPhee (Evol Comput 11(2):169–206, 2003) and rewrite it in a form that shows its direct relationship to Price's theorem. We then use this new formulation to derive theoretical results that show how to practically and optimally set the parsimony coefficient dynamically during a run so as to achieve complete control over the growth of the programs in a population. Experimental results confirm the effectiveness of the method, as we are able to tightly control the average program size under a variety of conditions. These include such unusual cases as dynamically varying target sizes so that the mean program size is allowed to grow during some phases of a run, while being forced to shrink in others.

9.1 Introduction

Starting in the early 1990s researchers began to notice that in addition to progressively increasing their mean and best fitness, genetic programming (GP) populations also exhibited certain other dynamics. In particular, it was often observed that, while the average size (number of nodes) of the programs in a population was initially fairly static (if noisy), at some point the average program size would start growing

R. Poli (✉)
School of Computer Science and Electronic Engineering, University of Essex, Colchester, Essex, UK
e-mail: rpoli@essex.ac.uk

N.F. McPhee
Division of Science and Mathematics, University of Minnesota, Morris, MN, USA
e-mail: mcphee@morris.umn.edu

at a rapid pace. Typically this increase in program size was not accompanied by any corresponding increase in fitness.

This phenomenon, which is known as *bloat* and can succinctly be defined as *program growth without (significant) return in terms of fitness*, has been the subject of intense study in GP, both because of its initially surprising nature, and because of its significant practical effects. (Large programs are computationally expensive to evolve and later use, can be hard to interpret, and may exhibit poor generalisation.) Over the years, many theories have been proposed to explain various aspects of bloat [27, Sect. 11.3]. We review the key theoretical results on bloat in Sect. 9.2, with special emphasis on the *size evolution equation* [28] as it forms the basis for this new approach.

While the jury was out on the causes of bloat, practitioners still had the practical problem of combating bloat in their runs. Consequently, a variety of practical techniques have been proposed to counteract bloat; we review these in Sect. 9.3. We will particularly focus on the *parsimony pressure method* [14, 38], which is perhaps the simplest and most frequently used method to control bloat in genetic programming. This method effectively treats the minimisation of size as a soft constraint and attempts to enforce this constraint using the penalty method, i.e., by decreasing the fitness of programs by an amount that depends on their size. The penalty is typically simply proportional to program size. The intensity with which bloat is controlled is, therefore, determined by one parameter called the *parsimony coefficient*. The value of this coefficient is very important: Too small a value and runs will still bloat wildly; too large a value and GP will take the minimisation of size as its main target and will almost ignore fitness, thus converging towards extremely small but useless programs. Unfortunately, however, the "correct" values of this coefficient are highly dependent on particulars such as the problem being solved, the choice of functions and terminals, and various parameter settings. Very little theory, however, has been put forward to aid in setting the parsimony coefficient in a principled manner, forcing users to proceed by trial and error.

This chapter presents a simple, effective, and theoretically grounded solution to this problem. In Sect. 9.4, we reconsider the size evolution equation for GP developed in [28], rewriting it in a form that shows its direct relationship to Price's theorem [18, 27, 29]. We then use this new formulation to derive theoretical results that tell us how to practically and optimally set the parsimony coefficient dynamically during a run so as to achieve very tight control over the average size of the programs in a population. We test our theory in Sect. 9.5, where we report extensive empirical results, showing how accurately the method controls program size in a variety of conditions. We then conclude in Sect. 9.6.

9.2 Bloat in Theory

As mentioned above, there are several theories of bloat. For example, the *replication accuracy theory* [22] states that the success of a GP individual depends on its ability to have offspring that are functionally similar to the parent. As a consequence, GP evolves towards (bloated) representations that increase replication accuracy. The *removal bias theory* [37] observes that inactive code in a GP tree (code that is not executed, or is executed but its output is then discarded) tends to be low down in the tree (i.e., near its leaves), residing therefore in smaller-than-average-size subtrees. Crossover events excising inactive subtrees produce offspring with the same fitness as their parents. On average the inserted subtree is bigger than the excised one, so such offspring are bigger than average while retaining the fitness of their parent, leading ultimately to growth in the average program size. Another important theory, the *nature of program search spaces theory* [17, 19], predicts that above a certain size, the distribution of fitnesses does not vary with size. Since there are more long programs, the number of long programs of a given fitness is greater than the number of short programs of the same fitness. Over time GP samples longer and longer programs simply because there are more of them.

The explanations for bloat mentioned above are largely qualitative. There have been, however, some efforts to mathematically formalise and verify these theories. For example, Banzhaf and Langdon [3] defined an executable model where only the fitness, the size of active code and the size of inactive code of programs were represented (i.e., there was no representation of program structure). Fitnesses of individuals were drawn from a bell-shaped distribution, while active and inactive code lengths were modified by a size-unbiased mutation operator. The model was able to reproduce some effects which are found in GP runs. Rosca proposed a similar, but slightly more sophisticated model which also included an analogue of crossover [31]. A strength of these types of models is their simplicity. A weakness is that they suppress or remove many details of the representation and operators typically used in GP. This makes it difficult to verify if all the phenomena observed in the model have analogues in GP runs, and if all important behaviours of GP in relation to bloat are captured by a model.

In [24, 28], a *size evolution equation* for genetic programming was developed, which is an exact formalisation of the dynamics of average program size:

$$E[\mu(t+1)] = \sum_l S(G_l)p(G_l,t), \tag{9.1}$$

Here $\mu(t+1)$ is the mean size of the programs in the population at generation $t+1$, $E[\ ]$ is the expectation operator, G_l is the set of all programs having a particular shape l, $S(G_l)$ is the size of programs in the set G_l (i.e., programs having the shape l), and $p(G_l,t)$ is the probability of selecting programs from G_l (i.e., of shape l) from the population in generation t. This can be rewritten in terms of the expected change in average program size as:

$$E[\mu(t+1) - \mu(t)] = \sum_l S(G_l)(p(G_l,t) - \Phi(G_l,t)), \tag{9.2}$$

where $\Phi(G_l,t)$ is the proportion of programs of shape G_l in the current generation. Both equations apply to a generational GP system with selection and any form of symmetric subtree crossover[1] (see [28] for a proof), but not mutation. Size-evolution equations can be derived also for mutation [35], but they are different from Eqs. (9.1) and (9.2). No other assumption is required by these equations (e.g., infinite population).

These equations make a prediction only one-step in the future. However, as we will see in the paper this is sufficient for many practical purposes. Note also that Eqs. (9.1) and (9.2) do not directly explain bloat. They are, however, important because they constrain what can and cannot happen size-wise in GP populations. So, any explanation for bloat (including the theories summarised in this section) has to agree with these results. In particular, Eq. (9.1) predicts that, for symmetric subtree-swapping crossover operators, the mean program size evolves as if selection only was acting on the population. This means that if there is a variation in mean size (bloat, for example) it must be the result of some form of positive or negative selective pressure on some or all of the shapes G_l. Equation (9.2) shows that there can be bloat only if the selection probability $p(G_l, t)$ is different from the proportion $\Phi(G_l, t)$ for at least some l. In particular, for bloat to happen there will have to be some short G_l's for which $p(G_l, t) < \Phi(G_l, t)$ and also some longer G_l's for which $p(G_l, t) > \Phi(G_l, t)$ (at least on average). As we will see later, Eqs. (9.1) and (9.2) are the starting point for the work reported here.

We conclude this review with a recent explanation for bloat called the *crossover bias theory* [5, 26] which is based in significant part and is consistent with the size evolution equation above. On average, each application of subtree crossover removes as much genetic material as it inserts. So, crossover on its own does not produce growth or shrinkage. However, while the *mean* program size is unaffected, *higher moments* of the distribution are. In particular, crossover pushes the population towards a particular distribution of program sizes (a Lagrange distribution of the second kind), where small programs have a much higher frequency than longer ones. For example, crossover generates a very high proportion of single-node individuals. In virtually all problems of practical interest, very small programs have no chance of solving the problem. As a result, programs of above-average length have a selective advantage over programs of below-average length. Consequently, the mean program size increases.

9.3 Bloat Control in Practice

The traditional technique of fixing a maximum size or depth for any individuals to be inserted in the population are by-and-large ineffective at controlling bloat. In fact, in some cases they can even induce growth [6]. So, over the years numerous empirical techniques have been proposed to control bloat [19, 36]. These include *size-fair*

[1] In a symmetric operator the probability of selecting particular crossover points in the parents does not depend on the order in which the parents are drawn from the population.

crossover and *size-fair mutation* [4, 16], which constrain the choices made during the execution of a genetic operation so as to actively prevent growth. In size-fair crossover, for example, the crossover point in the first parent is selected randomly, as in standard crossover. The size of the subtree to be excised is then used to constrain the choice of the second crossover point so as to guarantee that the subtree chosen from the second parent will not be "unfairly" big. Another technique, the *Tarpeian method* [25], controls bloat by acting directly on the selection probabilities in Eq. (9.2) by setting the fitness of randomly chosen longer than average programs to 0. *Multi-objective optimisation* (with two objectives: fitness and size) has also been used to control bloat. For example, [7] used a modified selection based on the Pareto criterion to reduce code growth without significant loss of solution accuracy, and [13] used a Pareto approach on regression problems in an industrial setting.

Older methods include several *mutation operators* that may help control the average tree size in the population while still introducing new genetic material. Kinnear [11] proposes a mutation operator which prevents the offspring's depth being more then 15 % larger than its parent. Langdon [15] proposes two mutation operators in which the new random subtree is on average the same size as the code it replaces. In *Hoist mutation* [12] the new subtree is selected from the subtree being removed from the parent, guaranteeing that the new program will be smaller than its parent. *Shrink mutation* [2] is a special case of subtree mutation where the randomly chosen subtree is replaced by a randomly chosen terminal. McPhee and Poli [23] provides theoretical analysis and empirical evidence that combinations of subtree crossover and subtree mutation operators can control bloat in linear GP systems.

None of the methods mentioned above, however, has gained as much widespread acceptance as the *parsimony pressure method* [14, 38]. The method works as follows. Let $f(x)$ be the fitness of program x. When the parsimony pressure is applied we define and use a new fitness function

$$f_p(x) = f(x) - c\ell(x) \tag{9.3}$$

where $\ell(x)$ is the size of program x and c is a constant known as the *parsimony coefficient*.[2] Zhang and Mühlenbein [38] showed the benefits of adaptively adjusting the coefficient c at each generation in experiments on the evolution of sigma-pi neural networks with GP, but most implementations and results in the literature actually keep c constant. As we will see in Sect. 9.4, however, a dynamic c is in fact essential to obtain full control of bloat.

The parsimony pressure method can be seen as a way to address the generalisation-accuracy tradeoff common in machine learning [33, 38]. There are also connections between this method and the minimum description length (MDL) principle used to control bloat in [8–10]. The MDL approach uses a fitness function which combines program complexity (expressed as the number of bits necessary

[2]Naturally, while f_p is used to guide evolution, one needs to still use the original fitness function f to recognise solutions and stop runs.

to encode the program's tree) and classification error (expressed as the number of bits necessary to encode the errors on all fitness cases). Rosca also linked the parsimony pressure method to his approximate evolution equations for rooted-tree schemata [30, 32–34].

Naturally, controlling bloat while at the same time maximising fitness turns the evolution of programs into either a multi-objective optimisation problem or, at least, into a constrained optimisation problem. Thus, as mentioned in Sect. 9.1, we should expect (and numerous results in the literature show this) that excessively aggressive methods to control bloat may lead to poor performance (in terms of ability to solve the problem at hand) of the evolved programs. The parsimony pressure method is not immune from this risk. So, although good control of bloat can be obtained with a careful choice of the parsimony coefficient, the choice of such a coefficient is an important but delicate matter. To date, however, trial and error remains the only general method for setting the parsimony coefficient. Furthermore, with a constant c the method can only achieve *partial control* over the dynamics of the average program size over time.

In this chapter we aim to change all that, theoretically deriving and testing an easy and practical modification of the parsimony pressure technique which provides *extremely tight* control over the dynamics of the mean program size.

9.4 Optimal Parsimony Pressure

In this section we show the relationship between the size evolution equation and Price's theorem [29]. We also show how to use this new form of the size evolution equation to solve for dynamic parsimony coefficients that will allow for various types of control of the average program size (e.g., Eqs. (9.14), (9.15), (9.19), and (9.20), which follow). Despite their theoretical origin, these forms of size control are straightforward to add to most GP systems and (as is shown in Sect. 9.5) can provide exceptionally tight control over the average population size.

Let us start by considering Eq. (9.1) again. With trivial manipulations it can be rewritten in terms of length-classes, rather than tree shapes, obtaining

$$E[\mu(t+1)] = \sum_{\ell} \ell p(\ell, t) \tag{9.4}$$

where the index ℓ ranges over all program sizes, and

$$p(\ell, t) = \sum_{l: S(G_l) = \ell} p(G_l, t) \tag{9.5}$$

is the probability of selecting a program of length ℓ. Similarly, we can rewrite Eq. (9.2) as

$$E[\Delta\mu] = E[\mu(t+1) - \mu(t)] = \sum_{\ell} \ell \left(p(\ell,t) - \Phi(\ell,t)\right), \tag{9.6}$$

where $\Phi(\ell,t) = \sum_{l:S(G_l)=\ell} \Phi(G_l,t)$.

We now restrict our attention to fitness proportionate selection. In this case

$$p(\ell,t) = \Phi(\ell,t)\frac{f(\ell,t)}{\bar{f}(t)}, \tag{9.7}$$

where $f(\ell,t)$ is the average fitness of the programs of size ℓ and $\bar{f}(t)$ is the average fitness of the programs in the population, both computed at generation t. Then from Eq. (9.6) we obtain

$$\begin{aligned}
E[\Delta\mu] &= \sum_{\ell} \ell \left(\Phi(\ell,t)\frac{f(\ell,t)}{\bar{f}(t)} - \Phi(\ell,t)\right) \\
&= \frac{1}{\bar{f}(t)} \sum_{\ell} \ell (f(\ell,t) - \bar{f}(t))\Phi(\ell,t) \\
&= \frac{1}{\bar{f}(t)} \Big(\underbrace{\sum_{\ell} (\ell - \mu(t))(f(\ell,t) - \bar{f}(t))\Phi(\ell,t)}_{=\ \mathrm{Cov}(\ell,f)\ \text{by definition}} \\
&\qquad + \mu(t) \underbrace{\sum_{\ell} (f(\ell,t) - \bar{f}(t))\Phi(\ell,t)}_{=\ 0\ \text{by definition of}\ \bar{f}(t)} \Big),
\end{aligned}$$

where $\mu(t) = \sum_{\ell} \ell\, \Phi(\ell,t)$ is the current average program size. So,

$$E[\Delta\mu] = \frac{\mathrm{Cov}(\ell,f)}{\bar{f}(t)}. \tag{9.8}$$

This result is important because it shows that Eq. (9.6), our coarse-grained version of Eq. (9.2), is in fact a form of Price's theorem (see [1, 18, 29] for a detailed review). While Price's theorem is generally applicable to "inheritable features" in an evolving system, only informal arguments have so far been made conjecturing that size might be one such feature [18]. Our result, *proves* the conjecture.

We are now in a position to more clearly see the effects of parsimony pressure and, more generally, of any form of program-size control based on the following generalisation of Eq. (9.3):

$$f_p(x,t) = f(x) - g(\ell(x),t) \tag{9.9}$$

where g is a function of program size, $\ell(x)$, and generation, t. To achieve this we consider Eq. (9.8) when the fitness function $f(x)$ is replaced by $f_p(x,t)$. We obtain

$$\begin{aligned} E[\Delta\mu] &= \frac{\mathrm{Cov}(\ell, f_p)}{\bar{f}_p} && (9.10)\\ &= \frac{\mathrm{Cov}(\ell, f-g)}{\bar{f}-\bar{g}} && (9.11)\\ &= \frac{\mathrm{Cov}(\ell, f)-\mathrm{Cov}(\ell, g)}{\bar{f}-\bar{g}} && (9.12)\end{aligned}$$

where we omitted t for brevity. So, absence of growth (and bloat), $E[\Delta\mu] = 0$, is obtained if

$$\mathrm{Cov}(\ell, g) = \mathrm{Cov}(\ell, f). \qquad (9.13)$$

In many conditions, this equation makes it possible to determine penalty functions g that can be used to control the program size dynamics in GP runs.

As an example, let us consider the case $g(\ell(x),t) = c(t)\ell(x)$ where $c(t)$ is a function of the generation number t. (This is essentially the traditional parsimony pressure in Eq. (9.3) but here the parsimony coefficient is allowed to change over time.) Then

$$\mathrm{Cov}(\ell, g) = c(t)\,\mathrm{Cov}(\ell,\ell) = c(t)\,\mathrm{Var}(\ell).$$

Substituting this into Eq. (9.13) and solving for $c(t)$ one finds that, in order to completely remove growth (or shrinking) from a run, one needs to set

$$c(t) = \frac{\mathrm{Cov}(\ell, f)}{\mathrm{Var}(\ell)}. \qquad (9.14)$$

Note that c is a function of t because both numerator and denominator can change from generation to generation.

Let us now consider the more general case $g(\ell(x),t) = c(t)\ell(x)^k$ where k is any real number (positive or negative). Here the no size-change condition requires

$$c(t) = \mathrm{Cov}(\ell, f)/\,\mathrm{Cov}(\ell, \ell^k). \qquad (9.15)$$

Note that when $k < 0$, instead of penalising longer individuals we give a fitness advantage to shorter individuals, which is an equally good strategy for controlling bloat as we illustrate in Sect. 9.5.

As another example, let us consider the case $g(\ell(x),t) = c(t)(\ell(x)-\mu(t))$. Here

$$\begin{aligned}\mathrm{Cov}(\ell, g) &= c(t)\,\mathrm{Cov}(\ell, \ell-\mu(t))\\ &= c(t)\,\mathrm{Cov}(\ell,\ell) - c(t)\,\mathrm{Cov}(\ell, \mu(t)).\end{aligned}$$

But, $\mathrm{Cov}(\ell, \mu(t)) = 0$ and $\mathrm{Cov}(\ell, \ell) = \mathrm{Var}(\ell)$, so we end up with Eq. (9.14) again (although the resulting penalty coefficient is then used in a different g).

What if we wanted $\mu(t)$ to follow, in expectation, a particular function $\gamma(t)$, e.g., the ramp $\gamma(t) = \mu(0) + b \times t$ or a sinusoidal function? The theory helps us in this case as well. Adding $\mu(t)$ to both sides of Eq. (9.10) we obtain:

$$\frac{\mathrm{Cov}(\ell, f) - \mathrm{Cov}(\ell, g)}{\bar{f} - \bar{g}} + \mu(t) = E[\mu(t+1)] = \gamma(t+1). \tag{9.16}$$

If g is a family of functions with a single parameter (as is true of all the functions g considered above), then we can use this constraint to solve for the free variable. For example, if we want to control bloat with parsimony terms of the form $g(\ell(x), t) = c(t)\ell(x)^k$ we can substitute this into Eq. (9.16), obtaining

$$\frac{\mathrm{Cov}(\ell, f) - c(t)\,\mathrm{Cov}(\ell, \ell^k)}{\bar{f} - c(t)E[\ell^k]} + \mu(t) = \gamma(t+1). \tag{9.17}$$

Solving for $c(t)$ gives:

$$c(t) = \frac{\mathrm{Cov}(\ell, f) - (\gamma(t+1) - \mu(t))\bar{f}}{\mathrm{Cov}(\ell, \ell^k) - (\gamma(t+1) - \mu(t))E[\ell^k]} \tag{9.18}$$

If $k = 1$, i.e., $g = c(t)\ell(x)$ (as in the standard parsimony pressure), this simplifies to

$$c(t) = \frac{\mathrm{Cov}(\ell, f) - (\gamma(t+1) - \mu(t))\bar{f}}{\mathrm{Var}(\ell) - (\gamma(t+1) - \mu(t))\mu(t)} \tag{9.19}$$

Note that, in the absence of sampling noise (i.e., for an infinite population), requiring that $E[\Delta\mu] = 0$ at each generation causes Eq. (9.13) to reduce to $\mu(t) = \mu(0)$ for all $t > 0$. However, in any finite population the parsimony pressure method can only achieve $\Delta\mu = 0$ *in expectation*, so there can be some random drift in $\mu(t)$ w.r.t. its starting value of $\mu(0)$. Experimentally we have found that this tends to be significant only for very small populations and long runs. If tighter control over the mean program size is desired, one can use Eq. (9.18) with the choice $\gamma(t) = \mu(0)$, which leads to the following formula

$$c(t) = \frac{\mathrm{Cov}(\ell, f) - (\mu(0) - \mu(t))\bar{f}}{\mathrm{Cov}(\ell, \ell^k) - (\mu(0) - \mu(t))E[\ell^k]}. \tag{9.20}$$

Note the similarities and differences between this and Eq. (9.15). In the presence of any drift moving $\mu(t)$ away from $\mu(0)$, this equation will actively strengthen the size control pressure to push the mean program size back to its initial value.[3]

[3]We talk about size control pressure rather than parsimony pressure because $\mu(t)$ can drift both above and below $\mu(0)$.

As we will see in the following section, our technique gives users almost complete control over the dynamics of the mean program size, and control can be obtained in a single generation. It is thus possible to design interesting schemes where the covariance-based bloat control is switched on or off at different times, perhaps depending on the particular conditions of a run. In the next section we will, for example, test the idea of letting the GP system run undisturbed until the mean program size reaches a threshold, at which point we start applying bloat control to prevent further growth.

Also, we note that while much of this theory assumes the use of fitness proportionate selection, Eq. (9.6) is valid in general and one could imagine selection schemes that directly penalise the selection probabilities $p(\ell,t)$ rather than fitnesses. As we will see in the experiments, however, the penalty coefficients estimated using the theory developed for fitness proportionate selection actually work very well without any modification also in systems based on other forms of selection, such as tournament selection.

Finally, we would like to make clear that while our covariant parsimony pressure technique can control the dynamics of program size, controlling program size is only one aspect of bloat. We are not explicitly addressing the causes of bloat here (these are discussed in detail elsewhere, e.g., [5, 26]): We are curing the worst symptom associated with such causes.

9.5 Experimental Results

To verify the theory in a variety of different conditions, we conducted experiments using three different GP systems—two linear register-based GP systems and one tree-based GP system [21]—and several problems. We briefly describe these systems and the problems in the next section, and then present some of our experimental results. Due to space limitations we will be able to report in detail on only a fraction of the tests we made, but the reported results generalise across a wide array of experiments.

9.5.1 GP Systems, Problems and Primitives

The first GP system we used was a linear generational GP system. It initialises the population by repeatedly creating random individuals with lengths uniformly distributed between 1 and 200 primitives. The primitives are drawn randomly and uniformly from a problem's primitive set. The system uses fitness proportionate selection and crossover applied with a rate of 100 %. Crossover creates offspring by selecting two random crossover points, one in each parent, and taking the first part of the first parent and the second part of the second w.r.t. their crossover points. We used populations of size 100, 1,000 and 10,000. In each condition we performed 100 independent runs, each lasting 500 generations.

With this linear GP system we used two artificial test problems. The first was the `Hole` problem, which simply allocates a fitness of 0.001 to programs of size smaller than 10 nodes, and a fitness of 1.0 to all other programs. This problem was used because it presents the minimal conditions for bloat to occur (according to the crossover-bias theory described in Sect. 9.2). The second problem, which we will call `Square Root`, was one where the fitness of programs was simply the square root of their size, i.e., $f(x) = \sqrt{\ell(x)}$. This problem also satisfies the conditions for bloat, but, unlike the previous one, here the entire fitness landscape is expected to favour bloat (not just sections containing the very short programs) because the correlation between length and fitness is very high for all sizes. Because of its very strong tendency to bloat, we consider this problem a good stress-test of our method. The fitness for both `Hole` and `Square Root` is determined completely by the size of the program, so any choice of primitive set produces the same results.

The second GP system we used was also linear and generational. It uses the same crossover (with the same rate) and the same form of initialisation as the first system, but initial program lengths are in the range 1–50. Runs lasted 100 generations. The system uses *tournament selection* (with tournament size 2) instead of fitness proportional selection. This allows us to test the generality of our method for controlling program size.

With this system we solved two classical symbolic regression problems. The objective was to evolve a function which fits a polynomial of the form $x + x^2 + \cdots + x^d$, where d is the degree of the polynomial, for x in the range $[-1, 1]$. In particular we considered degrees $d = 6$ and $d = 8$ and we sampled the polynomials at the 21 equally spaced points $x \in \{-1, -0.9, \ldots, 0.9, 1.0\}$. We call the resulting symbolic regression problems `Poly-6` and `Poly-8`. Polynomials of this type have been widely used as benchmark problems in the GP literature.

Fitness (to be maximised) was $1/(1 + \texttt{error})$, where `error` is the sum of the absolute differences between the target polynomial and the output produced by the program under evaluation over the 21 fitness cases. The primitive set used to solve these problems is shown in the first column of Table 9.1. The instructions refer to three registers: the input register `RIN`, which is loaded with the value of x before a fitness case is evaluated, and the two registers `R1` and `R2`, which can be used for numerical calculations. `R1` and `R2` are initialised to x and 0, respectively. The output of the program is read from `R1` at the end of its execution.

The third GP system was a classical generational tree-based GP system using binary tournament selection, with subtree crossover applied with 100 % probability. Thirty-five independent runs were done for each of five different targets for the average program size. The population size in each case was 2,000, and each run went for 500 generations. The populations were initialised using the PTC2 tree creation algorithm [20] with the initial trees having size 150.

With the tree-based GP system we used the `6-Multiplexer` problem. This is a classical Boolean function induction problem where the objective is to evolve a Boolean function with six inputs designed as `A0`, `A1`, `D0`, `D1`, `D2`, `D3`, which

Table 9.1 Primitive sets used in our experiments

Polynomial	6-MUX
R1 = RIN	AND
R2 = RIN	OR
R1 = R1 + R2	NAND
R2 = R1 + R2	NOR
R1 = R1 * R2	
R2 = R1 * R2	
Swap R1 R2	

produces as output a copy of one of the inputs D0–D3. These are known as the data lines of the multiplexer. The particular input copied over is determined by the inputs A0 and A1 (known as the address lines of the multiplexer), as follows: If A0 = 0 and A1 = 0, then Out = D0; if A0 = 1 and A1 = 0, then Out = D1; if A0 = 0 and A1 = 1, then Out = D2; if A0 = 1 and A1 = 1, then Out = D3. The function has 64 possible combinations of inputs, so we have 64 fitness cases. Fitness is the number of fitness cases a program correctly computes. The primitive set used is shown in the second column of Table 9.1.

9.5.2 Results

We start by looking at the Hole and Square Root problems. As Fig. 9.1 shows for populations of size 1,000, bloat is present in both cases, with the $\sqrt{\ell}$ fitness function bloating fiercely. Results for populations of size 100 and 10,000 are qualitatively similar.

To give a sense of the degree of control that can be achieved with our technique, Fig. 9.2 illustrates the behaviour of mean program size for the Hole and Square Root problems when five different flavours of our size control scheme are used. Results are for populations of size 1,000, but other population sizes provide similar behaviours. Note the small amount of drift present when Eq. (9.14) is used (first column). This is completely removed when instead we use Eq. (9.20) (column 5, note the different scale, and also column 3 after the transient). As columns 2 and 4 show, the user is free to impose any desired mean program size dynamics thanks to the use of Eq. (9.19).

We turn to the Poly-6 and Poly-8 problems. As Fig. 9.3 shows, bloat is present in both problems. The behaviour of mean program size is brought under complete control with our technique as shown in Figs. 9.4 and 9.5. Here we used the same targets as in Fig. 9.2 (although with slightly different parameters), but, to illustrate a further alternative, we used a parsimony term of the form $g(t) = c(t)/\ell$. This effectively promotes the shorter programs rather than penalising the longer ones.

Excellent size control was also obtained in tree-based GP when solving the *6-MUX* problem, as shown in Fig. 9.6. We used the same targets as in Fig. 9.2, but

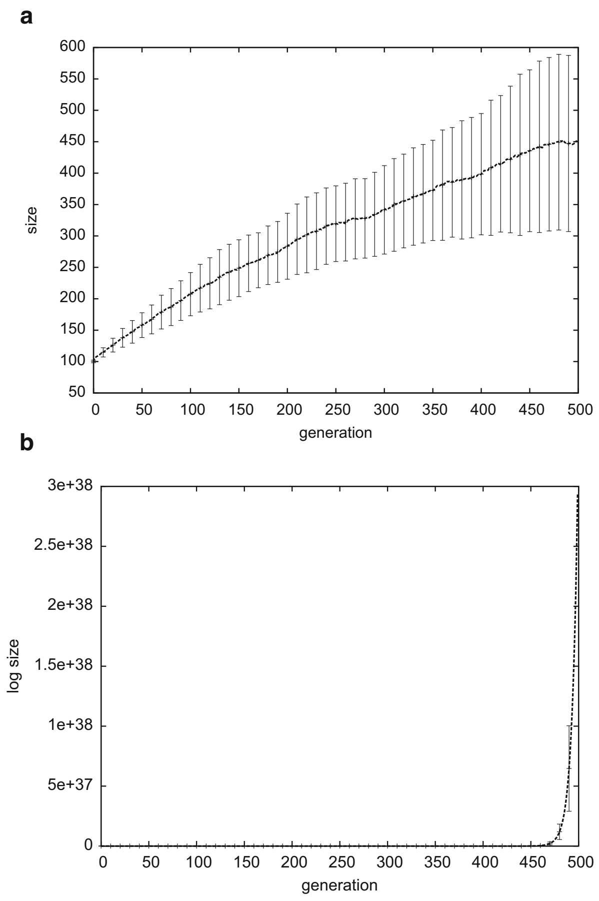


Fig. 9.1 Behaviour of the mean program size in a linear GP system when solving the `Hole` problem (**a**) and the `Square Root` problem (**b**) in the absence of bloat control for populations of size 1,000. Results are averages over 100 independent runs. The *error bars* indicate the standard deviation across the runs. Note the log scale on *plot* (**b**)

again with slightly different parameters. Here the drift that's possible when using Eq. (9.14) (the "Local" case in the figure) is quite apparent when compared to the very tight control obtained in the other configurations. Figure 9.7 shows how the average size varied in each of our runs. Only one run had average program size above 250. When one considers, however, that there is no size limit or other form of bloat control being used, having the mean sizes in that case remain below 300 for 500 generations is still a significant achievement. Much less size-variation is present if one requires $\mu(t+1) = \mu(0)$ as illustrated in Fig. 9.8. Note that in order to achieve full control over size, sometimes the penalty values $c(t)$ may have to be positive, as illustrated in Fig. 9.9.

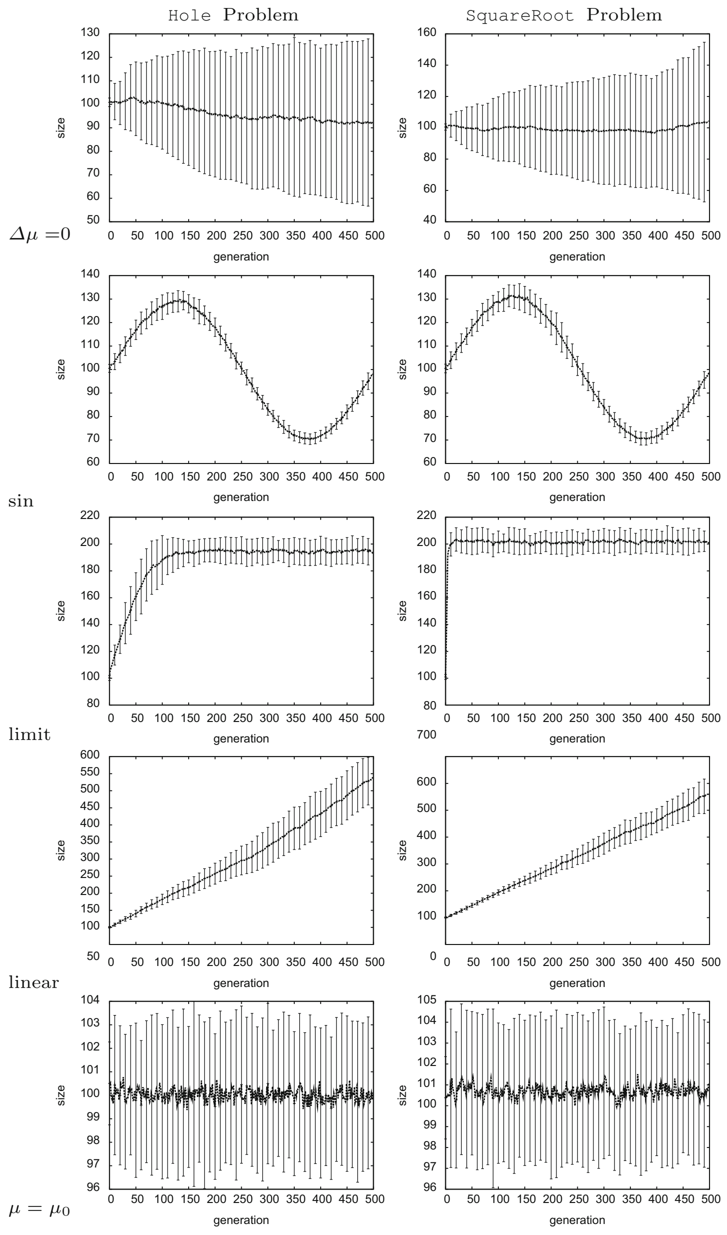
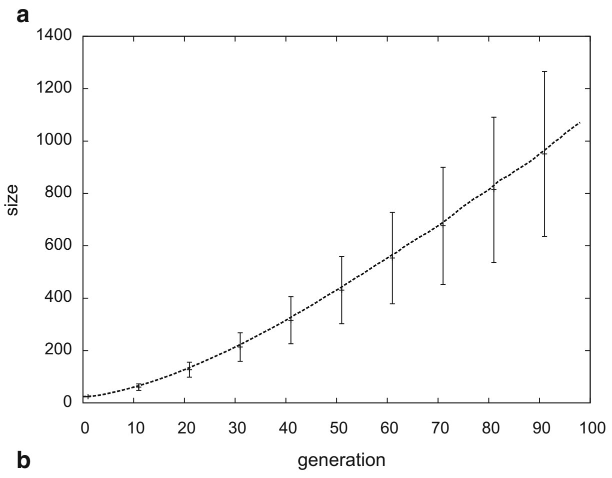


Fig. 9.2 Size control obtained in the `Hole` and `Square Root` problems with populations of size 1,000 using the penalty function $g(\ell(x),t) = c(t)(\ell(x) - \mu(t))$. $c(t)$ is computed via Eq. (9.14) so that $E[\Delta\mu] = 0$ for the plots in the *first row*. Equation (9.19) was used for the other plots so that $E[\mu(t)] = \gamma(t)$. $\gamma(t) = 30\sin(t/80) + \mu(0)$ in the *second row*, $\gamma(t) = t + \mu(0)$ in the *fourth row* and $\gamma(t) = \mu(0)$ in the *fifth row*. In the plots in the *third row* bloat control with $\gamma(t) = 200$ was activated when mean program size reached 200. Results are the average mean size over 100 independent runs. Note: The range of the y-axes vary across the plots



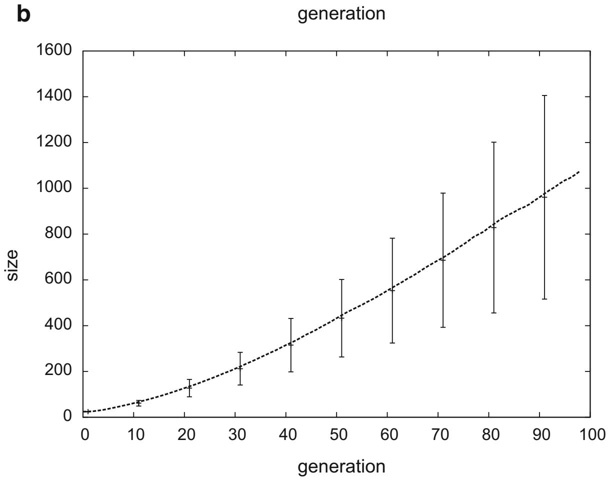


Fig. 9.3 Behaviour of the mean program size in a linear GP system when solving the `Poly-6` problem (**a**) and the `Poly-8` problem (**b**) in the absence of bloat control for populations of size 1,000. Results are averages over 100 independent runs

Performance comparisons are not the focus of this chapter. However, since virtually all bloat control methods need to balance parsimony and solution accuracy, it is reasonable to ask what sort of performance implications the use of our covariance-based bloat-control technique implies. As shown in Table 9.2 on page 200, for the two polynomial regression problems, there is generally very little performance loss associated with the use of our technique, and several of the configurations in fact increase the success rates.

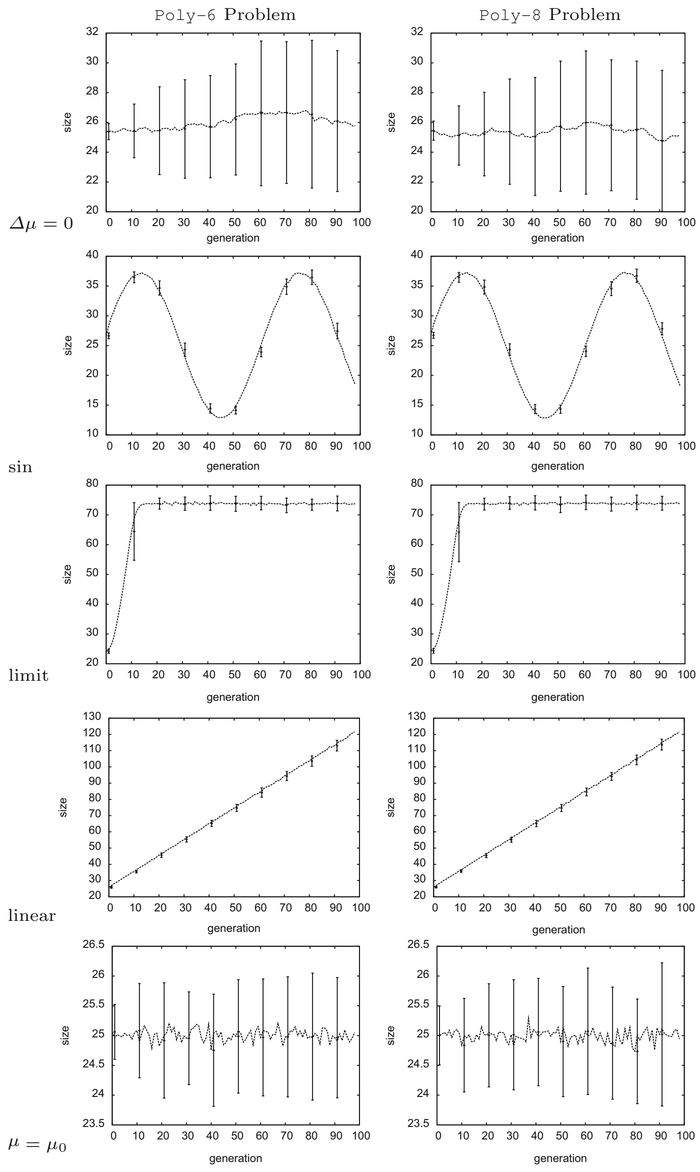


Fig. 9.4 Size control obtained in the Poly-6 and Poly-8 problems with populations of size 1,000 using the penalty function $g(\ell(x), t) = \frac{c(t)}{\ell(x)}$. $c(t)$ is computed via Eq. (9.15) with $k = -1$ so that $E[\Delta\mu] = 0$ for the plots in the *first row*. It was computed via Eq. (9.18) (with the same k) so that $E[\mu(t)] = \gamma(t)$ for the remaining plots. $\gamma(t) = 12.5 \sin(t/10) + \mu(0)$ in the *second row*, $\gamma(t) = t + \mu(0)$ in the *fourth row* and $\gamma(t) = \mu(0)$ in the *fifth row*. In the plots in the *third row* bloat control with $\gamma(t) = 75$ was activated only when mean program size reached 50. Results are the average mean size over 100 independent runs. Figure 9.5 shows results for populations of size 100. Populations size 10,000 provided qualitatively similar behaviours. Note: The range of the y-axes vary across the plots

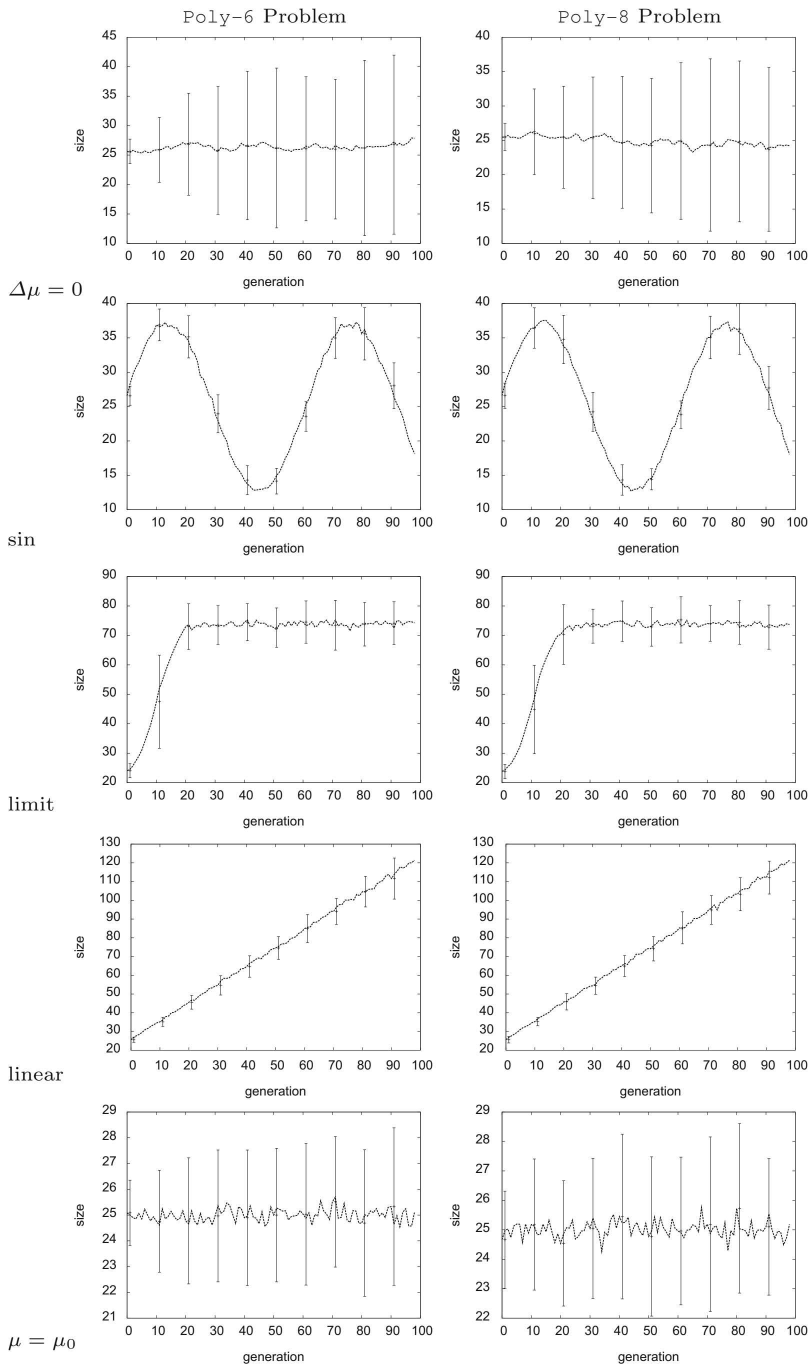


Fig. 9.5 As in Fig. 9.4 but for a population of size 100

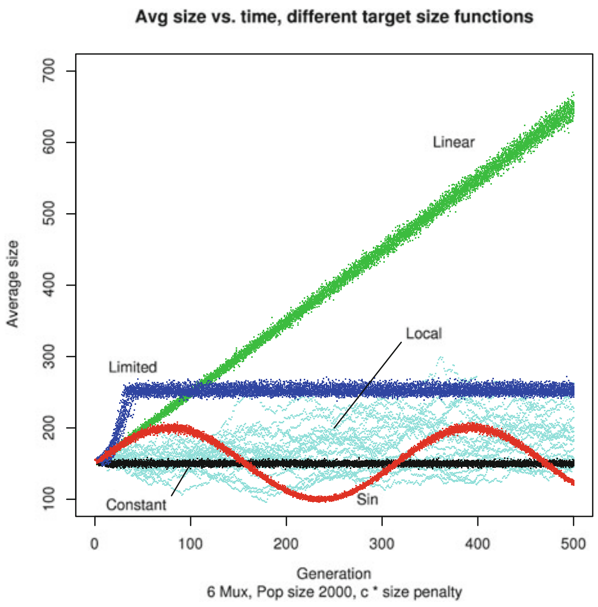


Fig. 9.6 Scatterplot of the average size over multiple runs of the 6-MUX problem with various size target functions. The population size was 2,000 and we used the penalty function $f - c\ell$. The "*Constant*" case had a constant target size of 150. "*Sin*" had the target size function $\sin((t+1)/50.0) \times 50.0 + 150$. "*Linear*" had the target function $150 + t$. "*Limited*" used no size control until the size exceeded the limit 250, after which a constant target of 250 was used. "*Local*" used a target of $\Delta\mu = 0$

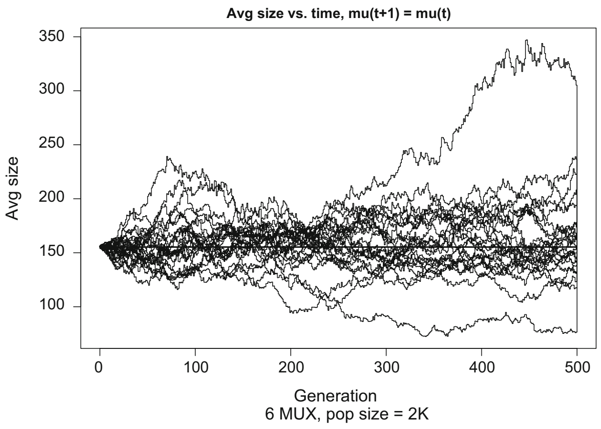


Fig. 9.7 Average size in different independent runs when using the target $\mu(t+1) = \mu(t)$ in the 6-MUX problem

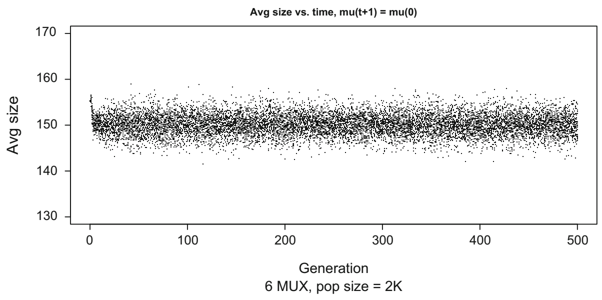


Fig. 9.8 Average size in different independent runs when using the target $\mu(t+1) = \mu(0)$ in the 6-MUX problem

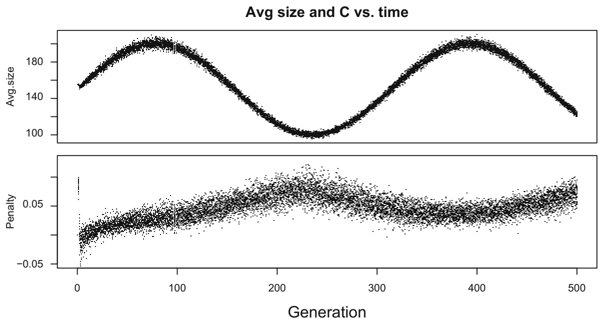


Fig. 9.9 Scatterplots of the average size over multiple runs of the 6-MUX problem with size target function $\sin((t+1)/50.0) \times 50.0 + 150$ (*top*) and of the values of penalty coefficient $c(t)$ for each independent run with the 6-MUX problem (*bottom*)

9.6 Conclusions

For many years scientists, engineers and practitioners in the GP community have used the parsimony pressure method to control bloat in genetic programming. Although more recent and sophisticated techniques exist, parsimony pressure remains the most widely known and used method.

The method suffers from two problems. Firstly, although good control of bloat can be obtained with a careful choice of the parsimony coefficient, such a choice is difficult and is often simply done by trial and error. Secondly, while it is clear that a constant parsimony coefficient can only achieve partial control over the dynamics of the average program size over time, no practical method to choose the parsimony coefficient dynamically and efficiently is available. The work presented in this chapter changes all of this.

Starting from the size evolution equation proposed in [28], we have developed a theory that tells us how to practically and optimally set the parsimony coefficient dynamically during a run so as to achieve complete control over the growth of the

Table 9.2 Success rate comparison for `Poly-6` and `Poly-8` runs with different bloat control settings

Poly	Anti-bloat		Success	Standard
degree	Target	Penalty	rate	deviation
6		**none**	**0.77**	0.04
6	$\Delta\mu = 0$	$c\ell$	0.83	0.04
6	sin	$c\ell$	0.77	0.04
6	limit	$c\ell$	0.86	0.03
6	linear	$c\ell$	0.83	0.04
6	$\mu = \mu_0$	$c\ell$	0.83	0.04
6	$\Delta\mu = 0$	$c(\ell - E[\ell])$	0.92	0.03
6	sin	$c(\ell - E[\ell])$	0.83	0.04
6	limit	$c(\ell - E[\ell])$	0.90	0.03
6	linear	$c(\ell - E[\ell])$	0.83	0.04
6	$\mu = \mu_0$	$c(\ell - E[\ell])$	0.86	0.03
6	$\Delta\mu = 0$	$c\ell^{-1}$	0.70	0.05
6	sin	$c\ell^{-1}$	0.77	0.04
6	limit	$c\ell^{-1}$	0.80	0.04
6	linear	$c\ell^{-1}$	0.79	0.04
6	$\mu = \mu_0$	$c\ell^{-1}$	0.71	0.05
6	$\Delta\mu = 0$	$c(\ell^{-1} - E[\ell^{-1}])$	0.62	0.05
6	sin	$c(\ell^{-1} - E[\ell^{-1}])$	0.41	0.05
6	limit	$c(\ell^{-1} - E[\ell^{-1}])$	0.86	0.03
6	linear	$c(\ell^{-1} - E[\ell^{-1}])$	0.45	0.05
6	$\mu = \mu_0$	$c(\ell^{-1} - E[\ell^{-1}])$	0.54	0.05
8		**none**	**0.24**	0.04
8	$\Delta\mu = 0$	$c\ell$	0.37	0.05
8	sin	$c\ell$	0.47	0.05
8	limit	$c\ell$	0.41	0.05
8	linear	$c\ell$	0.36	0.05
8	$\mu = \mu_0$	$c\ell$	0.35	0.05
8	$\Delta\mu = 0$	$c(\ell - E[\ell])$	0.30	0.05
8	sin	$c(\ell - E[\ell])$	0.41	0.05
8	limit	$c(\ell - E[\ell])$	0.33	0.05
8	linear	$c(\ell - E[\ell])$	0.33	0.05
8	$\mu = \mu_0$	$c(\ell - E[\ell])$	0.34	0.05
8	$\Delta\mu = 0$	$c\ell^{-1}$	0.26	0.04
8	sin	$c\ell^{-1}$	0.32	0.05
8	limit	$c\ell^{-1}$	0.26	0.04
8	linear	$c\ell^{-1}$	0.23	0.04
8	$\mu = \mu_0$	$c\ell^{-1}$	0.20	0.04
8	$\Delta\mu = 0$	$c(\ell^{-1} - E[\ell^{-1}])$	0.02	0.014
8	sin	$c(\ell^{-1} - E[\ell^{-1}])$	0.00	0
8	limit	$c(\ell^{-1} - E[\ell^{-1}])$	0.06	0.02
8	linear	$c(\ell^{-1} - E[\ell^{-1}])$	0.00	0
8	$\mu = \mu_0$	$c(\ell^{-1} - E[\ell^{-1}])$	0.02	0.014

programs in a population. The method is extremely general, applying to a large class of control strategies of which the classical parsimony pressure method is an instance. Experimental results with three different GP systems, using different selection strategies and five different problems all strongly confirm the effectiveness of the method.

Many instantiations of the technique presented here are possible. To a practitioner willing to try out our ideas, we would recommend to start from tuning the parsimony pressure coefficient of the traditional (linear) parsimony pressure method at every generation using our Eq. (9.15). This will do a great deal to control changes in program size. If more control is desired one could then adopt Eq. (9.20) with $k = 1$.

In future research it would be interesting to explore the applicability of Price's theorem to the control of bloat also in the case of crossover operators which are not symmetric and mutation operators. In this case Price's equation (Eq. (9.8)) would include additional terms which with a symmetric crossover evaluate to 0. It would also be interesting to explore the possibility of dynamically modulating the penalty coefficient not only as a function of size but also of the fitness distribution so as to achieve both fast progress towards high fitness values and bloat control.

References

1. L. Altenberg, The Schema Theorem, Price's Theorem, in *Foundations of Genetic Algorithms 3*, Estes Park, ed. by L.D. Whitley, M.D. Vose (Morgan Kaufmann, 1994), pp. 23–49. http://dynamics.org/~altenber/PAPERS/STPT/. Published 1995
2. P.J. Angeline, An investigation into the sensitivity of genetic programming to the frequency of leaf selection during subtree crossover, in *Genetic Programming 1996: Proceedings of the First Annual Conference*, Stanford University, ed. by J.R. Koza, D.E. Goldberg, D.B. Fogel, R.L. Riolo (MIT, 1996), pp. 21–29
3. W. Banzhaf, W.B. Langdon, Some considerations on the reason for bloat. Genet. Program. Evol. Mach. **3**(1), 81–91 (2002). doi:10.1023/A:1014548204452. http://web.cs.mun.ca/~banzhaf/papers/genp_bloat.pdf
4. R. Crawford-Marks, L. Spector, Size control via size fair genetic operators in the PushGP genetic programming system, in *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, New York, ed. by W.B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M.A. Potter, A.C. Schultz, J.F. Miller, E. Burke, N. Jonoska (Morgan Kaufmann, 2002), pp. 733–739
5. S. Dignum, R. Poli, Generalisation of the limiting distribution of program sizes in tree-based genetic programming and analysis of its effects on bloat, in *GECCO '07: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, London, vol. 2, ed. by D. Thierens, H.G. Beyer, J. Bongard, J. Branke, J.A. Clark, D. Cliff, C.B. Congdon, K. Deb, B. Doerr, T. Kovacs, S. Kumar, J.F. Miller, J. Moore, F. Neumann, M. Pelikan, R. Poli, K. Sastry, K.O. Stanley, T. Stützle, R.A. Watson, I. Wegener (ACM, 2007), pp. 1588–1595. http://www.cs.bham.ac.uk/~wbl/biblio/gecco2007/docs/p1588.pdf
6. S. Dignum, R. Poli, Crossover, sampling, bloat and the harmful effects of size limits, in *Proceedings of the 11th European Conference on Genetic Programming, EuroGP 2008*, Naples, ed. by M. O'Neill, L. Vanneschi, S. Gustafson, A.I. Esparcia Alcazar, I. De Falco, A. Della Cioppa, E. Tarantino. Lecture Notes in Computer Science, vol. 4971 (Springer, 2008), pp. 158–169. doi:10.1007/978-3-540-78671-9_14

7. A. Ekart, S.Z. Nemeth, Selection based on the Pareto nondomination criterion for controlling code growth in genetic programming. Genet. Program. Evol. Mach. **2**(1), 61–73 (2001). doi:10.1023/A:1010070616149
8. H. Iba, Complexity-based fitness evaluation for variable length representation (1997). Position paper at the Workshop on Evolutionary Computation with Variable Size Representation at ICGA-97
9. H. Iba, H. de Garis, T. Sato, Genetic programming using a minimum description length principle, in *Advances in Genetic Programming*, Chap. 12, ed. by K.E. Kinnear Jr. (MIT, 1994), pp. 265–284
10. H. Iba, H. de Garis, T. Sato, Temporal data processing using genetic programming, in *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, Pittsburgh, ed. by L. Eshelman (Morgan Kaufmann, 1995), pp. 279–286
11. K.E. Kinnear Jr., Evolving a sort: lessons in genetic programming, in *Proceedings of the 1993 International Conference on Neural Networks*, San Francisco, vol. 2 (IEEE, 1993), pp. 881–888. doi:10.1109/ICNN.1993.298674. http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/ftp.io.com/papers/kinnear.icnn93.ps.Z
12. K.E. Kinnear Jr., Fitness landscapes and difficulty in genetic programming, in *Proceedings of the 1994 IEEE World Conference on Computational Intelligence*, Orlando, vol. 1 (IEEE, 1994), pp. 142–147. doi:10.1109/ICEC.1994.350026. http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/ftp.io.com/papers/kinnear.wcci.ps.Z
13. M. Kotanchek, G. Smits, E. Vladislavleva, Pursuing the Pareto paradigm tournaments, algorithm variations & ordinal optimization, in *Genetic Programming Theory and Practice IV*, Ann Arbor, ed. by R.L. Riolo, T. Soule, B. Worzel. Genetic and Evolutionary Computation, vol. 5 (Springer, 2006), pp. 167–186
14. J.R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (MIT, Cambridge, 1992)
15. W.B. Langdon, The evolution of size in variable length representations, in *1998 IEEE International Conference on Evolutionary Computation*, Anchorage, Alaska (IEEE, 1998), pp. 633–638. doi:10.1109/ICEC.1998.700102. http://www.cs.bham.ac.uk/~wbl/ftp/papers/WBL.wcci98_bloat.pdf
16. W.B. Langdon, Size fair and homologous tree genetic programming crossovers. Genet. Program. Evol. Mach. **1**(1/2), 95–119 (2000). doi:10.1023/A:1010024515191. http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/WBL_fairxo.pdf
17. W.B. Langdon, R. Poli, Fitness causes bloat, in *Soft Computing in Engineering Design and Manufacturing*, ed. by P.K. Chawdhry, R. Roy, R.K. Pant (Springer, London, 1997), pp. 13–22. http://www.cs.bham.ac.uk/~wbl/ftp/papers/WBL.bloat_wsc2.ps.gz
18. W.B. Langdon, R. Poli, *Foundations of Genetic Programming* (Springer, 2002). http://www.cs.ucl.ac.uk/staff/W.Langdon/FOGP/
19. W.B. Langdon, T. Soule, R. Poli, J.A. Foster, The evolution of size and shape, in *Advances in Genetic Programming 3*, ed. by L. Spector, W.B. Langdon, U.M. O'Reilly, P.J. Angeline, Chap. 8 (MIT, Cambridge, 1999). pp. 163–190. http://www.cs.bham.ac.uk/~wbl/aigp3/ch08.pdf
20. S. Luke, Two fast tree-creation algorithms for genetic programming. IEEE Trans. Evol. Comput. **4**(3), 274–283 (2000). http://ieeexplore.ieee.org/iel5/4235/18897/00873237.pdf
21. N.F. McPhee, N.J. Hopper, M.L. Reierson, Sutherland: an extensible object-oriented software framework for evolutionary computation, in *Genetic Programming 1998: Proceedings of the Third Annual Conference*, University of Wisconsin, Madison, ed. by J.R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D.B. Fogel, M.H. Garzon, D.E. Goldberg, H. Iba, R. Riolo (Morgan Kaufmann, 1998), p. 241. http://www.mrs.umn.edu/~mcphee/Research/Sutherland/sutherland_gp98_announcement.ps.gz
22. N.F. McPhee, J.D. Miller, Accurate replication in genetic programming, in *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, Pittsburgh, ed. by L. Eshelman (Morgan Kaufmann, 1995), pp. 303–309. http://www.mrs.umn.edu/~mcphee/Research/Accurate_replication.ps

23. N.F. McPhee, R. Poli, Using schema theory to explore interactions of multiple operators, in *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, New York, ed. by W.B. Langdon, E. Cantú-Paz, K. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M.A. Potter, A.C. Schultz, J.F. Miller, E. Burke, N. Jonoska (Morgan Kaufmann, 2002), pp. 853–860. http://www.cs.bham.ac.uk/~wbl/biblio/gecco2002/GP139.pdf
24. R. Poli, General schema theory for genetic programming with subtree-swapping crossover, in *Genetic Programming, Proceedings of EuroGP 2001*, Lake Como. Lecture Notes in Computer Science (Springer, Milan, 2001)
25. R. Poli, A simple but theoretically-motivated method to control bloat in genetic programming, in *Genetic Programming, Proceedings of the 6th European Conference, EuroGP 2003*, Essex, ed. by C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, E. Costa. Lecture Notes in Computer Science (Springer, 2003), pp. 211–223
26. R. Poli, W.B. Langdon, S. Dignum, On the limiting distribution of program sizes in tree-based genetic programming, in *Proceedings of the 10th European Conference on Genetic Programming*, Valencia, vol. 4445, ed. by M. Ebner, M. O'Neill, A. Ekárt, L. Vanneschi, A.I. Esparcia-Alcázar. Lecture Notes in Computer Science (Springer, 2007), pp. 193–204. doi:10.1007/978-3-540-71605-1_18
27. R. Poli, W.B. Langdon, N.F. McPhee, A field guide to genetic programming. Published via http://lulu.com and freely available at http://www.gp-field-guide.org.uk (2008). (With contributions by J. R. Koza)
28. R. Poli, N.F. McPhee, General schema theory for genetic programming with subtree-swapping crossover: Part II. Evol. Comput. **11**(2), 169–206 (2003). doi:10.1162/106365603766646825. http://cswww.essex.ac.uk/staff/rpoli/papers/ecj2003partII.pdf
29. G.R. Price, Selection and covariance. Nature **227, August 1**, 520–521 (1970). http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/price_nature.pdf
30. J. Rosca, Generality versus size in genetic programming, in *Genetic Programming 1996: Proceedings of the First Annual Conference*, Stanford University, ed. by J.R. Koza, D.E. Goldberg, D.B. Fogel, R.L. Riolo (MIT, 1996), pp. 381–387. ftp://ftp.cs.rochester.edu/pub/u/rosca/gp/96.gp.ps.gz
31. J. Rosca, A probabilistic model of size drift, in *Genetic Programming Theory and Practice*, Chap. 8, ed. by R.L. Riolo, B. Worzel (Springer, New York, 2003), pp. 119–136
32. J.P. Rosca, Analysis of complexity drift in genetic programming, in *Genetic Programming 1997: Proceedings of the Second Annual Conference*, Stanford University, ed. by J.R. Koza, K. Deb, M. Dorigo, D.B. Fogel, M. Garzon, H. Iba, R.L. Riolo (Morgan Kaufmann, 1997), pp. 286–294. ftp://ftp.cs.rochester.edu/pub/u/rosca/gp/97.gp.ps.gz
33. J.P. Rosca, D.H. Ballard, Complexity drift in evolutionary computation with tree representations. Technical Report NRL5, University of Rochester, Computer Science Department, Rochester, 1996. ftp://ftp.cs.rochester.edu/pub/u/rosca/gp/96.drift.ps.gz
34. J.P. Rosca, D.H. Ballard, Rooted-tree schemata in genetic programming, in *Advances in Genetic Programming 3*, Chap. 11, ed. by L. Spector, W.B. Langdon, U.M. O'Reilly, P.J. Angeline (MIT, Cambridge, 1999), pp. 243–271. http://www.cs.bham.ac.uk/~wbl/aigp3/ch11.pdf
35. J.E. Rowe, N.F. McPhee, The effects of crossover and mutation operators on variable length linear structures, in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, San Francisco, ed. by L. Spector, E.D. Goodman, A. Wu, W.B. Langdon, H.M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M.H. Garzon, E. Burke (Morgan Kaufmann, San Francisco, 2001), pp. 535–542. http://www.cs.bham.ac.uk/~wbl/biblio/gecco2001/d03b.pdf
36. T. Soule, J.A. Foster, Effects of code growth and parsimony pressure on populations in genetic programming. Evol. Comput. **6**(4), 293–309 (1998). doi:10.1162/evco.1998.6.4.293

37. T. Soule, J.A. Foster, Removal bias: a new cause of code growth in tree based evolutionary programming, in *1998 IEEE International Conference on Evolutionary Computation*, Anchorage, Alaska (IEEE, 1998), pp. 781–186
38. B.T. Zhang, H. Mühlenbein, Balancing accuracy and parsimony in genetic programming. Evol. Comput. **3**(1), 17–38 (1995). doi:10.1162/evco.1995.3.1.17

Chapter 10
Experimental Analysis of Optimization Algorithms: Tuning and Beyond

Thomas Bartz-Beielstein and Mike Preuss

Abstract This chapter comprises the essence of several years of tutorials the authors gave on experimental research in evolutionary computation. We highlight the renaissance of experimental techniques also in other fields to especially focus on the specific conditions of experimental research in computer science, or more concretely, metaheuristic optimization. The experimental setup is discussed together with the pitfalls awaiting the unexperienced (and sometimes even the experienced). We present a severity criterion as a meta statistical concept for evaluating statistical inferences, which can be used to avoid fallacies, i.e., misconceptions resulting from incorrect reasoning in argumentation caused by floor or ceiling effects. The sequential parameter optimization is discussed as a meta statistical framework which integrates concepts such as severity. Parameter tuning is considered as a relatively new tool in method design and analysis, and it leads to the question of adaptability of optimization algorithms. Another branch of experimentation aims at attaining more concrete problem knowledge, we may term it "exploratory landscape analysis", containing sample and visualization techniques that are often applied but not seen as being a methodological contribution. However, this chapter is not only a renarration of well-known facts. We also attempt to look into the future to estimate what the hot topics of methodological research will be in the coming years and what changes we may expect for the whole community.

T. Bartz-Beielstein (✉)
Faculty of Computer Science and Engineering Science, Institute of Computer Science, Cologne University of Applied Sciences, Cologne, Germany
e-mail: thomas.bartz-beielstein@fh-koeln.de

M. Preuss
Algorithm Engineering, Department of Computer Science, TU Dortmund, Germany
e-mail: mike.preuss@tu-dortmund.de

10.1 Introduction

As in many natural sciences, research on metaheuristics and especially *evolutionary computation* (EC) mainly rests on two pillars: theory and practice. Undoubtedly, theory in EC has made good forward progress during the last decade. However, the larger part of published work in this area is still dealing almost exclusively with the application of EC and related methods to real-world and/or benchmark problems. Qualms regarding the meaningfulness of theoretical approaches are rarely expressed, but doubts concerning the reliability of experimental results are often raised, especially by practitioners. This may lead to the question: “Can we get rid of experimentation in EC as a whole and resort to theory only?”

Our experience is that this will not happen, because there are simply too many different real-world applications of EC techniques. Moreover, theory and practice have different rhythms, and one may design and implement a useful algorithm modification in minutes or hours, but adapting the existing theory to it may take days or weeks. It may be worth noting that in other related sciences and in philosophy of science, experimentation is currently experiencing a renaissance [54].

If we presume that experimentation is necessary, we need to ponder how to strengthen the experimental methodology in order to make experimental results more reliable and thus also more useful for theoretical approaches. It may help to make clear what experimental works in EC are actually about and if they can be split into categories. It seems that during the last decades, two motivations for experimental works have been predominant:

- Solving a real-world problem, or at least showing that it could be solved by some EC-based method
- Demonstrating the ability of a (preferably new and self-defined) algorithm

These two form the extremes, and mixtures with various weights are frequently encountered. They resemble a problem-centric and an algorithm-centric view, respectively. The former strongly leans towards engineering and often focuses on representations, simulating, modeling, long runtimes, and technical issues, whereas the latter is much nearer to algorithmics and mathematics. One deals with constructed problems that can be computed fast and for which the most important properties are well known.

Setting up an experiment can be far from trivial as there are numerous mistakes that may render an experiment useless. Rardin and Uzsoy [71] state the following: “No one who has ever tried it thinks conducting empirical research on heuristics is easy”, and we would like to add that this stems from the complexity of the systems with which heuristics deals with. Many influences which are simply ignored (or removed “by construction”) in theoretical investigations cannot be removed but must rather be controlled in experimental studies, thereby at least trying to avoid the most obvious mistakes.

Performing experiments in computer science can address the following (related) tasks:

T-1 Find the best parameters or algorithms given k sets of random numbers representing the outcome of some experiments.

T-2 Find the best assignment for a set of real variables representing the parameters of the algorithm (within a given range) for a problem class.

T-3 Given m possible ways to modify algorithm A (e.g., by using extra operators) find the best combination for a problem class.

Regarding task T-1, we will restrict our analysis to problems with $k = 2$. We are using SPOT (introduced in Sect. 10.3.2) to find the best assignment. SPOT can also be used to determine the best combination of algorithm operators for a given problem class.

Although conceptually different, task T-3 can be tackled in the framework of task T-2. In a complex experiment the experimenter might have to consider, hierarchically, all three tasks (for example he might want to retune the parameters for every different combination of operators in T-3). The tuning procedure can be performed hierarchically, e.g., for every different combination of operators. However, we recommend an alternative approach which includes the operators into the algorithm design. Settings for the operators can be included as factors in the algorithm design and treated in a similar way as numerical parameters; see Chap. 14 in [18] for an example.

We report on these methodological foundations in Sect. 10.2, also reflecting on approaches in other fields, especially in algorithmics. This section also describes how reports from experimental studies can be structured.

Section 10.3 introduces the framework of *active experimentation*. It describes the sequential parameter optimization toolbox. Since the comparison of results plays a central role in experimentation, we discuss key elements from a metastatistical perspective.

The algorithm-centric research has made good progress during the last years; two notable developments are different tuning techniques (e.g., *F-Race* [22], *sequential parameter optimization* (SPO) [1, 17] and the *relevance and calibration* method (REVAC) [61]) and new benchmark competitions/libraries as BBOB'09 [36] and the CEC'05 competition [75]. We highlight SPO as one interesting tuning approach and its use for the experimental analysis of a *simulated annealing* (SANN) heuristic, which is presented in Sect. 10.4. Hypothesis testing is discussed in Sect. 10.5. We present an introduction and discuss problems related to hypothesis testing as well.

Some researchers claim that scientific progress is based on accepting high-level theories, whereas others view progress "based on the growth of more localized experimental knowledge"[54]. Actually, there is an interesting debate about the importance of experimentation in the philosophy of science, e.g., [24] can be recommended as a good starting point. We will not detail this discussion in this chapter, but will transfer some important results from this on-going debate in the following. The focus of our work lies on severity as a metastatistical concept for evaluating statistical inferences, which can be used to avoid fallacies, i.e., misconceptions resulting from incorrect reasoning in argumentation caused by floor or ceiling effects. Severity, as an extension of the power used in hypothesis testing, is introduced in Sect. 10.6.

Based on the considerations from the previous sections, meta statistical principles can be applied. Metastatistical rules, as discussed in Sect. 10.7, are necessary, because statistical results can be misleading and need some interpretation. Here the concept of severity comes into play, which is one element of the sequential parameter optimization.

For the problem-centric approach, it is often most important to collect problem knowledge during design and test of an optimization method. This resembles some kind of exploratory analysis and is dealt with in Sect. 10.8. Finally, the chapter concludes with a summary and a view onto envisaged future developments in Sect. 10.9.

10.2 Towards an Experimental Methodology

The following sections provide basic elements and fundamental considerations, which are necessary for active experimentation in computer science. *Active experimentation*, which implements a framework of the approach presented in this section, is visualized in Fig. 10.1.

Section 10.2.1 discusses the role of experiments in computer science. Generally, experiments should be based on well-founded research questions. We present research questions related to demonstration, robustness, comparison, understanding, and novelty detection in Sect. 10.2.2. The key problem which occurs in nearly every experimental study is the selection of an appropriate measure. The most prominent measures are introduced in Sect. 10.2.3. After an adequate measure is selected, the pre-experimental planning phase can begin. During this phase, which is described in Sect. 10.2.4, the experimental setup is calibrated. Problems, which are related to parameter settings, are briefly discussed in Sect. 10.2.5. Then, experiments can be performed, see Sect. 10.2.6. These experiments can generate lots of data, which are used for the experimental analysis. We consider key features of comparisons in Sect. 10.2.7. Findings from these analyses should be presented, e.g., as articles. We propose a structured report scheme in Sect. 10.2.8. We can conclude from our experience that meaningful results require several experiments as discussed in Sect. 10.2.9. Finally, we consider the determination of scientifically meaningful results in Sect. 10.2.10.

10.2.1 Performing Experiments in Computer Science

In theoretical computer science, one is usually interested in pessimistic generalizations, in knowing what an algorithm does in the worst possible case. Experimental

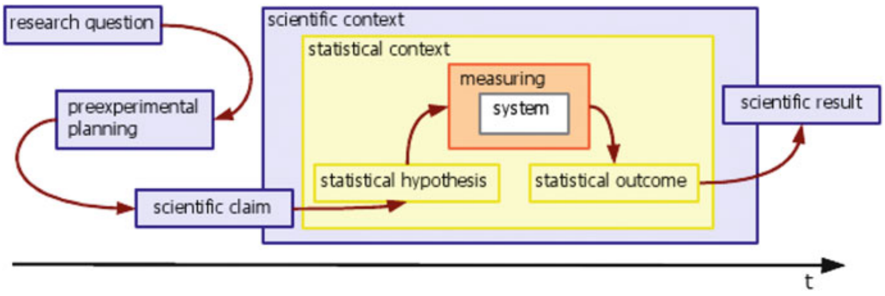


Fig. 10.1 Steps and contexts of performing an experiment from research question to scientific result

results are considered with a certain amount of skepticism. This may have two reasons:

- Many experimental works of the past are not very well crafted. Unlike from other sciences with very expensive (in time and financial effort) experiments, computer science is in the luxurious position of allowing for nearly unlimited sampling. Thus, one should not stop at the first results but instead use the obtained knowledge to refine the experiment. Every possible setup is like a simple point in a high-dimensional space, defined by the many different factors that could influence the outcome. Sampling just at one point will not allow for any conclusions about, e.g., the overall quality of certain algorithms.
- Experimental investigations rarely care about worst cases. If the treated problem has a real-world background, the worst case view is usually not applicable: One simply does not know how it could look, and as no analytic formulation is available, there is no alternative way of approaching the worst case. One therefore often follows an average case approach, where the average consists of some reasonably likely cases or representatives thereof. Thus, experiments can only lead to probabilistic reasoning because it is not possible to give a conclusion that holds for every member of a problem class. This approach generalizes neutrally over the samples made, and even if the experiment is bug-free and cleverly set up, exact knowledge is only attained for the points that are actually measured. Performance guarantees cannot be given for any deviating setting.

This said, we do not recommend to cease experimentation altogether. In many situations, it is the only way to advance scientific progress. In others, it can be a valuable addition to theory, as emphasized by the *Algorithm Engineering* approach [26]. However, it is necessary to be aware of the problems one may run into and to follow a suitable experimental methodology instead of doing ad-hoc tests.

Several authors from different fields have cautioned against experimental mistakes and provided guidelines for scientifically meaningful experimentation on algorithms; we name only a small subset here. Moret gives a methodology overview from the viewpoint of algorithmics, also reporting about testing heuristics [59].

Johnson [45] provides a comprehensive list of pitfalls and solutions for experiments in algorithmics, mostly dealing with deterministic algorithms. However, a large part of the list also applies to metaheuristics/evolutionary algorithms. Nevertheless, there are problems stemming from the nondeterministic nature of these algorithms. These are especially treated, e.g., by Hooker [40] and Eiben [29]. In the following, we describe how to start an experimental investigation on metaheuristic optimization methods and how to avoid the most commonly made mistakes.

10.2.2 Research Questions

Around two decades ago, Cohen [27] hinted at the fact that in artificial intelligence, experimental studies were often not well tasked. In the evolutionary computation or metaheuristics fields, the situation at that time was certainly not better. Surprisingly, nowadays many experimental works still come without a clear statement about the overall *research question* that is going to be answered. Instead, the implicitly assumed task often is to show that any new algorithm A is *better* than a standard method A^* or many of them. Sampling comparison data at a small number of points (defined by algorithms, optimization problems, parameters, termination criteria, etc.) does not necessarily allow for general conclusions about the compared algorithms. Besides, the tackled research question should not be that general. As an example, it may make more sense to ask under which conditions (problems, runtimes, etc.) A is better than A^* and why. It goes without saying that the research question must be stated in the paper, so that the reader gets a chance to value the experimental findings.

In our view, the following questions or aspects of questions are fundamental for experiments in computer science and may serve as a guideline for setting up new experiments. The experimenter should clearly state if experiments are performed to:

1. Demonstrate the performance of one algorithm,
2. Verify the robustness of one algorithm on several problem instances,
3. Compare two (or several) known algorithms,
4. Explain and understand an observed behavior, or
5. Detect something new.

Each of these five research goals, which can also be characterized as demonstration, robustness, comparison, understanding, and novelty detection, requires a different experimental setup.

We discourage mixing too many different aspects of an experimental investigation into one experiment. Rather, one shall consider if it makes sense to split the investigation into multiple experiments, each one reported separately. This simplifies understanding the outcome and also enhances *reproducibility*, which should be a primary concern when presenting exciting new facts obtained by experiment. If we want to employ statistical techniques as hypothesis tests to bolster up our confidence in the findings, it is necessary to switch the context from a domain-specific scientific one to a statistical one and back. We can first formulate one or several scientific claims. As an example, we consider the claim: "Algorithm A is faster than algorithm A^* under the defined conditions (test problems, performance measure, parameter settings, etc.)." These then have to be formulated as statistical hypotheses, which can be tested by experimentation.

10.2.3 What to Measure?

Once the direction of the experimental investigation is set, one has to decide how to measure. This may not be a trivial issue. McGeoch [55] demonstrates that even for deterministic algorithms, the influence of the measure should not be underestimated and that setting it up properly can be a decisive factor for obtaining interesting results. When investigating nondeterministic optimization algorithms on not too simple problems, there are two principal possibilities. We can employ a quality task and measure how long the algorithm needs to get there, or we can set up a budget (usually regarded as equivalent to runtime in black-box optimization) task and measure the performance obtained under scarce resources. As discussed in the BBOB'09 setup [36], fixing a quality task (there also called *horizontal* measuring) often leads to a better understanding of algorithm performance than fixing a budget task (*vertical* measuring). The competition was thus run under the *expected running time* (ERT) measure:

$$ERT(f_{target}) = \frac{\#FEs(f_{best}(FE) \geq f_{target})}{\#succ} \tag{10.1}$$

The number of "unsuccessful evaluations" (where the observed objective function value is worse than a given target, $f_{best}(FE) \geq f_{target}$, restarts are assumed) per repeat is summed up and divided by the number of successes ($\#succ$) of attaining the target function value f_{target}. The ERT is certainly good at capturing the performance of algorithms under relaxed resource conditions. However, real-world problems often do not allow such generous conditions so that only a few hundred or thousand function evaluations can be invested. It is a philosophical question if one can justify applying the term "optimization" in this context, but apart from that it is obvious that an algorithm with a good final best function value does not necessarily provide a good approximation of this value fast. Thus it also makes sense to set a budget task in the predicted range of allowed function evaluations under application conditions and ask which algorithm provides the best solution and how reliable it is.

Next to the ERT, some often-used measures are the *mean best fitness* (MBF) and the *success rate* (SR). However, these come with some difficulties. Averaging is very sensitive regarding outliers, so it may be more suitable to work with the median instead of the mean. Success rates were frequently used in the past, but this measure removes the whole runtime information. A fast method always reaching 100 %

becomes indistinguishable from a slow one also reaching this success rate. For setting up proper conditions for measuring, one may rely on runtime distributions as proposed by Hoos and Stützle [41]. Chapter 7 in [1] presents 18 different performance measures and mentions relevant publications.

Sometimes, however, the available base measures do not match the intention of the experiment well. For example, if one has a certain, possibly unusual tradeoff between quality and runtime in mind. In these cases, it may be necessary to define a new measure as, e.g., suggested by Rardin and Uzsoy [71] (see [67] for an example). However, this path should be walked with caution: It does not make sense to stick to a measure that does not express what one actually wants to investigate. But too many measures render results incomparable.

10.2.4 Pre-experimental Planning

If research question and measure are chosen, and the implementation issues have been resolved, it is time for the first tests. We name this phase *pre-experimental planning* and its main purpose is to check if the envisioned experimental setup is meaningful. This may apply to the selection of f_{target} values for measuring, or the maximum allowed number of function evaluations, or the set of benchmark problems one is going to investigate.

During the pre-experimental planning phase, several practical problems have to be considered, e.g., how many comparisons should be performed? How many repeat runs of each algorithm should be done? Should a one-stage or multistage procedure be used? Classical textbooks on statistics provide useful answers to these questions, which are related to *pre-data* information, i.e., before the experimental data is available. We highly recommend Bechhofer et al.'s comprehensive work "*Design and Analysis of Experiments for Statistical Selection, Screening, and Multiple Comparisons*" [20].

Additionally, one should try to make sure that other possible problems that could influence the outcome of an experiment are found and remedied. As Knight [47] argues, one should apply common sense concerning setup and results. Are the first results plausible? Or do they hint to a possible code problem? What makes a meaningful difference in my test cases? We would explicitly suggest to use as much visualization as possible at this stage. A problem revealed only after the experiment is finished is much more annoying than one found early, especially if time constraints do not allow for redoing the whole experiment.

At the end of the pre-experimental phase, one should be able to set up an experiment in a way that it leads to results that address the given research question. We may assume that not all possible problems with the setup can be identified during this first phase, but it serves as a filter preventing the most obvious mistakes, some of which we highlight in the following.

10.2.5 Fair Parameter Settings

There are different opinions on how much effort should be put into obtaining good parameters for the algorithms which are to be compared. It certainly makes sense to compare algorithms under default parameters. This resembles an application setting where parameter adaptation to a problem is not possible, e.g., due to time restrictions. On the other hand, it may also be important to know how the algorithm would perform under good parameter settings. In this case, tuning algorithms can be applied before running the comparison. In either case, the comparison should be fair, meaning that the same amount of tuning should go into setting up each algorithm. It is clear that a tuned algorithm will perform better than an algorithm running under default parameters in most cases; this is hardly worth an experimental investigation. For newly invented algorithms, a robust set of default parameters may not be available, thus the ad hoc parameter values chosen to make it run can serve as such. However, it is even more necessary to explore parameter effects and interactions in this case, e.g., by applying tuning. Regardless of the statements above, it may be a viable research question to ask if *any* parameter setting for a new algorithm leads to a performance advantage over a standard algorithm. Then the next question should be: How robust are these parameters, or is the advantage only achieved for very specific problems, and if so, for which ones?

10.2.6 Performing the Experiments

Now that the research question and the performance measure for a given problem and algorithm are fixed, experiments can be performed. It is important to set up the scientific claims and their matching statistical hypotheses *before* looking at the obtained results [59] to achieve as much objectivity as possible. Otherwise, one could set up hypotheses in a way so that they are always supported, which renders the scientific contribution insignificant. On the other hand, this requires a change in how experimental results are received. More than the raw numbers or outcomes of statistical tests, the knowledge gain is important. Do the results contain previously unknown facts? This may also happen if two well-known algorithms, neither of which is new, are compared on a new set of benchmark functions; it is not necessary to always invent new algorithms. Or do they support or even refute known facts?

Additionally, we would like to give a practical hint here: “Never watch a running experiment.” Once one can be sure that the experiment is indeed going to produce useful data, one should wait with the analysis until the experiment is finished. The reason is simply that one is easily mislead by the first results coming in and may get a wrong impression that is later on hard to get rid of, even in the face of the full data set. In this case the experimenter is in danger of losing the necessary neutrality.

10.2.7 Key Features of Comparisons

Many experimental studies are based on comparisons. Consider, e.g., tuning procedures which can be used to improve algorithm's performance. Obviously, each tuning procedure itself requires comparisons, because the performance of the algorithm before, say

$$x = \mathrm{perf}(A), \tag{10.2}$$

and after the tuning, say $x^* = \mathrm{perf}(A^*)$ has to be compared. Many tuning procedures are based on stochastic data, i.e., noisy data. This noise can be caused by:

1. The algorithm, e.g., evolutionary algorithms,
2. The problem, e.g., simulation model,
3. Or both.

Therefore, the comparison of two real values x and x^* is not sufficient and multiple runs of the algorithm have to be performed. We are considering (at least) two data vectors: $\mathbf{x}$ and $\mathbf{x}^*$, where $\mathbf{x}$ denotes the vector of n performance values of the untuned algorithm A, and $\mathbf{x}^*$ the vector of m runs of the tuned algorithm A^*. Note, a similar situation might occur even if algorithm and problem are purely deterministic when multiple problem instances are tested.

In many cases we are facing the following fundamental problem after all the experiments were performed, i.e., post-data: Given two data sets, $\mathbf{x}$ and $\mathbf{x}^*$, representing data from associated algorithms A and A^*, respectively. Decide whether A is better than A^*.

In order to answer this question, performance has to be measured. Although simple statistics such as the mean or median of the run time are adequate to gain a first impression of the algorithm's performance, a sound statistical analysis requires more sophisticated measures. At this stage, statistical tests can be are carried out.

10.2.8 Reporting Results

After the results are in, they should be visualized to enable a basic consistency check. Figures are much easier to interpret than tables, so this effort is not wasted and greatly helps when looking for interesting effects. When conducting the experiment as well as when creating a structured report of it, it may be helpful to work alongside the eight-step procedure presented in Fig. 10.2 and to write down decisions, setups and results as they are obtained during the experiment. This structure is largely similar to the one often applied in natural sciences for many decades.

Note that we separate observations from discussion. This may seem artificial and the distinction is not in all cases obvious. However, the intention is to keep

R-1: Research question. State in short the general objective of the experiment.
R-2: Pre-experimental planning. Report the results of first tests which are important for setting up the main experiment, e.g., for choosing problems, parameter settings, termination criteria.
R-3: Task. Formulate one or several scientific claims (only applicable if more concrete than the research question) and give matching statistical hypotheses, together with significance conditions.
R-4: Setup. Here goes everything that is needed to replicate the experiment, if not previously described. This consists of the applied algorithms, test problems, parameter settings, important outer conditions (e.g., if relevant, details of the employed hardware). Now, experiments can be performed.
R-5: Results/Visualization. This holds numerical results or links to the tables or figures made of them and also reports on the outcome of the hypothesis tests.
R-6: Observations. Unexpected or otherwise notable behavior that has been detected by reviewing the results, *without* interpretation.
R-7: Discussion of the statistical relevance. Statistical hypotheses from step R-3 are reconsidered (accepted/rejected).
R-8: Discussion of the scientific meaning. Attempts to give explanations for the results/observations obtained and puts the results of the experiment in a context with other scientific findings. This paragraph is meant to contain subjective statements which might lead to new hypotheses or research questions based on the results from current experiments.

Fig. 10.2 Structured report

objective differences apart from their interpretation. If, for example, an algorithm is surprisingly good on specific problem instances, this is surely an observation. Giving a presumed reason why this is the case belongs to the discussion, as another author may come up with another explanation even if the observation can be replicated.

10.2.9 Iterating the Experimental Process

As already stated by McGeoch [55] and others, experimentation with algorithms should not be limited to a one-shot event but rather should be regarded as an iterated process where the results obtained from the first experiment lead to new hypotheses and a refined experimental setup for the next. For example, it may happen that the first experiment revealed some unexpected facts and one has an intuition concerning the causing factors, which can be tested in a second experiment.

In order to support the incremental experimentation process, we recommend keeping an *experimental journal* of all experiments undertaken in a specific context. The journal should contain at least a list of running numbers, time stamps, names/research questions, and a short description of the outcome. The journal can be helpful for obtaining an overview of the progress of an investigation and keeping the data well organized.

10.2.10 Scientifically Meaningful Results?

Finally, after performing all these tests as described in Sect. 10.2.7, one fundamental qualm remains: "How can we guarantee that results are scientifically meaningful?" This question is related to *post-data* information—it includes data which is available after the experiments were performed. We will focus on this question in the following by introducing the concept of severity. A technical treatment of the concept of severity is given in Sect. 10.6.

In the severe testing philosophy, the quantitative assessment offered by error statistics provides tools to test how well-probed hypotheses are. Mayo [53] introduces the concept of severity as follows: "Stated simply, *a passing result is a severe test of hypothesis H just to the extent that it is very improbable for such a passing result to occur, were H false.*"

Although this approach is based on classical hypothesis testing, i.e., the Neyman–Pearson statistical paradigm, it is relevant to different statistical frameworks, e.g., non parametric approaches. Classical hypotheses testing dominates today's scientific publications, therefore this first approach is justified by everyday practice.

10.3 Active Experimentation

Section 10.3.1 presents *active experimentation* as a framework which implements features of the experimental methodology introduced in Sect. 10.2. This framework can be used for demonstration, robustness, comparison, understanding, and novelty detection. In Sect. 10.3.2 the sequential parameter optimization toolbox is introduced. It comprehends the computational steps of the active experimentation framework, i.e., design of experiments, response surface methods, or statistical analysis and visualization. Automated experimentation has gained much attention in the last years. Several automated approaches were proposed, especially in the context of demonstration and robustness, e.g., automated tuning of algorithms. Therefore, we will compare automated and interactive approaches in Sect. 10.3.3.

10.3.1 Definition

Definition 10.1 (Active Experimentation). *Active experimentation* (AEX) is a framework for tuning and understanding of algorithms. AEX employs methods from error statistics to obtain reliable results. It comprises the following elements:

AEX-1:	Scientific questions	AEX-3:	Experiments
AEX-2:	Statistical hypotheses	AEX-4:	Scientific meaning

□

These elements can be explained as follows. The starting point of the investigation is a scientific question (AEX-1). This question often deals with assumptions about algorithms, e.g., influence of parameter values or new operators. This (complex) question is broken down into (simple) statistical hypotheses (AEX-2) for testing. Next, experiments can be performed for each hypothesis:

(a) Select a model, e.g., a linear regression model to describe a functional relationship.
(b) Select an experimental design.
(c) Generate data, i.e., perform experiments.
(d) Refine the model until the hypothesis can be accepted/rejected.

Finally, to assess the scientific meaning of the results from an experiment, conclusions are drawn from the hypotheses. This is step AEX-4 in the active experimentation framework. Here, the concept of severity as introduced in Sect. 10.6 comes into play.

10.3.2 Sequential Parameter Optimization Toolbox

We introduce the *sequential parameter optimization toolbox* (SPOT) as one possible implementation of the experimental approach in the AEX framework. The SPO *toolbox* was developed over recent years by Thomas Bartz-Beielstein, Christian Lasarczyk, and Mike Preuß [17]. Main goals of SPOT are (i) to determine improved parameter settings for optimization algorithms and (ii) to provide statistical tools for analyzing and understanding their performance.

Definition 10.2 (Sequential Parameter Optimization Toolbox). The sequential parameter optimization toolbox (SPOT) implements the following features, which are related to step AEX-3.

SPOT-1: Use the available budget (e.g., simulator runs, number of function evaluations) sequentially, i.e., use information from the exploration of the search space to guide the search by building one or several metamodels. Choose new design points based on predictions from the metamodel(s). Refine the metamodel(s) stepwise to improve knowledge about the search space.
SPOT-2: If necessary, try to cope with noise (see Sect. 10.2.7) by improving confidence. Guarantee comparable confidence for search points.
SPOT-3: Collect information to learn from this tuning process, e.g., apply exploratory data analysis.
SPOT-4: Provide mechanisms both for interactive and automated tuning. □

The article “Sequential Parameter Optimization” [17] was the first attempt to summarize results from tutorials and make this approach known to and available for a broader audience. Since 2004, a series of tutorials was presented during the leading conferences in the field of computational intelligence, e.g., [6, 7, 9–12, 14, 15].

SPOT was successfully applied in the fields of bioinformatics [32, 33, 79], environmental engineering [30, 48], shipbuilding [72], fuzzy logic [82], multimodal optimization [68], statistical analysis of algorithms [50, 78], multicriteria optimization [80], genetic programming [51], particle swarm optimization [16, 49], automated and manual parameter tuning [31, 42, 43, 74], graph drawing [65, 77], aerospace and shipbuilding industry [63], mechanical engineering [56], and chemical engineering [39]. Bartz-Beielstein [3] collects publications related to sequential parameter optimization.

SPOT employs a sequentially improved model to estimate the relationship between algorithm input variables and its output. This serves two primary goals. One is to enable determination of good parameter settings; thus SPOT may be used as a tuner. Second, variable interactions can be revealed that help to understand how the tested algorithm works when confronted with a specific problem or how changes in the problem influence the algorithm's performance. Concerning the model, SPOT allows the insertion of virtually every available metamodel. However, regression and Kriging models, or a combination thereof, are most frequently used as prediction models (as defined as F in Algorithm 10.1). Bartz-Beielstein [4, 5] describes integration and use of these prediction models in SPOT.

Algorithm 10.1 presents a formal description of the SPOT scheme. This scheme consists of two phases, namely the first construction of the model (lines 1–6) and its sequential improvement (lines 8–27). Phase 1 determines a population of initial designs in algorithm parameter space and runs the algorithm k times for each design. Phase 2 consists of a loop with the following components:

1. Update the model by means of the obtained data.
2. Generate a (large) set of design points and compute their utility by sampling the model.
3. Select the seemingly best design points and run the algorithm for these.
4. The new design points are added to the population and the loop starts over if the termination criterion is not reached.

A counter k is increased in each cycle and is used to determine the number of repeats that are performed for each setting to be statistically sound in the obtained results. Consequently, this means that the best design points so far are also run again to obtain a comparable number of repeats. SPOT provides tools to perform the following tasks:

1. *Initialize*. An initial design is generated. This is usually the first step during experimentation. The employed parameter region and the constant algorithm parameters have to be provided by the user.
2. *Run*. This is usually the second step. The optimization algorithm is started with configurations of the generated design. Additionally information about the algorithm's problem design are used in this step. The algorithm provides the results to SPOT.

Algorithm 10.1. SPOT

```
// phase 1, building the model:
let $A$ be the tuned algorithm;
generate an initial population $\mathbf{P} = \{\mathbf{p}^1, \ldots, \mathbf{p}^m\}$ of $m$ parameter vectors;
let $k = k_0$ be the initial number of tests for determining estimated utilities;
foreach $\mathbf{p} \in \mathbf{P}$ do
    run $A$ with $\mathbf{p}$ $k$ times to determine the estimated utility $x$ of $\mathbf{p}$;
end
// phase 2, using and improving the model:
while termination criterion not true do
    let $\mathbf{a}$ denote the parameter vector from $\mathbf{P}$ with best estimated utility;
    let $k'$ the number of repeats already computed for $\mathbf{a}$;
    build prediction model $F$ based on $\mathbf{P}$ and $\{x^1, \ldots, x^{|\mathbf{P}|}\}$;
    // (alternatively: Use several prediction models in
        parallel)
    generate a set $\mathbf{P}'$ of $l$ new parameter vectors by random sampling;
    foreach $\mathbf{p} \in \mathbf{P}'$ do
        calculate $f(\mathbf{p})$ to determine the predicted utility $F(\mathbf{p})$;
    end
    select set $\mathbf{P}''$ of $d$ parameter vectors from $\mathbf{P}'$ with best predicted utility ($d \ll l$);
    run $A$ with $\mathbf{a}$ $k - k' + 1$ times and recalculate its estimated utility using all $k + 1$ test
        results; // (improve confidence)
    let $k = k + 1$;
    // (alternatively: Use more enhanced update schemes like
        OCBA)
    run $A$ $k$ times with each $\mathbf{p} \in \mathbf{P}''$ to determine the estimated utility $F(\mathbf{p})$ extend the
    population by $\mathbf{P} = \mathbf{P} \cup \mathbf{P}''$;
end
```

3. *Sequential step*. A new design is generated. A prediction model is used in this step. Several generic prediction models are available in SPOT already. To perform an efficient analysis, especially in situations when only few algorithms runs are possible, user-specified prediction models can easily be integrated into SPOT. Prediction models can also be used in parallel [19], which results in the so-called ensemble-based modeling approach. To improve confidence, the number of repeats can be increased. *Optimal computational budget allocation* (OCBA) [8, 25] is implemented as the default method for assigning new evaluations to algorithm configurations.
4. *Report*. An analysis, based on information from the results, is generated. Since all data flow is stored in files, new report facilities can be added very easily. SPOT contains some scripts to perform a basic regression analysis and plots such as histograms, scatter plots, plots of the residuals, etc.
5. *Automatic* mode. In the automatic mode, the steps *run* and *sequential* are performed after an initialization for a predetermined number of times.

10.3.3 Comparison of Automated and Interactive Tuning

SPOT can be run in an automated and in an interactive mode. The automated mode might be of interest for the user who is primarily interested in the result and who can afford a tuning procedure which is not restricted to a very small number of algorithm runs. Similar to microscopes in biology, SPOT can be used as a “datascope” to gain insight into algorithm behavior, by revealing factor effects and their importance to the experimenter. Such insights are not only used to guide the interactive parameter optimization process, but are also of intrinsic value to the developer or end user of a target algorithm.

The classical response surface methodology (as discussed in Chap. 15 of [23]) underlying our interactive approach was developed not only for finding parameter settings that achieve improved performance, but also to provide insights into how the performance of a target algorithm is affected by parameter changes. This latter question is related to the analysis of the response surface in the region of interest. Contour plots, which can easily be obtained in the SPOT framework, are useful tools to answer it.

We recommend using classical regression models at the first stage of an interactive approach, because these models can be interpreted quite easily; features of the response surface can be seen directly from the regression equation $Y = X\beta$. This is not the case for more sophisticated prediction models, such as neural networks or Gaussian process models. Furthermore, as demonstrated in [42], it is possible to obtain competitive results using such simple models. Nevertheless, in principle, more complex regression models could be used in the context of the interactive sequential parameter optimization approach. Furthermore, we note that observations and hypotheses regarding the dependence of a given target algorithm’s performance on its parameter settings could also be obtained by analyzing more complex models, including the Gaussian process models constructed by the automatic sequential parameter optimization procedures.

Clearly, the interactive approach makes it easy to use results from early stages of the sequential parameter optimization process to effectively guide decisions made at later stages. For example, looking back at the initial stages of the process, the experimenter can detect that the set of variables studied at this stage was chosen poorly, or that inappropriate ranges were chosen for certain variables. We note that the models used in early stages of the automated procedures also provide guidance to later stages of the process. However, the interactive process leaves room for expert human judgment, which can often be more effective in terms of the improvement achieved based on a small number of target algorithm runs.

The human expertise required to use the interactive approach successfully can be seen as a drawback compared to fully automated approaches. However, by providing dedicated support for the various operations that need to be carried out in this context, SPOT eases the burden on the experimenter and lowers the barrier to using the interactive approach effectively.

10.4 Case Study: Tuning Simulated Annealing

This section presents a working example to demonstrate essential principles of AEX introduced in Sect. 10.3. The following study will be referred to later in Sect. 10.7, which discusses metastatistical principles. This case study was set up to illustrate key elements of the AEX framework. It does not present a complete analysis of simulated annealing, but can serve as a starting point for an experimental analysis.

Our goal is to determine an improved parameter-setting for a simulated annealing search heuristic. This goal can be formulated as the following scientific question (this refers to AEX-1):

> Can we modify the algorithm design in such a manner that SANN's performance and robustness is improved?

Furthermore, we are seeking for tools which provide support in deciding whether this improvement is scientifically (or in practice) meaningful.

10.4.1 Simulated Annealing

Simulated Annealing (SANN) belongs to the class of stochastic global optimization methods. The R implementation, which was investigated in our study, uses the Metropolis function for the acceptance probability. It is a variant of the simulated annealing algorithm given in [21]. SANN uses only function values but is relatively slow. It will also work for non-differentiable functions. By default the next candidate point is generated from a Gaussian Markov kernel with scale proportional to the actual temperature. Temperatures are decreased according to the logarithmic cooling schedule as given in [21]; specifically, the temperature is set to $\texttt{temp}/\log(((t-1)/\texttt{tmax})\times\texttt{tmax}+\exp(1))$, where t is the current iteration step, and `temp` and `tmax` are specifiable via control. SANN is not a general-purpose method but can be very useful in getting to a good value on a very rough surface.

SANN uses two design variables, which were tuned during our study:

`temp`	is the starting temperature for the cooling schedule. Defaults to 10.
`tmax`	is the number of function evaluations at each temperature. Defaults to 10.

The interval from 1 to 50 was chosen as the *region of interest* (ROI) for both design variables in our experiments. The total number of function evaluations (of the Branin function, see Sect. 10.4.2) was set to $\texttt{maxit} = 250$ for all experiments. The starting point, i.e., the initial value for the parameters to be optimized over, was $\mathbf{x}_0 = (10, 10)$.

10.4.2 Description of the Objective Function

The Branin function

$$f(x_1, x_2) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10 \times \left(1 - \frac{1}{8\pi}\right)\cos(x_1) + 10,$$

with

$$x_1 \in [-5, 10] \text{ and } x_2 \in [0, 15] \tag{10.3}$$

was chosen as a test function, because it is well-known in the global optimization community, so results are comparable. It has three global minima, $\mathbf{x}_1^* = [3.1416, 2.2750]$, $\mathbf{x}_2^* = [9.4248, 2.4750]$, and $\mathbf{x}_3^* = [-3.1416, 12.2750]$ with $y^* = f(\mathbf{x}_i^*) = 0.3979$, $(i = 1, 2, 3)$. Results from the corresponding tuning experiments will be used in Sect. 10.7.1 to discuss particular aspects of the active experimentation framework.

10.5 Hypothesis Testing

We will describe the classical Neyman–Pearson statistical framework, which includes pre-data statistics such as significance levels, errors of the first and second kind, and power.

10.5.1 Neyman–Pearson Tests

To illustrate the concept of hypothesis testing, we introduce some basics from statistics. Hypothesis testing is the key element of step AEX-2 in the active experimentation framework.

Following the definition in Mayo [53], *Neyman and Pearson* (N-P) tests can be described as follows. The set of all possible values of the sample $\mathbf{X} = (X_1, X_2, \ldots, X_n)$ with realizations $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is $\mathcal{X}$, and Θ is the set of all possible values of the unknown parameters θ. A *statistical model* is represented as a pair $(\mathcal{X}, \Theta)$.

A null hypothesis, H_0, and an alternative hypothesis, H_1 are stated. These hypotheses partition the parameter space of the statistical model. The generic form of the null and alternative hypotheses is

$$H_0 : \theta \in \Theta_0 \text{ versus } H_1 : \theta \in \Theta_1, \text{ where } (\Theta_0, \Theta_1) \text{ is a partition of } \Theta.$$

We will use $P(\mathbf{x}; H)$ for the probability of $\mathbf{x}$ under H to avoid confusion with conditional probabilities, $P(\mathbf{x}|H)$, where H denotes a random variable (Bayes' rule).

A *test statistic* $d(\mathbf{X})$ reflects the distance from H_0 in the direction of H_1. To simplify the following discussion, we consider a sample $\mathbf{X} = (X_1, X_2, \dots, X_n)$ where the X_is are assumed to be normal, independent, and identically distributed with known standard deviation σ, i.e., $X_i \sim \mathcal{N}(\mu, \sigma^2)$. Here, the unknown parameter θ is the mean μ of the normal distribution. The test statistic is

$$d(\mathbf{X}) = \frac{\overline{X} - \mu_0}{\sigma/\sqrt{n}} = \frac{\overline{X} - \mu_0}{\sigma_x}, \tag{10.4}$$

where $\overline{X}$ is the sample mean, μ_0 is the hypothesized population mean under the null hypothesis, and σ_x denotes the standard error. We will consider one-sided tests in the following. Based on the prespecified α value, the *critical value* $c_{1-\alpha}$, which partitions Θ into the *region of acceptance*, $C_0(\alpha)$ and the *region of rejection*, $C_1(\alpha)$ of the null hypothesis, can be determined as the quantile $z_{1-\alpha}$ of the standard normal distribution.

$$C_0(\alpha) = \{\mathbf{x} \in \mathcal{X} : d(\mathbf{x}) \leq c_{1-\alpha}\}$$
$$C_1(\alpha) = \{\mathbf{x} \in \mathcal{X} : d(\mathbf{x}) > c_{1-\alpha}\}.$$

The *type I error probability* (or error of the first kind, α error) is

$$P(d(\mathbf{X}) > c_{1-\alpha}; H_0) \leq \alpha,$$

and represents the probability that the null hypothesis is rejected erroneously. The *type II error probability* (or error of the second kind, β error) is

$$P(d(\mathbf{X}) \leq c_{1-\alpha}; H_1) = \beta(\mu_1),$$

where μ_1 is the hypothesized population mean under the alternative hypothesis, *error-statistical methods* describe methods using error probabilities based on the relative frequencies of errors in repeated sampling. Probability is used to quantify how frequently methods are able to discriminate between alternative hypotheses and their reliability of error detection [54]. Following Mayo [53], we will use the term *error statistics* for hypothesis tests, statistical significance tests, and related error probability methods.

Example 10.1. We consider one particular test, $T(\mathbf{x}; \alpha; \theta) = T(\alpha)$ about the mean with significance level $\alpha = 0.025$. The null hypothesis $H_0 : \mu \leq \mu_0$ is tested versus the alternative hypothesis $H_1 : \mu > \mu_0$. Here, $c_{1-\alpha}$ can be determined as the quantile $z_{1-\alpha}$ of the standard normal distribution, e.g., $c_{1-0.025} = z_{1-0.025} = 1.96$, therefore

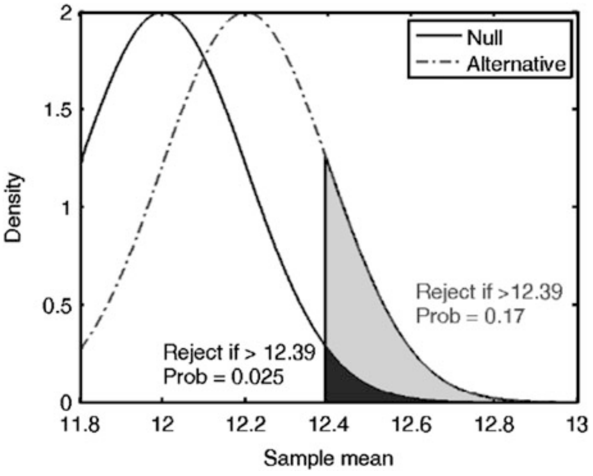


Fig. 10.3 One-sided hypotheses test. Based on the null and alternative hypotheses and α, the significance level, a test can be performed. We assume a known standard deviation, say $\sigma = 2$ and a sample size of $n = 100$. If the mean value $\overline{x}$ is larger than 12.39, the null hypothesis is rejected, otherwise it is accepted. The *dark gray* shaded region represents the Type I error probability. The alternative hypothesis reads $\mu_1 = 12.2$. The *light gray* shaded region (which includes also the *dark gray* shaded region) represents the power of the test, i.e., $1 - \beta$

$$C_0(\alpha) = \{\mathbf{x} \in \mathcal{X} : d(\mathbf{x}) \leq 1.96\},$$
$$C_1(\alpha) = \{\mathbf{x} \in \mathcal{X} : d(\mathbf{x}) > 1.96\}.$$

Furthermore, let $\mu_0 = 12$, $\sigma = 2$, and $n = 100$. The null hypothesis $H_0 : \mu \leq 12$ is tested versus the alternative hypothesis $H_1 : \mu > 12$. The test rule derived from this test reads: Reject H_0 if $d(\mathbf{x}_0) > 1.96$, or if $\overline{x} = \mu_0 + d(\mathbf{x}_0) \times \sigma_x > 12.39$, see Fig. 10.3. If the observed value of the test statistic falls into the rejection region, we will reject the null hypothesis at a 2.5 % significance level. □

Acceptance and rejection are associated with certain actions, e.g., publishing a result about effects of modifying the recombination operator of an evolutionary algorithm. But, how can we be sure that this action is justified, e.g., scientifically correct or meaningful? The *behavioristic rationale* answers this as follows:

We are justified in "accepting/rejecting" hypotheses in accordance with tests having low error probabilities because we will rarely err in repeated applications.[64]

This rationale, which was formulated by J. Neyman and E.S. Pearson in 1933, is based on the idea that error probabilities are means to determine "the evidence a set of data x_0 supplies for making warranted inferences about the process giving rise to data x_0" [53].

10.5.2 Power of a Test

The severity concept, which is introduced in Sect. 10.6, is related to the power of a test. The power of a test is a standard concept in hypotheses testing. It is the test's probability of correctly rejecting the null hypothesis, i.e., the complement of the false negative rate, β. The *power of a test* is defined as

$$\mathrm{POW}(T(\alpha);\mu_1) = P(d(\mathbf{X}) > c_{1-\alpha};\mu_1), \text{ for } \mu_1 > \mu_0. \tag{10.5}$$

Power curves illustrate the effect on power of varying the alternate hypothesis. Severity, which uses post-data information, was introduced by Mayo [53] as "the attained power". Severity can be seen as an extension of the power of a test, cf. Sect. 10.6.2.

Example 10.2. The power of the test specified in Example 10.1, where the null hypothesis $H_0 : \mu \leq 12$ is tested versus the alternative hypothesis $H_1 : \mu > 12$ can be determined with Eq. (10.5) as follows:

$$\mathrm{POW}(T(\alpha);\mu_1) = P(d(\mathbf{X}) > c_{1-\alpha};\mu = \mu_1) = P\left(\frac{\overline{X} - \mu_0}{\sigma_x} > c_{1-\alpha};\mu = \mu_1\right) \tag{10.6}$$

Since $\mu = \mu_1$,

$$\frac{\overline{X} - \mu_0}{\sigma_x} \quad \text{follows a} \quad \mathcal{N}\left(\frac{\mu - \mu_0}{\sigma_x}, 1\right)$$

distribution. Therefore

$$\mathrm{POW}(T(\alpha);\mu_1) = 1 - \Phi\left(c_{1-\alpha} + \frac{\mu_0 - \mu_1}{\sigma_x}\right), \tag{10.7}$$

where Φ denotes the cumulative distribution of the probability density function of the standard normal distribution, i.e.,

$$\Phi(x) = P(t < x) = \frac{1}{\sqrt{2\pi}} \int_{t=-\infty}^{t=x} \exp\left(-\frac{t^2}{2}\right) dt.$$

We are using the values from Example 10.1, i.e., $\alpha = 0.025$, $c_{1-0.025} = z_{1-0.025} = 1.96$, $\mu_0 = 12$, $\sigma = 2$, and $n = 100$. Power depends on the value of the alternative, i.e., μ_1. For $\mu_1 = 12.2$, we determine:

$$\begin{aligned}&\mathrm{POW}(T(\alpha = 0.025);\mu_1 = 12.2)\\ &= P\left(Z > 1.96 + \frac{12 - 12.2}{0.2}\right) = 1 - \Phi(0.96) = 0.1685.\end{aligned}$$

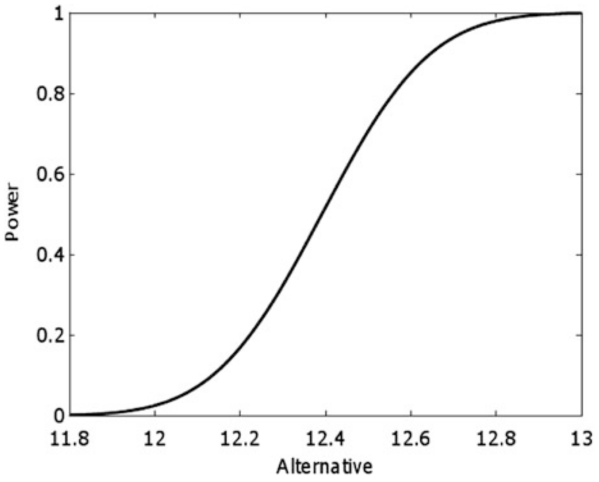


Fig. 10.4 Power of a test. We are using the values from Example 10.1, i.e., $\alpha = 0.025$, $c_{1-0.025} = z_{1-0.025} = 1.96$, $\mu_0 = 12$, $\sigma = 2$, and $n = 100$. Power depends on the alternative, i.e., μ_1. For $\mu_1 = 12.2$, we determine: $\mathrm{POW}(T(\alpha = 0.025); \mu_1 = 12.2) = 0.1685$

To determine the power for various μ_1 values, we obtain

$$\mathrm{POW}(T(\alpha = 0.025); \mu_1) = P\left(Z > 1.96 + \frac{12 - \mu_1}{0.2}\right) \text{ with } Z \sim \mathcal{N}(0, 1), \tag{10.8}$$

see Fig. 10.4. □

Very significant results can be obtained with high power, even if the size of the effect is of no practical relevance: The effect is there, but its magnitude is of little value. This is similar to the situation with p values, see [13]. On the other extreme, a study with low power will have indecisive results, even if the effect is real and relevant.

N-P theory has been under attack, basically for the following three problems.

P-1: N-P tests are too coarse, because they tell us to reject or accept a certain hypothesis H, but do not indicate the level of rejection or acceptance.
P-2: Since statistical significance and not scientific importance is considered, N-P tests give rise to fallacies of rejection and of acceptance.
P-3: N-P tests focus on pre-data, i.e., information from new data is not considered.

The power of a test does not depend on the experimental result $\mathbf{x}_0$, it remains the same for different outcomes. Even if the experimental result gives better evidence for accepting or rejecting the null hypothesis, the power will be identical. Power is no solution to problems P-1 to P-3, because the power of a test retains its coarseness, it does not consider its scientific importance, and it relies on pre-experimental data. This applies to confidence intervals as well, because they do not consider the experimental outcome. Mayo introduces severity as a basic concept for post-data inference. Example 10.4 illustrates the difference between power and severity.

10.6 Severity

10.6.1 Motivation

Severity provides a metastatistical principle for evaluating proposed statistical inferences. It tells us how "well probed" (not "how probable") hypotheses are and is an attribute of the test procedure as a whole. That is, severity should be calculated after the test procedure is finished. Once the data $\mathbf{x}_0$ of the test $T(\alpha)$ are available, they enable us to evaluate the severity of the test and the resulting inference.

Example 10.3. In order to exemplify the concept of severity, we consider the following situation (see also [52, p. 183]): A randomized search algorithm, say (A), has scored high success rate on a test problem. It is able to detect the optimum in 96.3 % of the runs. Consider the following situations:

1. First, suppose that it would be extraordinary for an algorithm, say A^*, that has no domain knowledge at all, to have a score as high, or higher, than A. Is this score good evidence that A is well-suited for solving this problem? Based on A's and A^*'s test results, the severity rationale would be in this case that this inference is warranted.
2. Next, suppose that it would be no problem for an algorithm A^* that has no domain knowledge, e.g., random search, to have a score as high as 96 %. Again, we may ask the same question: Is this score of 96.3 % good evidence that A is well-suited for this test problem? Based on information about A's and A^*'s high score results, the severity rationale would be in this case that this inference is not warranted. The severity concept should provide tools for detecting ceiling effects.

□

10.6.2 Severe Tests

These considerations lead to the definition of severity as a concept for post-data inference. Here, we are facing the situation that a test has been performed and a decision ("accept" or "reject" hypothesis H) has been made. The following definition of a severe test is presented in [53, p. 7]:

Definition 10.3 (Severe Test). A statistical hypothesis H passes a *severe test* T with data $\mathbf{x}_0$ if,

S-1 $\mathbf{x}_0$ agrees with H, and
S-2 with very high probability, test T would have produced a result that accords less well with H than $\mathbf{x}_0$ does, if H were false.

□

Instead of calculating the power, which does not include information from the test result,

$$\text{POW}(T(\alpha); \mu_1) = P(d(\mathbf{X}) > c_{1-\alpha}; \mu = \mu_1),$$

for $\mu_1 > \mu_0$, see Eq. (10.5), Mayo [53] introduces the attained power or *severity*

$$\text{SEV}(T(\alpha); d(\mathbf{x}_0); \mu \leq \mu_1) = P(d(\mathbf{X}) > d(\mathbf{x}_0); \mu > \mu_1) \tag{10.9}$$

in case of acceptance of the null and

$$\text{SEV}(T(\alpha); d(\mathbf{x}_0); \mu > \mu_1) = P(d(\mathbf{X}) \leq d(\mathbf{x}_0); \mu \leq \mu_1) \tag{10.10}$$

in case of rejection of the null. In order to simplify notation, we suppress the arguments $T(\alpha)$ and $d(\mathbf{x}_0)$ in the following and use the abbreviations $\text{SEV}(\mu \leq \mu_1)$ and $\text{SEV}(\mu > \mu_1)$, respectively.

Equation (10.9) states that $\mu \leq \mu_1$ passes the test with high severity if there is a very high probability that $d(\mathbf{x}_0)$ would have been larger than it is, were $\mu > \mu_1$. And Eq. (10.10) states that $\mu > \mu_1$ passes the test with high severity if there is a very high probability that $d(\mathbf{x}_0)$ would have been smaller than it is, were $\mu \leq \mu_1$. Note, severity depends on the test and the test result, i.e., it includes post-data information from $d(\mathbf{x}_0)$ instead of $c_{1-\alpha}$. Similar to the calculation of the power, the severity can be determined. Note, that based on severity criterion S-1, we have to determine whether data from the test result lead to an acceptance or an rejection of the hypothesis H.

10.6.2.1 Severity in the Case of Acceptance of the Null

First, the determination of severity of acceptance for test $T(\alpha)$ is considered. For example, this situation arises if no difference in means can be found. Based on the outcome $d(\mathbf{x}_0) \leq c_{1-\alpha}$, H_0 has survived the test. In this case, a statistically insignificant result ("accept H_0" or "$\mu \leq \mu_1$") is considered. Severity can be calculated as follows:

$$\begin{aligned}
\text{SEV}(\mu \leq \mu_1) =& P\ (d(\mathbf{X}) > d(\mathbf{x}_0); \mu \leq \mu_1 \text{is false }) \\
=& P\ (d(\mathbf{X}) > d(\mathbf{x}_0); \mu > \mu_1) \\
=& P\left(\frac{\overline{X} - \mu_0}{\sigma_x} > \frac{\overline{x}_0 - \mu_0}{\sigma_x}; \mu > \mu_1\right) \\
=& P\left(Z > \frac{\overline{x}_0 - \mu}{\sigma_x}\right) \text{ with } Z \sim \mathcal{N}(0, 1) \\
=& 1 - \Phi\left(\frac{\overline{x}_0 - \mu}{\sigma_x}\right).
\end{aligned} \tag{10.11}$$

□

Note, a t distribution is used if σ is unknown. In this case, the test statistic of Eq. (10.4) reads

$$d(\mathbf{x}_0) = \frac{\overline{X} - \mu_0}{S_n/\sqrt{n}},$$

where S_n is defined as

$$S_n = \sqrt{\frac{1}{n-1}\sum(X_i - \overline{X})^2}.$$

10.6.2.2 Severity in the Case of Rejection of the Null Hypothesis

Severity can be calculated as

$$\text{SEV}(\mu > \mu_1) = 1 - \text{SEV}(\mu \le \mu_1),$$

because

$$\begin{aligned}\text{SEV}(\mu > \mu_1) =& P\ (d(\mathbf{X}) \le d(\mathbf{x}_0); \mu > \mu_1 \text{ is false })\\ =& \Phi\left(\frac{\overline{x}_0 - \mu}{\sigma_x}\right).\end{aligned}$$

□

A comparison of Eqs. (10.7) and (10.11) shows that severity does not directly use the information from the critical value $c_{1-\alpha}$ and from the significance level α. This information is used indirectly, because the inference (accept/reject) is used to calculate severity.

Example 10.4. Similar to the calculation of the power of a test in Example 10.2 we will determine the severity. We are using the values from Example 10.1, i.e., $\alpha = 0.025$, $c_{1-0.025} = z_{1-0.025} = 1.96$, $\mu_0 = 12$, $\sigma = 2$, and $n = 100$. Again, the null hypothesis $H_0 : \mu \le 12$ is tested versus the alternative hypothesis $H_1 : \mu > 12$. Similar to power, severity is evaluated at a point $\mu_1 = \mu_0 + \gamma$, where γ denotes the difference from μ_0 which is considered meaningful. Here, we have chosen $\gamma = 0.2$, which results in $\mu_1 = 12 + 0.2 = 12.2$. As can be seen from Eq. (10.11), severity depends on the experimental outcome, i.e., $\overline{x}_0$. For $\overline{x}_0 = 11.8$, we obtain:

$$\begin{aligned}\text{SEV}(\mu \le \mu_1) =& P\left(Z > \frac{11.8 - 12.2}{0.2}\right)\\ =& P(Z > -2) \text{ with } Z \sim \mathcal{N}(0,1)\\ =& 1 - \Phi(-2) = 0.977.\end{aligned}$$

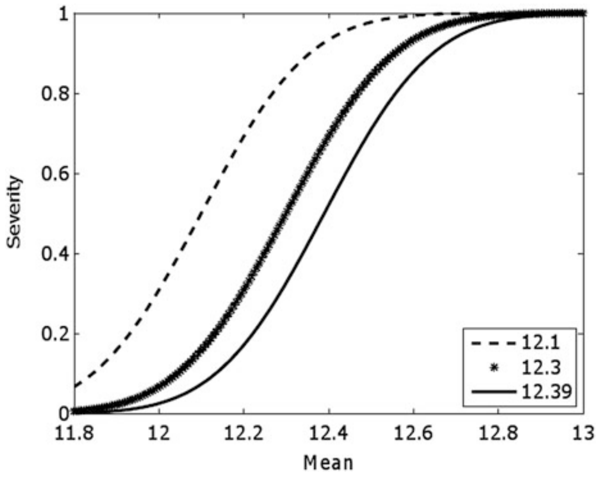


Fig. 10.5 Severity for three different results $\overline{x}_0$: 12.1, 12.3, and 12.39. These curves can be interpreted as follows: Consider, e.g., $\overline{x}_0 = 12.3$, which gives $d(\mathbf{x}_0) = 1.5$: The assertion that $\mu \leq 13$ severely passes because $\mathrm{SEV}(\mu \leq 13) = 0.9998$

$\square$

In case of a rejection of the alternative, the power of a test provides a lower bound for the severity. This can be seen from Eqs. (10.6) and (10.9). Power and severity are the same, if $d(\mathbf{x}_0)$ equals $c_{1-\alpha}$.

10.6.2.3 Usage of the Severity Concept

The framework presented in this section can be used as a metastatistical check to evaluate which inferences are warranted. Figure 10.5 illustrates this check. The severity for three different outcomes $\overline{x}_0$ is shown. Severity increases for smaller values of $\overline{x}_0$. The power curve and the severity curve coincide for $\overline{x}_0 = 12.39$, i.e., $d(\mathbf{x}_0) = c_{1-\alpha}$ or $\overline{x}_0 = \mu_0 + c_{1-\alpha} \times \sigma_x$.

The curves from Fig. 10.5 can be used to compare the severity of the assertion $\mu \leq 12.4$ for different experimental outcomes: First, $\overline{x}_0 = 12.1$ is considered. Here, we obtain

$$\begin{aligned}&\mathrm{SEV}(T(\alpha = 0.025); d(\mathbf{x}_0) = 12.1; \mu \leq 12.4)\\ &= P\left(Z > \frac{12.1 - 12.4}{0.2}\right) = 1 - \Phi(-1.5) = 0.9332.\end{aligned}$$

So, the assertion $\mu \leq 12.4$ passes with high severity. Next, the experimental outcome $\overline{x}_0 = 12.39$ is considered. Here, we obtain

$$\begin{aligned}&\mathrm{SEV}(T(\alpha = 0.025); d(\mathbf{x}_0) = 12.39; \mu \leq 12.4)\\ &= P\left(Z > \frac{12.39 - 12.4}{0.2}\right) = 1 - \Phi(-0.05) = 0.5199,\end{aligned}$$

i.e., this experimental result decreases severity.

The severity curves from Fig. 10.5 can also be used in the following way: Let $\overline{x}_0 = 12.1$. The practitioner selects a relatively high severity value, say 0.9. The related μ value is calculated, say 12.4. That is, the assertion that "$\mu \leq 12.4$" severely passes with the result $\overline{x}_0 = 12.1$.

In addition, we present one useful application of the severity criterion for the experimental analysis of algorithms.

Example 10.5. Comparing a newly developed algorithm A^* to the best known algorithm A might lead to a situation where the result is interpreted too readily as positive evidence of no difference in their performances. Here, we are using the following values: $\alpha = 0.025$, $c_{1-0.025} = z_{1-0.025} = 1.96$, $\mu_0 = 0.0$, $\sigma = 2$, and $n = 100$. The null hypothesis $H_0 : \mu \leq 0$ is tested versus the alternative hypothesis $H_1 : \mu > 0$. Let the test $T(\alpha)$ yield a statistically insignificant result $\overline{x}_0 = 0.3$, i.e., the alternative is rejected. The experimenter states that "any discrepancy from $\mu_0 = 0$ is absent or no greater than 0.1." How severely does $\mu \leq 0.1$ pass with $\overline{x}_0 = 0.3$? We obtain

$$\begin{aligned} &\mathrm{SEV}(T(\alpha = 0.025); d(\mathbf{x}_0) = 0.3; \mu \leq 0.1) \\ &= P\left(Z > \frac{0.3 - 0.1}{0.2}\right) = 0.1587. \end{aligned}$$

So, even if a difference of 0.1 exists, such a result would occur 84 % of the time. Clearly, severity does not support the experimenter's statement in this case. □

Severity was developed as an error statistical tool in the framework of the *new experimentalism*. The new experimentalists claim that theories present only heuristic rules, leading us to experimental knowledge [24]. They view progress in terms of the accumulation of experimental discoveries. These findings are independent of high-level theory. How to produce scientifically meaningful results is the central theme in the research of the new experimentalists. Bartz-Beielstein [1, 2] demonstrates how these concepts can be transferred from the philosophy of science to computer science.

10.7 Metastatistical Principles

This section refers to the third and fourth step (AEX-3 and AEX-4, respectively) of the active experimentation framework. As introduced in Sect. 10.4, we will present a working example to illustrate particular aspects of the active experimentation framework. Next, experiments will be performed. The corresponding results are shown in Sect. 10.7.1. These results will be used to discuss differences between statistically significant and scientifically meaningful results in Sect. 10.7.2. Finally, ceiling effects will be revisited in Sect. 10.7.3.

Table 10.1 SANN results. Results from $n = 100$ repeats. Smaller values are better. The optimal function value is $y^* = 0.3979$

Model	Min.	First qu.	Median	Mean	Third qu.	Max.
Default	0.3982	0.4037	0.4130	0.8281	0.5032	6.1120
Random	0.3988	0.5326	1.2160	2.0720	2.9820	8.8800
Tuned	**0.3979**	**0.3987**	0.4000	**0.4010**	**0.4022**	**0.4184**

10.7.1 Results from Default, Random, and Tuned Settings

Experiments are performed at this stage, i.e., SPOT is used to execute algorithm runs in an objective and reproducible manner. This is step AEX-3 from the active experimentation framework. As a baseline for our experiments, we run SANN 100 times—first, with default parameters (`tmax` = `temp` = 10), and second, with randomly chosen parameter values from the interval $[1, 50]$.[1] These experiments were performed to quantify the benefit of tuning for our experiments. Results from these two experiments are shown in the first and second result row from Table 10.1. SANN was not able to determine the optimal function value with these settings. Now that the baseline results are available, we can examine SANN's tunability.

The final best configuration found by SPOT reads `temp` = 1.115982 and `tmax` = 38. Now that these results are available, we would like to determine their statistical significance and scientific meaning.

10.7.2 Spurious Effects

As mentioned in Sect. 10.2.7, our focus lies on metastatistical principles that can be applied after the experiments are performed. These principles are necessary to avoid fallacies, i.e., misconceptions resulting from incorrect reasoning in argumentation caused by spurious effects. Following Cohen [28], we define *spurious effects* as effects that suggest that a treatment is

- Effective when it is not, or
- Not effective when it is.

One prominent example for spurious effects is the *ceiling effect*. If one wants to investigate performance differences between different methods, it is important to select the test problems/settings so that these differences indeed can occur. It is of little interest to see result tables with nearly all the methods always obtaining success rates of 100 %. This would be a *ceiling effect*: The test problems are too easy, so all

[1]SPOT can generate 100 randomly chosen design points of the SANN by using the following setting in the CONF file: `init.design.size = 100` and `init.design.repeats = 1`.

algorithms "crash into the ceiling". On the other hand, test problems can also be too hard, then we have a *floor effect* because most measured algorithms never obtain a measurable progress: All remain "on the floor". Particularly quality tasks that may not be reached (and thus not counted for success rates) have to be set up with care. When floor/ceiling effects occur, there is almost no variability in the data, and thus all compared algorithms appear to obtain similar performance. Ceiling effects occur when test problems are not sufficiently challenging. In the hypothesis-testing framework from statistics, the situation can be formulated as the following claim.

Claim 1. *Let A and A^* denote two algorithms and consider the hypothesis*

$$H : perf(A) \geq perf(A^*).$$

If A and A^* achieve a performance which is close to the maximum level of performance, H should not be confirmed due to a ceiling effect. □

Claim 1 describes a situation in which there is a high probability that algorithms A and A^* reach a similar high performance, i.e., the difference in their performances $\mathrm{perf}(A^*) - \mathrm{perf}(A)$ is small. This corresponds to S-1 and S-2 from Definition 10.3: With very low probability, the comparison of $\mathrm{perf}(A)$ with $\mathrm{perf}(A^*)$ would have produced a result that accords with the hypothesis "H: There is no difference in their performances" as well as or better than the test result does, if H were false and a given difference were present. Consequently, severity can be used to detect ceiling effects.

10.7.3 Ceiling Effects Revisited

Now the necessary tools for performing a post-data analysis are available and can be applied to the results from the SANN case study (see Sect. 10.4). This refers to the fourth step of the active experimentation framework, i.e., AEX-4. The function `spotSeverity()` and a related plotting functions are implemented in the R version of the SPO toolbox,[2] which is available via CRAN [4].

Summary statistics from these two run configurations were shown in Table 10.1. These results indicate that SANN with tuned parameters outperforms SANN with default parameters. We will apply error statistical tools to analyze the scientific meaning of this result. Based on results from this case study, a power and a severity plot is generated, see Fig. 10.6. A histogram illustrates the importance of EDA tools:

[2] R is a freely available language and environment for statistical computing and graphics which provides a wide variety of statistical and graphical techniques. CRAN is a network of ftp and web servers around the world that store identical, up-to-date versions of code and documentation for R, see http://cran.r-project.org.

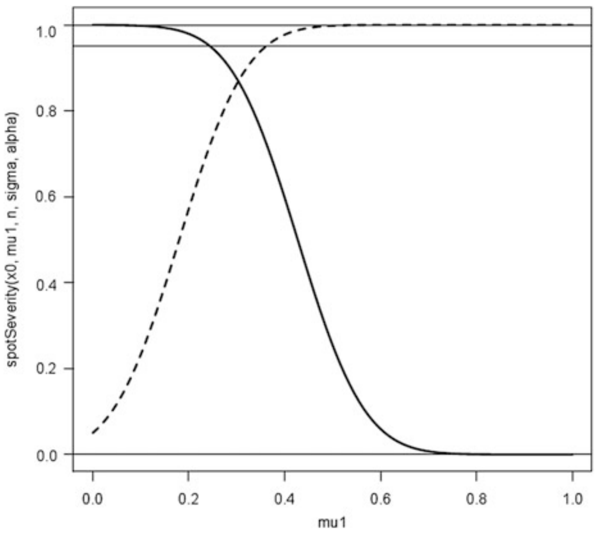


Fig. 10.6 Comparison of the tuned and the default SANN configuration on the Branin function with 250 function evaluations. Each run configuration was run 100 times. The null hypothesis “there is no difference in means” is rejected. The *dotted line* illustrates the power of the test, whereas the *solid line* represents the severity. This plot was generated with the function `spotPlotSeverity()` from R’s SPOT package

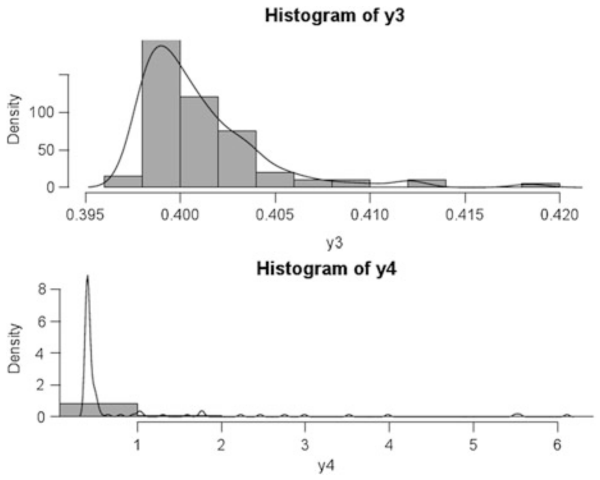


Fig. 10.7 Comparison of the tuned and the default SANN configuration on the Branin function with 250 function evaluations. Each run configuration was run 100 times. The null hypothesis “there is no difference in means” is rejected

In a second experiment, we increased the number of SANN function evaluations from `maxit` = 250 to 1,000,000 (Fig. 10.7). This problem design is used to illustrate a ceiling effect (the problem is too easy, because of the large number of function evaluations): Again, each configuration is run 100 times. First, we will compare simple summary statistics from these two run configurations, see Table 10.2.

Both algorithms show the same behavior (up to four digits after the decimal) if the number of function evaluations is set to $10e6$. However, a t-test claims that there is a statistically significant difference in means. In addition, we generate a t-test:

```
     Paired t-test
t = -8.8975, df = 99, p-value = 1.384e-14
alternative hypothesis: True difference in means is less than 0
```

Table 10.2 SANN results. Results from $n = 100$ repeats. Smaller values are better. The optimal function value is $y^* = 0.3979$

Model	`maxit`	Min.	First qu.	Median	Mean	Third qu.	Max.
Tuned	250	**0.3979**	**0.3987**	0.4000	**0.4010**	**0.4022**	**0.4184**
Default	250	0.3982	0.4037	0.4130	0.8281	0.5032	6.1120
Tuned	1e6	0.3979	0.3979	0.3979	0.3979	0.3979	0.3979
Default	1e6	0.3979	0.3979	0.3979	0.3979	0.3979	0.3979

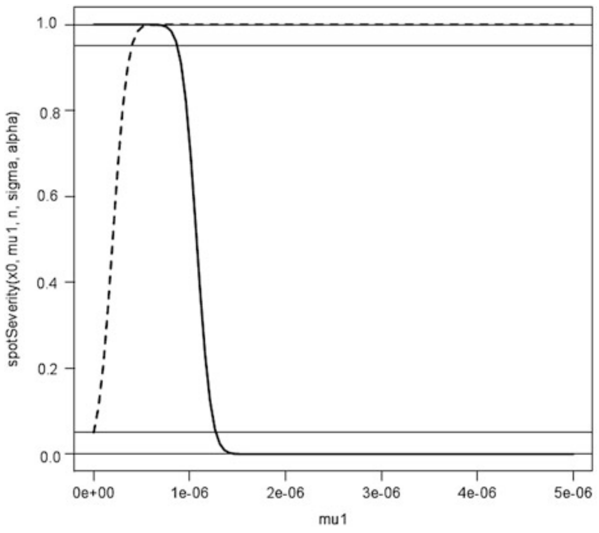


Fig. 10.8 Comparison of the tuned and the default SANN configuration on the Branin function with $1e6$ function evaluations. Each run configuration was run 100 times. The null hypothesis "there is no difference in means" is rejected. The *dotted line* illustrates the power of the test, whereas the *solid line* represents the severity

In this situation, error statistical tools and tools from exploratory data analysis might be helpful. Figure 10.8 shows the corresponding plots of power and severity. The severity interpretation of rejection shows that only very small differences in mean ($\mu < 2e-6$) pass with high severity.

A histogram (see Fig. 10.9) illustrates the importance of EDA tools. Results from the tuned SANN have a smaller standard deviation. However, it is up to the practitioner to decide whether a difference as small as $1e-6$ is of importance for this kind of problem instance. Error statistical tools provide support for this decision.

Now, we can answer the question from Sect. 10.4:

SANN's performance and robustness could be improved. Severity and EDA provide useful decision support tools.

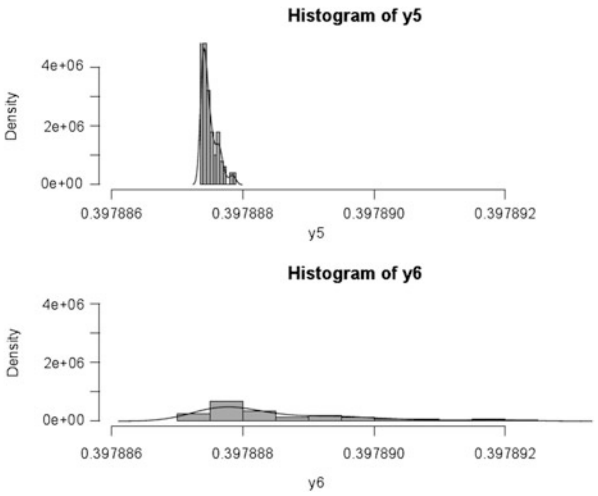


Fig. 10.9 Comparison of the tuned and the default SANN configuration on the Branin function with $1e6$ function evaluations. Each run configuration was run 100 times. The null hypothesis "there is no difference in means" is rejected

10.8 Exploratory Landscape Analysis

If the treated problem is not a constructed benchmark and thus most of its properties are unknown, it makes sense to use the test runs done with the optimization algorithm of choice to acquire some additional problem knowledge. If one evaluation of the optimization problem takes on the order of minutes or more to compute, one cannot simply apply standard tuning techniques (as documented in Sect. 10.3). Instead, one could generate a surrogate model from the evaluated points and tune on this surrogate problem [69], or integrate the tuning process within the algorithm itself, which is, e.g., easily possible when restarts are performed [81]. *Exploratory Landscape Analysis* (ELA) [57, 58] follows another approach, namely to detect problem properties first in order to to make a reasonably informed decision for some optimization algorithm.

10.8.1 Important Problem Properties

Problem properties which need to be determined to set up an optimization algorithm that matches the problem well are (more than these may be suitable, depending on the problem and the optimization algorithm):

Multimodality Most classic optimization algorithms inherently expect a unimodal (convex) problem. However, experience shows that most simulator-based problems are multimodal. But how multimodal? Do they have few local optima (as Schwefel's problem 2.13), or many (as Rastrigin's problem)? In the first case, a niching or time-parallel method may be useful; in the latter case, one has to rely on multistarts (see [66] for a discussion) or on using large populations to inherently average out the many peaks in order to detect the global basin structure. If indeed a convex problem is found, classic optimization methods as BFGS (see, e.g., [62]) are most likely more effective than evolutionary ones.

Global Basin Structure Rastrigin's problem is not as difficult as it may seem at first. It has a huge amount of local optima, but also a global basin structure due to the quadratic term: Seen from a large distance, it appears as parabola. Problems without global structure are more difficult because one virtually needs to "look in every corner." As an example, one may refer to the Gaussian mixture problem generator by Gallagher [34]. See [37] for more examples.

Separability If a problem is fully or partly separable, it may be partitioned into subproblems which are then of lower dimensionality and should be considerably easier to solve. For benchmark problems, it is known that separable problems become inseparable by simple geometric transformations as rotation [73].

Variable Scaling Even if the considered search space bounds are the same for all variables, the problem may behave very different in the single dimensions. It can be essential to perform small steps in some dimensions, and large ones in others. Some algorithms as, e.g., the CMA-ES [35] handle such problems well, but most standard EA variants do not.

Search Space Homogeneity Most benchmark sets are created with a homogeneous problem structure in mind, which is expressed by a single, relatively simple formula. However, real-world problems do not necessarily behave like this. The CEC'05 benchmark set [75] contains hybrid problems that consist of different ones blended into another, so that the resulting problem behaves differently in different search space areas.

Basin Size Homogeneity As, for example, emphasized by Törn [76], the basin size of the global optimum certainly influences the hardness of a problem. However, in the presence of many optima, the size relations of all encountered basins can lead to additional difficulties. Many algorithms for multimodal problems (e.g., most niching EA methods) assume similar basin sizes and use appropriately adjusted distances to differentiate between basins. If size differences are huge, these methods are doomed to fail.

Global to Local Optima Contrast This property refers to the height (quality) differences between global and local peaks in comparison to the average fitness level of a problem. It thus determines if very good peaks are easily recognized as such. Together with basin size homogeneity, the influence of this property on niching methods was reviewed in [70].

Size of Plateaus Plateaus make optimization problems harder as they do not provide any information about good directions to turn to. Large plateaus effectively cut the search space into parts that prevent path-oriented optimization algorithms from moving from one embedded peak area (possibly also a multimodal landscape itself) to another one.

This set of properties is a rather phenomenologically (in the sense that these features have been either explicitly modeled into benchmark problems or observed in visualizations of existing ones) motivated collection which stresses the importance of the global structure in real-valued search spaces. Some time ago, many measures based on mathematical properties of mainly bit-coded problems were suggested and employed for expressing hardness of problems for evolutionary algorithms. The most prominent of these may be the *fitness-distance correlation* (FDC) [46]. However, further theoretical investigations of Jansen [44] and He et al. [38] largely found the existing measures unsuitable for predictive purposes, so watching out for new properties surely makes sense. Furthermore, in exploratory landscape analysis, one is especially interested in what can be achieved with only few evaluations of the problem, as the ultimate goal usually is to set up a good optimization algorithm for expensive problems with unknown properties.

10.8.2 Exploratory Testing

Attempts to acquire experimentally property knowledge on expensive functions are usually performed manually, without a guiding algorithm. Instead they follow the intuition of the experimenter, which, during the process, adapts to the already-known facts.

Sampling, dimension reduction techniques, and especially visualization are important techniques to obtain problem knowledge. Interestingly, search points visited during stagnation phases may reveal interesting problem properties. We provide a real-world example as proof of concept. Figure 10.10 shows a fitness distance correlation plot from the last best point of one optimization run, using all successively sampled points (around 300). The treated problem [72] is the 15-variable engineering task to construct a ship propulsion system with high efficiency and low cavitation (low pressure bulbs at high velocity spots that lead to noise and deterioration), formulated as a constraint penalized single-objective problem. Simulation times are on the order of minutes. The question we tackled here was to find out why optimization always got stuck early. The plot shows several layers of fitness values, most likely stemming from the penalization when hitting a constraint.

Exploratory testing (sampling) can focus either on global or local features of a problem. Global sampling may employ any space-filling design as, for example, a *Latin hypercube design* (LHD), which is useful for obtaining a rough idea of the problem nature. Putting the sampled points into a model enables one to visualize the landscape, e.g., Kriging models as employed in Sect. 10.3 may be used. However, note that a model comes with certain assumptions (such as smoothness of the landscape), so that some features such as high-frequency ruggedness may completely go unnoticed.

Testing locally makes sense if one needs to find out which properties of the problem lead to stagnation in the optimization process, and then should be conducted in the neighborhood of the best-yet obtained points. However, for

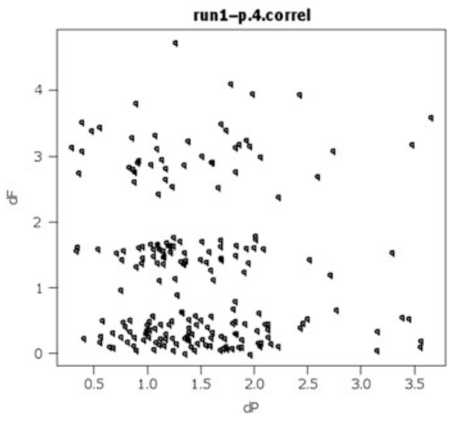


Fig. 10.10 Quality to search space distance correlation around a good solution, Δquality (dF) over Δsearch space distance (dP). Obtained from the last 300 samples performed by an (1 + 1)-ES on a moderately expensive engineering problem. All sampled points are equal or worse than the center point. The plot reveals a layered structure which is most likely due to constraint penalties applied to infeasible solutions

high-dimensional problems, running extensive grid tests in all possible variable combinations around a best solution is infeasible. Nevertheless, one may try grid tests in some combinations, especially if the variables can be grouped according to domain knowledge.

We investigate this for the ship propulsion problem described above, choosing two variable pairs with different properties. The first two variables, $c(x1)$ and $c(x05)$, both refer to the shape of the rotor and should interact strongly; the other two ($cStat(x05)$ and $kr(x0)$) are expected to interact only weakly. Figure 10.11 shows the results of 21×21 point grid scans around the same best found point as employed in Fig. 10.10, keeping all other variables constant. The histograms help in assessing the frequency of encountering infeasible solutions (fitness values around -1.5), which is surprisingly similar in both situations, while the landscapes are completely different. For the two strongly interacting variables, we obtain a highly rugged landscape, but for the weakly interacting ones it is relatively flat with linear cliffs. We can safely assume that this contrast makes the problem harder.

An alternative approach to grid scans would be a local model (e.g., generalized linear or Kriging), generated from a space-filling sample around the point of interest. However, even identifying the properties of optimization problems near high-fitness spots may keep other important properties secret. Unless a rather complete scan of the problem in all dimensions is possible (which would remove the necessity for an optimization algorithm), we cannot be sure to have revealed the information that could be helpful for setting up an optimization process fully. Our landscape analysis remains exploratory.

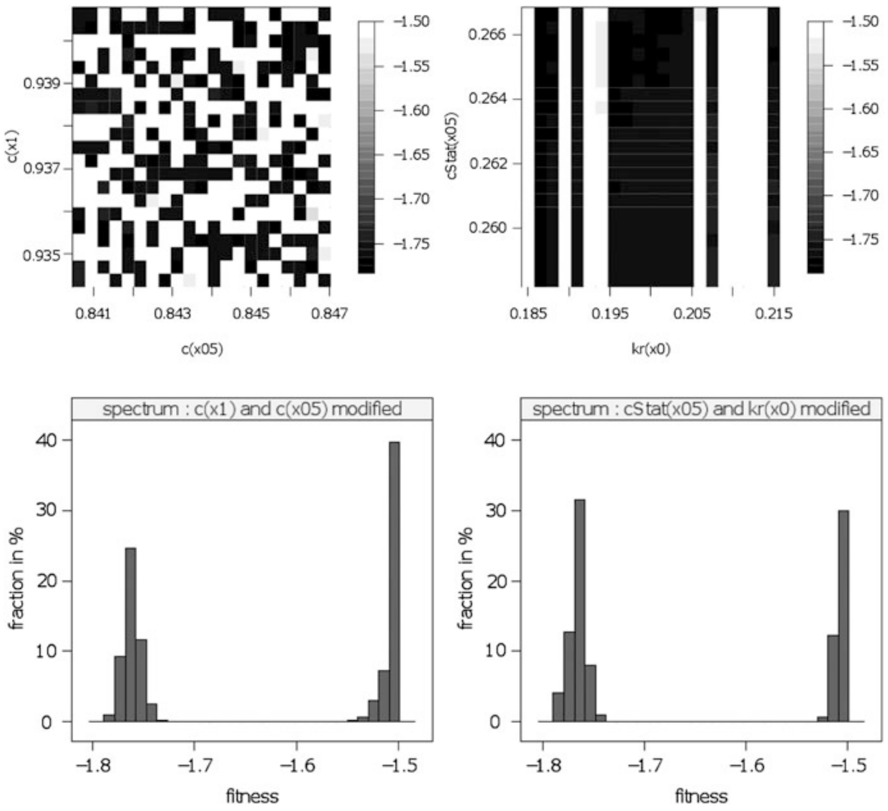


Fig. 10.11 Grid sample around a good search point identified in a prior optimization run, 20 % of the available search space in each of the two tested dimensions were covered with 21 × 21 samples. *Upper row*: Two different two-variable combinations visualized as contour plot. *Lower row*: Histograms of same data. The variables on the *left* are expected to interact strongly; the ones on the *right* should be nearly uncorrelated

10.9 Summary and Future Developments

In this chapter, we have run through the current state of experimental research in evolutionary computation as we see it. We presented severity as a metastatistical rule for evaluating statistical tests. Limitations of simply accepting or rejecting hypotheses are avoided, which refers to Problem P-1 from Sect. 10.5. If an inference could only be said to pass a test with low severity, then there fails to be evidence for this inference. Summarizing, severity provides a method that quantifies the extent of the difference from the null hypothesis that is (or is not) warranted by data $\mathbf{x}_0$. This provides an answer to Problem P-2. One important feature of the severity concept is the extension of significance level and power, which are pre-data error probabilities. In contrast to the power of a test, severity uses values from the test statistic, and enables a post-data interpretation of statistical tests. This provides an answer to problem P-3.

Active experimentation is a flexible and general framework which can be applied in many situations. Note that we do not claim that AEX is the only suitable way for tuning algorithms. Far from it! We state that AEX presents only one possible way—which might not be the best for your specific problem. We highly recommend other approaches in this field, namely F-Race [22] and REVAC [60].

We have also elaborated on the specific problems when setting up an experimental investigation in evolutionary computation, and provided hints on how to avoid the most common mistakes, and suggestions for how to write up and iterate experiments in order to concretize and validate the findings.

For cases where the problem properties are largely unknown, we suggest employing an *exploratory landscape analysis* approach which is in a relatively unstructured way often applied by practitioners already, without considering it as a working scheme of its own. Better understanding of (optimization) algorithm performance, however, needs to achieve some kind of parameter to property matching and on this path, visualization may play the key role.

As a general conclusion, we feel that still, more emphasis on experimental methodology is needed and much work is still left undone in this area. Especially, the cooperation between theory and practice should be improved, and moving towards each other may be an important task for the near future. Theory should consider current experimental results as a starting point for investigations, and established theory should be validated (e.g., concerning assumptions made) by means of structured experimental analysis.

Acknowledgements This work was supported by the Bundesministerium für Bildung und Forschung (BMBF) under the grants FIWA (AIF FKZ 17N2309), MCIOP (AIF FKZ 17N0311), and by the Cologne University of Applied Sciences under the research focus grant COSA.

References

1. T. Bartz-Beielstein, *Experimental Research in Evolutionary Computation—The New Experimentalism*. Natural Computing Series (Springer, Berlin/Heidelberg/New York, 2006)
2. T. Bartz-Beielstein, How experimental algorithmics can benefit from Mayo's extensions to Neyman-Pearson theory of testing. Synthese **163**(3), 385–396 (2008). doi:10.1007/s11229-007-9297-z
3. T. Bartz-Beielstein, Sequential parameter optimization—an annotated bibliography. CIOP technical report 04/10, Research Center CIOP (Computational Intelligence, Optimization and Data Mining), Cologne University of Applied Science, Faculty of Computer Science and Engineering Science, Apr 2010
4. T. Bartz-Beielstein, SPOT: an R package for automatic and interactive tuning of optimization algorithms by sequential parameter optimization. CIOP technical report 05/10, Research Center CIOP (Computational Intelligence, Optimization and Data Mining), Cologne University of Applied Science, Faculty of Computer Science and Engineering Science, Jun 2010. Comments: related software can be downloaded from http://cran.r-project.org/web/packages/SPOT/index.html

5. T. Bartz-Beielstein, Writing interfaces for the sequential parameter optimization toolbox SPOT. CIOP technical report 07/10, Cologne University of Applied Sciences, Cologne University of Applied Science, Faculty of Computer Science and Engineering Science, July 2010
6. T. Bartz-Beielstein, M. Preuss, CEC tutorial on experimental research in evolutionary computation, in *IEEE Congress on Evolutionary Computation, Tutorial Program*, Tutorials given at CEC 2004, San Diego and CEC 2005, Edinburgh
7. T. Bartz-Beielstein, M. Preuss, Experimental research in evolutionary computation (tutorial), in *Genetic and Evolutionary Computation Conference (GECCO 2005)*, Washington, June 2005
8. T. Bartz-Beielstein, M. Preuss, Considerations of budget allocation for sequential parameter optimization (SPO), in *Workshop on Empirical Methods for the Analysis of Algorithms, Proceedings*, Reykjavik, ed. by L. Paquete et al., 2006, pp. 35–40
9. T. Bartz-Beielstein, M. Preuss, Experimental research in evolutionary computation (tutorial), in *Genetic and Evolutionary Computation Conference (GECCO 2006)*, Seattle, July, 2006
10. T. Bartz-Beielstein, M. Preuss, Experimental research in evolutionary computation–the future of experimental research (tutorial), in *Genetic and Evolutionary Computation Conference (GECCO 2007)*, London, July 2007
11. T. Bartz-Beielstein, M. Preuss, Experimental research in evolutionary computation–the future of experimental research (tutorial), in *Genetic and Evolutionary Computation Conference (GECCO 2008)*, Atlanta, July 2008
12. T. Bartz-Beielstein, M. Preuss, Experimental research in evolutionary computation–the future of experimental research (tutorial), in *Genetic and Evolutionary Computation Conference (GECCO 2009)*, Montreal, July 2009
13. T. Bartz-Beielstein, M. Preuss, The future of experimental research, in *Experimental Methods for the Analysis of Optimization Algorithms*, ed. by T. Bartz-Beielstein, M. Chiarandini, L. Paquete, M. Preuß (Springer, Berlin/Heidelberg/New York, 2010), pp. 17–46
14. T. Bartz-Beielstein, M. Preuss, Tuning and experimental analysis in evolutionary computation: what we still have wrong (tutorial), in *Genetic and Evolutionary Computation Conference (GECCO 2010)*, Portland, July 2010
15. T. Bartz-Beielstein, M. Preuss, Automatic and interactive tuning of algorithms, in *GECCO 2011 (Companion)*, ed. by N. Krasnogor, P.L. Lanzi (ACM, New York, 2011), pp. 1361–1380
16. T. Bartz-Beielstein, K.E. Parsopoulos, M.N. Vrahatis, Design and analysis of optimization algorithms using computational statistics. Appl. Numer. Anal. Comput. Math. **1**(2), 413–433 (2004)
17. T. Bartz-Beielstein, C. Lasarczyk, M. Preuss, Sequential parameter optimization, in *Proceedings 2005 Congress on Evolutionary Computation (CEC'05)*, Edinburgh, vol. 1, ed. by B. McKay et al. (IEEE, Piscataway, 2005), pp. 773–780
18. T. Bartz-Beielstein, M. Chiarandini, L. Paquete, M. Preuss (ed.), *Experimental Methods for the Analysis of Optimization Algorithms*. (Springer, Berlin/Heidelberg/New York, 2010)
19. T. Bartz-Beielstein, M. Friese, O. Flasch, W. Konen, P. Koch, B. Naujoks, Ensemble-based modeling. CIOP technical report 06/11, Research Center CIOP (Computational Intelligence, Optimization and Data Mining), Cologne University of Applied Science, Faculty of Computer Science and Engineering Science, July 2011
20. R.E. Bechhofer, T.J. Santner, D.M. Goldsman, *Design and Analysis of Experiments for Statistical Selection, Screening, and Multiple Comparisons* (Wiley, New York, 1995)
21. C.J.P. Belisle, Convergence theorems for a class of simulated annealing algorithms. J. Appl. Probab. **29**, 885–895 (1992)
22. M. Birattari, *Tuning Metaheuristics* (Springer, Berlin/Heidelberg/New York, 2005)
23. G.E.P. Box, W.G. Hunter, J.S. Hunter, *Statistics for Experimenters* (Wiley, New York, 1978)
24. A.F. Chalmers, *What Is This Thing Called Science* (University of Queensland Press, St. Lucia, 1999)
25. C.H. Chen, An effective approach to smartly allocate computing budget for discrete event simulation, in *Proceedings of the 34th IEEE Conference on Decision and Control*, New Orleans, 1995, pp. 2598–2605

26. M. Chimani, K. Klein, Algorithm engineering: concepts and practice, in *Experimental Methods for the Analysis of Optimization Algorithms*, ed. by T. Bartz-Beielstein, M. Chiarandini, L. Paquete, M. Preuß (Springer, New York, 2010)
27. P.R. Cohen, A survey of the eighth national conference on artificial intelligence: pulling together or pulling apart? AI Mag. **12**(1), 16–41 (1991)
28. P.R. Cohen, *Empirical Methods for Artificial Intelligence* (MIT, Cambridge, 1995)
29. A.E. Eiben, M. Jelasity, A critical note on experimental research methodology in EC, in *Proceedings of the 2002 Congress on Evolutionary Computation (CEC'2002)*, Hawaii (IEEE, 2002), pp. 582–587
30. O. Flasch, T. Bartz-Beielstein, A. Davtyan, P. Koch, W. Konen, T.D. Oyetoyan, M. Tamutan, Comparing CI methods for prediction models in environmental engineering. CIOP technical report 02/10, Research Center CIOP (Computational Intelligence, Optimization and Data Mining), Faculty of Computer Science and Engineering Science, Cologne University of Applied Sciences, Germany, Feb 2010
31. T. Fober, Experimentelle Analyse Evolutionärer Algorithmen auf dem CEC 2005 Testfunktionensatz. Master's thesis, Universität Dortmund, 2006
32. T. Fober, M. Mernberger, G. Klebe, E. Hüllermeier, Evolutionary construction of multiple graph alignments for the structural analysis of biomolecules. Bioinformatics **25**(16), 2110–2117 (2009)
33. T. Fober, S. Glinca, G. Klebe, E. Hüllermeier, Superposition and alignment of labeled point clouds. IEEE/ACM Trans. Comput. Biol. Bioinfo. **8**(6), 1653–1666 (2011)
34. M. Gallagher, B. Yuan, A general-purpose tunable landscape generator. IEEE Trans. Evol. Comput. **10**(5), 590–603 (2006)
35. N. Hansen, A. Ostermeier, Completely derandomized self-adaptation in evolution strategies. Evol. Comput. **9**(2), 159–195 (2001)
36. N. Hansen, A. Auger, S. Finck, R. Ros, Real-parameter black-box optimization benchmarking 2009: experimental setup. Technical report RR-6828, INRIA, 2009
37. N. Hansen, S. Finck, R. Ros, A. Auger, Real-parameter black-box optimization benchmarking 2009: noiseless functions definitions. Technical report RR-6829, INRIA, 2009
38. J. He, C. Reeves, C. Witt, X. Yao, A note on problem difficulty measures in black-box optimization: classification, realizations and predictability. Evol. Comput. **15**(4), 435–443 (2007)
39. F. Henrich, C. Bouvy, C. Kausch, K. Lucas, M. Preuss, G. Rudolph, P. Roosen, Economic optimization of non-sharp separation sequences by means of evolutionary algorithms. Comput. Chem. Eng. **32**(7), 1411–1432 (2008)
40. J.N. Hooker, Testing heuristics: we have it all wrong. J. Heuristics **1**(1), 33–42 (1996)
41. H.H. Hoos, T. Stützle, Evaluating Las Vegas algorithms: pitfalls and remedies, in *UAI '98: Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, Madison, ed. by G.F. Cooper, S. Moral (Morgan Kaufmann, 1998), pp. 238–245
42. F. Hutter, T. Bartz-Beielstein, H. Hoos, K. Leyton-Brown, K.P. Murphy, Sequential model-based parameter optimisation: an experimental investigation of automated and interactive approaches empirical methods for the analysis of optimization algorithms, in *Experimental Methods for the Analysis of Optimization Algorithms*, ed. by T. Bartz-Beielstein, M. Chiarandini, L. Paquete, M. Preuß (Springer, Berlin/Heidelberg/New York, 2010), pp. 361–414
43. F. Hutter, H.H. Hoos, K. Leyton-Brown, K.P. Murphy, Time-bounded sequential parameter optimization, in *Proceedings of LION 2010*, Venice. LNCS, 6073 (2010), pp. 281–298
44. T. Jansen, On classifications of fitness functions, in *Theoretical Aspects of Evolutionary Computing*, ed. by L. Kallel, B. Naudts, A. Rogers (Springer, Berlin, 2001), pp. 371–386
45. D.S. Johnson, A theoretician's guide to the experimental analysis of algorithms, in *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges* (AMS, Providence, 2002), pp. 215–250
46. T. Jones, S. Forrest, Fitness distance correlation as a measure of problem difficulty for genetic algorithms, in *Proceedings of the Sixth International Conference on Genetic Algorithms*, Pittsburgh (Morgan Kaufmann, 1995), pp. 184–192

47. K. Knight, P. Langley, P.R. Cohen, What makes a compelling empirical evaluation? IEEE Intel. Syst. **11**, 10–14 (1996)
48. W. Konen, T. Zimmer, T. Bartz-Beielstein, Optimized modelling of fill levels in stormwater tanks using CI-based parameter selection schemes (in German). at-Automatisierungstechnik **57**(3), 155–166 (2009)
49. O. Kramer, B. Gloger, A. Goebels, An experimental analysis of evolution strategies and particle swarm optimisers using design of experiments, in *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, GECCO '07*, London (ACM, 2007), pp. 674–681
50. C.W.G. Lasarczyk, Genetische Programmierung einer algorithmischen Chemie. PhD thesis, Technische Universität Dortmund, 2007
51. C.W.G. Lasarczyk, W. Banzhaf, Total synthesis of algorithmic chemistries, in *GECCO '05: Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, Washington D.C. (ACM, New York, 2005), pp. 1635–1640
52. D.G. Mayo, *Error and the Growth of Experimental Knowledge* (The University of Chicago Press, Chicago, 1996)
53. D.G. Mayo, A. Spanos, Severe testing as a basic concept in a Neyman–Pearson philosophy of induction. Br. J. Philos. Sci. **57**, 323–357 (2006)
54. D.G. Mayo, A. Spanos, *Error and Inference* (Cambridge University Press, Cambridge, 2010)
55. C.C. McGeoch, Toward an experimental method for algorithm simulation. INFORMS J. Comput. **8**(1), 1–15 (1996)
56. J. Mehnen, T. Michelitsch, C. Lasarczyk, T. Bartz-Beielstein, Multi-objective evolutionary design of mold temperature control using DACE for parameter optimization. Int. J. Appl. Electromagn. Mech. **25**(1–4), 661–667 (2007)
57. O. Mersmann, M. Preuss, H. Trautmann, Benchmarking evolutionary algorithms: towards exploratory landscape analysis, in *Proceedings of the 11th International Conference on Parallel Problem Solving from Nature: Part I, PPSN'10*, Krakow (Springer, Berlin/Heidelberg, 2010), pp. 73–82
58. O. Mersmann, B. Bischl, H. Trautmann, M. Preuss, C. Weihs, G. Rudolph, Exploratory landscape analysis, in *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO '11*, Dublin (ACM, New York, 2011), pp. 829–836
59. B.M. Moret, H.D. Shapiro, Algorithms and experiments: the new (and old) methodology. J. Univers. Comput. Sci. **7**(5), 434–446 (2001)
60. V. Nannen, Evolutionary agent-based policy analysis in dynamic environments. PhD thesis, Vrije Universiteit Amsterdam, 2009
61. V. Nannen, A.E. Eiben, A method for parameter calibration and relevance estimation in evolutionary algorithms, in *Genetic and Evolutionary Computation Conference, GECCO 2006, Proceedings*, Seattle, ed. by M. Cattolico (ACM, 2006), pp. 183–190
62. J.C. Nash, *Compact Numerical Methods for Computers: Linear Algebra and Function Minimisation*, 2nd edn. (IOP, Bristol, 1990)
63. B. Naujoks, D. Quagliarella, T. Bartz-Beielstein, Sequential parameter optimisation of evolutionary algorithms for airfoil design, in *Proceedings Design and Optimization: Methods and Applications (ERCOFTAC'06)*, Berlin, ed. by G. Winter et al. (University of Las Palmas de Gran Canaria, 2006), pp. 231–235
64. J. Neyman, E.S. Pearson, On the problem of the most efficient tests of statistical hypotheses. Philos. Trans. R. Soc. A **231**, 289–337 (1933)
65. N.H. Pothmann, Kreuzungsminimierung für k-seitige Buchzeichnungen von Graphen mit Ameisenalgorithmen. Master's thesis, Universität Dortmund, 2007
66. M. Preuss, Niching prospects, in *Bioinspired Optimization Methods and Their Applications (BIOMA 2006)*, ed. by B. Filipic, J. Silc (Jozef Stefan Institute, Ljubljana, 2006), pp. 25–34
67. M. Preuss, T. Bartz-Beielstein, Sequential parameter optimization applied to self-adaptation for binary-coded evolutionary algorithms, in *Parameter Setting in Evolutionary Algorithms*, ed. by F. Lobo, C. Lima, Z. Michalewicz. Studies in Computational Intelligence (Springer, New York, 2007), pp. 91–120

68. M. Preuss, G. Rudolph, F. Tumakaka, Solving multimodal problems via multiobjective techniques with application to phase equilibrium detection, in *Proceedings of the International Congress on Evolutionary Computation (CEC2007)*, Singapore (IEEE, Piscataway, 2007)
69. M. Preuss, G. Rudolph, S. Wessing, Tuning optimization algorithms for real-world problems by means of surrogate modeling, in *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, GECCO '10*, Portland (ACM, New York, 2010), pp. 401–408
70. M. Preuss, C. Stoean, R. Stoean, Niching foundations: basin identification on fixed-property generated landscapes, in *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO '11*, Dublin (ACM, 2011), pp. 837–844
71. R.L. Rardin, R. Uzsoy, Experimental evaluation of heuristic optimization algorithms: a tutorial. J. Heuristics **7**(3), 261–304 (2001)
72. G. Rudolph, M. Preuss, J. Quadflieg, Two-layered surrogate modeling for tuning optimization metaheuristics. Algorithm engineering report TR09-2-005, Faculty of Computer Science, Algorithm Engineering (Ls11), Technische Universität Dortmund, Sept 2009
73. R. Salomon, Re-evaluating genetic algorithm performance under coordinate rotation of benchmark functions: a survey of some theoretical and practical aspects of genetic algorithms. BioSystems **39**, 263–278 (1996)
74. S.K. Smit, A.E. Eiben, Comparing parameter tuning methods for evolutionary algorithms, in *IEEE Congress on Evolutionary Computation (CEC)*, Trondheim, 2009, pp. 399–406
75. P.N. Suganthan, N. Hansen, J.J. Liang, K. Deb, Y.-P. Chen, A. Auger, S. Tiwari, Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. Technical report, Nanyang Technological University, Singapore, 2005. http://www.ntu.edu.sg/home/EPNSugan
76. A. Törn, M. Ali, S. Viitanen, Stochastic global optimization: problem classes and solution techniques. J. Glob. Optim. **14**(4), 437–447 (1999)
77. M. Tosic, Evolutionäre Kreuzungsminimierung. Diploma thesis, University of Dortmund, Jan 2006
78. H. Trautmann, J. Mehnen, Statistical methods for improving multi-objective evolutionary optimisation. Intern. J. Comput. Intell. Res. **5**(2), 72–78 (2009)
79. L. Volkert, Investigating EA based training of HMM using a sequential parameter optimization approach, in *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, Vancouver, ed. by G.G. Yen et al. (IEEE, 2006), pp. 2742–2749
80. S. Wessing, Towards optimal parameterizations of the S-metric selection evolutionary multi-objective algorithms. Algorithm engineering report TR09-2-006, Universität Dortmund, Sept 2009
81. S. Wessing, M. Preuß, G. Rudolph, When parameter tuning actually is parameter control, in *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO '11*, Dublin (ACM, 2011), pp. 821–828
82. Y. Yi, Fuzzy operator trees for modeling utility functions. PhD thesis, Philipps-Universität Marburg, 2008

Chapter 11
Formal Search Algorithms + Problem Characterisations = Executable Search Strategies

Patrick D. Surry and Nicholas J. Radcliffe

Abstract Traditional evolutionary algorithms use a standard, predetermined representation space, often in conjunction with a similarly standard and predetermined set of genetic move operators, themselves defined in terms of the representation space. This approach, while simple, is awkward and—we contend—inappropriate for many classes of problem, especially those in which there are dependencies between problem variables (e.g., problems naturally defined over permutations). For these reasons, over time a much wider variety of representations have come into common use. This paper presents a method for specifying algorithms with respect to abstract or *formal* representations, making them independent of both problem domain and representation. It also defines a procedure for generating an appropriate problem representation from an explicit *characterisation* of a problem domain that captures beliefs about its structure. We are then able to apply a formal search algorithm to a given problem domain by providing a suitable characterization of it and using this to generate a formal representation of problems in the domain, resulting in a practical, executable search strategy specific to that domain. This process is illustrated by showing how *identical* formal algorithms can be applied to both the travelling sales-rep problem (TSP) and real parameter optimisation to yield familiar (but superficially very different) concrete search strategies.

P.D. Surry (✉)
Hopper Inc, 275 Third Street, Cambridge, MA, 02142, USA
e-mail: patrick@hopper.com

N.J. Radcliffe
Stochastic Solutions Limited, 5 Atholl Crescent, Edinburgh, EH3 8EJ, UK

Department of Mathematics, University of Edinburgh, King's Buildings, EH9 3JZ, Edinburgh, UK
e-mail: Nicholas.Radcliffe@StochasticSolutions.com

11.1 Preamble

This paper is a revised and expanded version of a paper entitled *Formal Algorithms + Formal Representations = Search Strategies* by Surry and Radcliffe [25]. While still unavoidably technical, we have attempted to simplify the presentation in this version of the paper, and have added a glossary at the end that may help the reader for whom much of the terminology is new.

Since the publication of this paper, there have been a number of papers which, directly or indirectly, extend the ideas of forma analysis, or develop related ideas. The interested reader is referred, in particular, to the works of Moraglio [14], Rowe et al. [23], Jones [12], Troya and Cotta [2, 3] and Gong [9].

It is likely that some of the arguments made in this paper will seem laboured to the modern reader as attitudes within the field have changed in the 15 years since this paper was originally prepared. Although the authors have revised the paper and have to some extent modified the language, the reader is asked to make some allowances. The authors believe, and have been encouraged by the editors to believe, that the core approach outlined in this paper is, if anything, more relevant today, when a wider variety of representations are being actively used, than when the paper was originally drawn up, and that the approaches described can offer practical ways of transferring operators and insights between problem domains and representations.

11.2 Introduction

The traditional genetic algorithm (e.g., the Simple Genetic Algorithm of Goldberg [7]) was defined over a fixed representation space, namely that of binary strings. A common historical perception was that to employ such an algorithm for a new problem, one need only define a fitness function. (Indeed, standard software packages exist which literally require only computer code for a fitness function, e.g., GENESIS by Grefenstette [10].) For problems defined explicitly over binary strings (one counting, royal road, etc.) this does not present any difficulty. For others, such as real-parameter optimisation, some encoding from the problem variables into binary strings must be formulated, in order that the fitness of binary chromosomes can be calculated by decoding them. However, such "shoe-horning" may make much of the structure of the search problem unavailable to the algorithm in terms of heritable allele patterns (see, for example, Goldberg's discussion of meaningful alphabets in [8]). For problems in which candidate solutions are more complicated objects, such as the travelling sales-rep problem (TSP), a direct binary encoding may be unnatural or even infeasible. A particular case is when the "natural variables" of the problem are not orthogonal, in that the valid settings of one variable depend on the value of another (e.g., permutations). Faced with such a situation, practitioners typically adopt an ad hoc approach, drawing on evolutionary "concepts" to define pragmatic new move operators perceived as appropriate in the new domain (e.g., operators such as subtour inversion, partially matched crossover and order crossover have been devised for the TSP by Oliver et al. [15]).

Neither the use of a fixed representation space nor an ad hoc approach to each new problem are satisfactorily problem-independent, making it difficult to form meaningful comparisons between different algorithms or transfer algorithms between problem domains. (Attempting to compare, for instance, "genetic algorithms" and "simulated annealing" is futile until both a problem domain and set of move operators are specified to begin to define each algorithm precisely.) Algorithms using a fixed representation space cannot naturally be applied to nonorthogonal problems, and worse, cannot easily incorporate knowledge about the structure of the problem—one is forced to change either the growth function (the *genotype–phenotype mapping*) or the move operators, making the algorithm even less independent of problem.

In this paper we discuss a methodology by which these difficulties can be overcome. We show how we can characterise mathematically our assumptions about the relationship between fitness and the structure of the search space, and then generate a representation embedding those assumptions. We further demonstrate how (evolutionary) algorithms can be precisely specified independent of any particular representation or problem domain. In combination, this allows us to instantiate a given algorithm for *any* appropriate representation of *any* particular problem domain. This is the basis for formulating testable hypotheses about algorithm and representation quality within and across problem domains. We illustrate the technique by starting with a single formal search algorithm and mathematically deriving concrete, implementable and familiar search strategies for the disparate problem domains of real parameter optimization and the TSP.

In order to fix terminology,[1] a *search problem* is taken to be the task of solving any problem instance from a well-specified problem domain, where by solving we mean attempting to find some optimal or near-optimal solution from a set of candidate solutions. A *problem domain* is a set of *problem instances* sharing similar structure, each of which takes the form of a *search space* (of candidate solutions) together with some fitness function defined on that search space, as illustrated in Fig. 11.1. For instance, "symmetric travelling sales-rep problems" is a problem domain, within which a particular set of $n(n-1)/2$ intercity distances defines an instance (yielding a search space of $(n-1)!/2$ tours and an associated fitness function). A *search strategy* is then simply a prescription that specifies, for any problem instance, how to sample successive candidate solutions from the search space, typically biasing the samples depending on the observed quality (fitness) of previously sampled points. Any such strategy can be viewed as utilising one or more *move operators,* such as recombination, mutation and hill-climbing, that produce new candidate solutions from those previously visited.

Because the objects in the search space can be arbitrary structures (e.g., real-valued vectors, TSP tours, neural-network topologies, etc.), search strategies

[1] See also the glossary in Sect. 11.7.

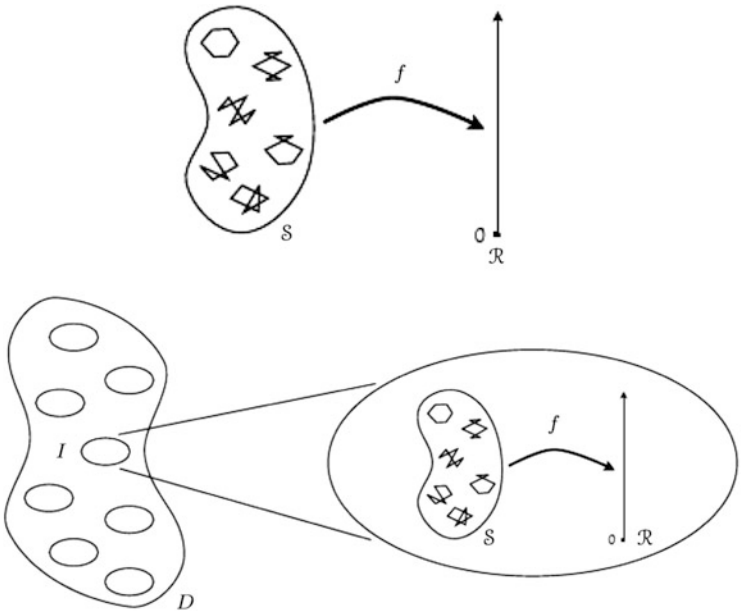


Fig. 11.1 (*Top*) A *problem instance* consists of a *search space* $\mathcal{S}$ (a set of candidate solutions), a *fitness function* f, and a space $\mathcal{R}$ from which fitness values are drawn. $\mathcal{R}$ is usually $\mathbb{R}_0^+$ (the set of non-negative reals), but can have a more complex form, including being multidimensional, sometimes with extra dimensions quantifying constraint violations. (*Bottom*) A particular problem instance I is usually but one of a large number of related search problems that together form a *problem domain* D

are necessarily problem-specific. In order to make meaningful comparisons between search strategies in different problem domains, it is helpful to define *formal search algorithms* with respect to a *formal representation* of the search space, allowing the transfer of search algorithms between problem domains. In general, a representation consists of a *representation space* and a *growth function*. The representation space defines a set of *chromosomes* which will be manipulated (by the move operators) during search, and the growth function defines a mapping between chromosomes and solutions. The move operators can then be defined on the representation space, with the quality of any chromosome determined using the growth function in conjunction with the fitness function.

Note that both the representation and algorithm are mathematical constructions and need not be directly related to the way in which the data structures and computer code for the resulting search strategy is implemented on a computer.[2] So rather than simply plugging together different bits of computer code (for representation

[2]Thus the title of this paper has been inspired by but differentiated carefully from that of Wirth's [30] and Michalewicz's [13].

and algorithm), we plug together different bits of mathematics from which we can formally *derive* a concrete search strategy that can be implemented in a well-specified way.

One obvious method to achieve the goal of problem-independent search is to fix the representation space, and then construct an appropriate growth function for each new problem domain. Such a search algorithm would sample chromosomes from the fixed representation space, using the growth and fitness functions as a "black-box" to evaluate them. This was essentially the traditional viewpoint, whereby deploying a genetic algorithm to a new problem domain simply involved finding a way to map candidate solutions on to binary strings. There are, however, strong arguments against this approach. First, it may be extremely difficult or impossible to construct an appropriate growth function. Secondly, it is increasingly recognised that effective search is only possible if search strategies incorporate appropriate *domain knowledge* (of the structure of the function being optimised); this is clearly impossible in the black-box approach.

Work on the so-called No Free Lunch Theorem (by Wolpert and Macready [31] and Radcliffe and Surry [22]) formalised these ideas, and showed that true black-box optimisation can be at most as efficient as enumerative search, despite the claims of some authors. We argue that by abstracting the definition of a search algorithm away from a fixed representation-space (just as we strove for independence from a particular problem domain), we can realise the goal of a truly problem-independent algorithm while at the same time making the role played by domain knowledge much more explicit.

In particular, our approach supports testable hypotheses about algorithm and representation quality within and across problem domains. For example, Radcliffe and Surry [20] show that the performance of typical evolutionary algorithms is highly correlated with the degree to which alleles (induced by the representation) group solutions of similar fitness (as measured by the variance of the fitness function). This points toward a more principled approach to studying the practical convergence properties of evolutionary algorithms, analogous to work on global random search such as Zhigljavsky's [32].

The formulation presented in Sect. 11.3 postulates a problem-dependent characterisation χ which captures knowledge about a problem domain. This characterisation mechanically generates a *formal representation* (representation space and growth function) for any instance of the problem, by defining a number of equivalences over the search space, as shown in Fig. 11.2. These equivalences induce subsets of the search space thought to contain solutions with related performance, possibly as *partitions* generated by equivalence relations, or simply as groups of solutions sharing some characteristic. For a given solution, the pattern of its membership of the specified subsets is used to define its alleles (and possibly genes). Although in some problems the search space can be partitioned *orthogonally* (informally, meaning that all combinations of alleles represent legal solutions), this is not always the case. For example, in the travelling sales-rep problem, natural representations involve characterising tours by the edges (links) that they share, and it is clear that an arbitrary collection of edges does not always represent a valid tour.

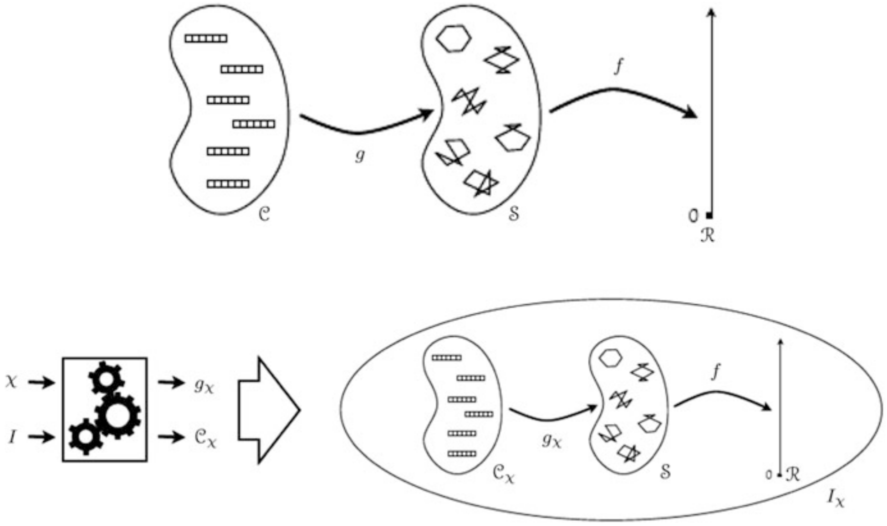


Fig. 11.2 (*Top*) The choice of representation is particularly important in evolutionary algorithms. We normally represent solutions with a *genotype* or *chromosome,* drawn from some space $\mathcal{C}$ over which the genetic operators are defined. Traditionally, this has most often been bit strings (in the case of genetic algorithms), real vectors (in the case of evolution strategies and evolutionary programming), or S-expressions (in the case of genetic programming) but can in principle be any space at least as large as the solution space $\mathcal{S}$. We then use a *growth function* g to implement the genotype-phenotype mapping from $\mathcal{C}$ to $\mathcal{S}$.
(*Bottom*) The fundamental idea with forma analysis is to generate both a representation space and a set of genetic operators for a problem domain through a mechanical procedure that takes as its input a problem domain D together with a characterization χ of that domain. We typically characterize the search domain by specifying a set of equivalences among the solutions for any instance I. These equivalences induce a *representation* made up of a *representation space* $\mathcal{C}_\chi$ (of *chromosomes*) and a *growth function* g_χ mapping chromosomes to the objects in $\mathcal{S}$. A chromosome $\eta \in \mathcal{C}_\chi$ is a string of *alleles*, each of which indicates that η satisfies a particular equivalence on $\mathcal{S}$. Algorithms can be completely specified by their action on the alleles of these generalised chromosomes, making them totally independent of the problem domain itself

In Sect. 11.4 we show how *formal algorithms* can be precisely specified. The effectiveness of these formal algorithms is a direct function of the quality of the domain knowledge captured by the allele structure generated by the characterisation. Such algorithms are themselves independent of any particular problem domain or representation; the move operators they use are defined to manipulate solutions only in terms of their abstracted subset-membership properties (alleles).

Because many problem domains are most naturally characterised using non-orthogonal representations, the traditional operators are seen not to be fully general (e.g., consider one-point crossover between permutations). We provide examples of generalisations of N-point and uniform crossover, and of mutation and hill-climbing operators, and later show how they reduce to traditional forms in familiar problem domains.

We proceed in Sect. 11.5 to show how *any* formal algorithm can be instantiated with *any* suitable representation of a problem domain of interest to produce a concrete *search strategy*. This is illustrated by defining a simple representation-independent evolutionary algorithm and instantiating it, on the one hand, to solve the TSP and, on the other, to solve a real-parameter optimisation problem. The resulting search strategies for the two problems look very different from each other but are both similar to evolutionary algorithms commonly applied in their respective domains. We thus prove the surprising result that two apparently quite different algorithms, in two completely different problem domains, are in fact *identical,* in a strong mathematical sense.

In Sect. 11.6 we summarise the implications of this more formal approach. We see that by separating algorithm and representation, we achieve the goal of truly problem-independent algorithms. This separation also makes the role of domain knowledge in the search process much more explicit, allowing us to pose more carefully questions such as "What is a good algorithm given certain properties of the characterisation?" and "What is a good characterisation of a given problem domain?" Although this formalism might be argued to contain a certain degree of circularity, it is seen to yield practical benefits. For instance, we are able to transfer such algorithms between arbitrary problem domains and to compare different algorithms fairly, independent of a particular problem.

11.3 Formal Representations

In tackling a domain of search problems, we often prefer to search over a set of structures (chromosomes) representing the objects in the search space rather than directly over the objects themselves. Use of such a representation (made up of a representation space and associated growth function) makes the search algorithm much more generic. A general method for defining a representation is to classify subsets of solutions according to characteristics which they share.

Holland [11] used exactly this approach. He identified subsets of a search space of binary strings using *schemata*—sets of strings that share particular bit values. His Schema Theorem shows how the *observed* fitness of any schema in a population can be used to bound the *expected* instantiation of the same schema in the next generation, under the action of fitness-proportionate selection. Several authors then generalised the notion of a schema and have shown that the theorem applies to arbitrary subsets of the search space, provided that suitable disruption coefficients are chosen (see Radcliffe [16] and Vose and Liepins [28]).

In particular, Radcliffe [16, 17] has developed the idea of *forma analysis,* in which general subsets of the search space are termed *formae*. Typically, the formae are defined as the equivalence classes induced by a set of equivalence relations, although this need not be the case. Any solution can then be identified by specifying the equivalence class to which it belongs for each of the equivalence relations (provided the set of relations is sufficiently rich). Loosely speaking, we identify genes with a set of basic equivalence relations and alleles with the corresponding equivalence classes. For instance, in a search space of faces, "hair colour" and "eye colour" might be two basic equivalence relations, which would induce the formae "red hair", "brown hair", "blue eyes", etc. Higher-order formae are then constructed by intersection, e.g., "brown hair and blue eyes". Chromosomes made up of strings of alleles can then be used to *represent* the original structures of the search space (faces in our example). These chromosomes make up the representation space and the objects they encode define the growth function. In certain cases, the genes are orthogonal, meaning that any combination of allele values represents a valid solution, but in many cases this is not so (certain alleles are incompatible).

It is also not always easy to define equivalence relations; in these cases we simply identify particular subsets of the search space that share some characteristic. In such cases, genes are not defined and a chromosome consists simply of a set of alleles. For instance, in the TSP, we could identify $n(n-1)/2$ subsets of the search space, each containing all tours in which city i is linked to city j. Then a particular tour would be represented by the set of (undirected) edges it contained. Although in this case each chromosome would have the same number of alleles, this need not be so, and the ideas of formal representations and operators generalise easily to variable-length chromosomes [18]. This and other TSP representations are discussed further in Sect. 11.5.1.

For any problem domain we require a (problem-dependent) *characterisation*: a mathematical procedure for generating the equivalences between candidate solutions that induce both the (formal) representation and the growth function for any problem instance (see Fig. 11.2). The characterisation explicitly captures all of the structure that will be exploited by a search algorithm, allowing us (for example) to test alternative hypotheses about what makes a "good" representation for a particular algorithm. The selection of an appropriate characterisation for a particular problem domain is an open problem. However, several *design principles* have been previously proposed by Radcliffe [16]. The most important of these is that the generated formae should group together solutions of related fitness [20], in order to create nonrandom structure which can be exploited by the move operators. We also require that it is possible to find a member of any given formae in reasonable time without resorting to enumeration, but this is true of most reasonable characterisations.

Examples of several representations designed for various problem domains are shown in Table 11.1. These include the traditional binary representation for real parameters, the Dedekind representation for real parameters introduced in [26], two natural representations for the TSP (for comparisons of these and others see [20]), and two representations for subset-selection problems (used in neural-network topology optimisation), one in which only set membership is considered to be important and one in which both membership and non-membership is used [18].

Table 11.1 This table summarises the characteristics of several representations for different problem domains (see Sect. 11.5.1 for TSP representations; Sect. 11.5.2 for real-parameter representations; and see [18] for subset representations). Basic formae indicates the way in which basic subsets of the search space are identified, and the existence of genes is noted. Orthogonal representations are those in which any combination of alleles defines a valid solution. Degeneracy occurs when multiple chromosomes represent the same solution, and redundancy measures the amount of excess information in the chromosome (i.e., the number of alleles that could be removed while still identifying a unique solution)

Representation	Basic formae	Genes?	Orthogonal?	Degeneracy	Redundancy
Binary-coded reals	Same value for ith bit	Yes	Yes	None	None
Dedekind real parameters	Same side of cut at value x	Yes	No	None	Huge
TSP: permutations	Same city in position i	Yes	No	$\times 2n$	Low
TSP: undirected edges	Both contain link $\overline{jk}$	No	No	None	Low
Subset-selection: inclusive	Both include ith element	No	Yes	None	None
Subset-selection: incl/excl	Both include or both exclude ith element	Yes	Yes	None	None

11.4 Formal Algorithms

Traditional evolutionary algorithms are typically defined using a set of move operators that assume a particular form of the representation space. For example, many genetic algorithms assume chromosomes are binary strings, and most evolution strategies assume chromosomes are strings of real parameter values. Although some of the operators used by such algorithms can be generalised straightforwardly to related representation spaces (for example, N-point crossover between binary strings is easily generalised to k-ary chromosomes), they typically are not general enough to handle arbitrary representations. In particular, variable-length genomes and nonorthogonal representations both present difficulties, and have generally led in the past to ad hoc construction of problem-specific move operators (for example, in the TSP).

We will specify formal algorithms by defining formal move operators. These are defined by specifying how they manipulate the chromosomes' alleles (equivalence classes) for *any* formal representation derived from a suitable problem characterisation. Such algorithms are completely independent of representation, and we will show in Sect. 11.5 that they can be applied to any problem domain by mathematically deriving problem-specific instances of the formal move operators using a representation appropriate to that domain.

A number of design principles have been proposed to facilitate the development of simple structure-preserving move operators. This has led to the definition of a number of *representation-independent* recombination and mutation operators, permitting the construction of truly representation-independent algorithms. These design principles [16, 19] and associated operators include the following.

11.4.1 Respect

Respect requires that children produced by recombination are members of all formae to which both their parents belong. For example, if our representation included equivalence relations for hair colour and eye colour, then if both parents had red hair and green eyes, so should all of the children produced by a respectful crossover operator.

R^3. Random respectful recombination is an example of a recombination operator defined in terms of a formal representation. It is defined to be that operator which selects a child uniformly at random from the set of all solutions which share all characteristics possessed by both parents (their *similarity set*).

11.4.2 Transmission

A recombination operator is said to be *strictly transmitting* if every child it produces is equivalent to one of its parents under each of the basic equivalence relations (loosely, a child shares each gene value with at least one of its parents). Thus, if one parent had red hair and the other had brown hair, then transmission would require that the child had either red or brown hair.

RTR. The random transmitting recombination operator is defined as that operator which selects a child uniformly at random from the set of all solutions belonging only to basic formae present in either of the parents (their *dynastic potential*).

11.4.3 Assortment

Assortment requires that a recombination operator be capable of generating a child with any compatible characteristics taken from the two parents. In our example above, if one parent had green eyes and the other had red hair, then if those two characteristics are compatible, assortment would require that we could generate a child with green eyes and red hair.

RAR. The random assorting recombination operator, a generalised form of uniform crossover, has been previously defined [18]. It proceeds by placing all alleles from both parents in a conceptual bag (possibly with different multiplicities), and then repeatedly draws out alleles for insertion into the child, discarding them if they are incompatible with those already there. If the bag empties before the child is complete, which can happen if not all combinations of alleles are legal (so that the representation is *nonorthogonal*) remaining genes are set to random values that are compatible with the alleles already present in the child.

All of R^3, RTR and RAR may be viewed as generalizations of uniform crossover, in the sense that they all reduce to uniform crossover in some cases, yet all handle non-orthogonal representations.

GNX. A generalised version of N-point crossover has also been defined by Radcliffe and Surry [20]. This proceeds in much the same way as standard N-point crossover, dividing the two parents with N cut-points, and then using genetic material from alternating segments. The alleles within each segment are tested in a random order for inclusion in the child, and any remaining gaps are patched by randomly selecting compatible alleles first from the unused alleles in the parents, and then from all possible alleles.

11.4.4 Ergodicity

This demands that we select operators such that it is possible to move from any location in the search space to any other by their repeated action. (Typically a standard mutation operator is sufficient.)

BMM. Binomial minimal mutation, a generalisation of standard point-wise mutation, has been proposed in [20]. Minimal mutations are defined to be those moves that change the fewest possible number of alleles in a solution (in nonorthogonal representations it may be necessary to change more than one allele at a time to maintain legality). BMM performs a binomially-distributed number (parameterised by the genome length and a gene-wise mutation probability) of minimal mutations, and does not forbid mutations which 'undo' previous ones.

Hill-climbers. The definition of representation-independent "minimal mutation" allows us to define a number of representation-independent hill-climbing operators, and to define *memetic algorithms* based on the idea of searching over a subspace of local-optima [21].

11.4.5 Example

Using the above operators, we can define algorithms that are independent from any particular representation or problem, such as the example shown in Fig. 11.3. Note that every step of the algorithm is precisely defined, and that given a representation of a problem domain, we can mathematically derive a concrete search strategy suitable for implementation on a computer (see Sect. 11.5). This is different from traditional evolutionary algorithms, in which steps 4 and 5 would have to be modified for any problem domain which required a new representation space.

1. Generate an initial population by randomly sampling p times from the space of chromosomes.
2. Evaluate the p members of the initial population via the growth and fitness functions.
3. Select two parents using binary-tournament selection.
4. Recombine the parents using RAR.
5. Mutate the resulting child using BMM.
6. If the child does not exist in the population, evaluate it and replace the member having the worst fitness.
7. Repeat to step 3 until termination criterion.

Fig. 11.3 A representation-independent evolutionary algorithm: This example shows how representation-independent move operators allow us to define specific evolutionary algorithms precisely (in a mathematical sense) without first choosing a problem domain or representation

11.5 Search Strategies

In order to construct a practical search strategy for a given problem domain, we simply combine a formal algorithm with an appropriate representation of the problem domain. There is no need to construct new move operators, as we can merely instantiate those defined in the formal algorithm of choice. Since *exactly* the same formal algorithm (for example, that shown above) can be instantiated for two different representations (of either the same or different problem domains), one can begin to make more definite statements about the quality of the algorithm itself as it is defined independently of any problem. We can fix the representation and vary the algorithm, allowing more meaningful comparisons between algorithms, or conversely, fix the algorithm to compare several alternative representations for a given problem domain.

For several of the representations shown in Table 11.1, the generalised operators defined in Sect. 11.4 reduce to traditional domain-specific variants. For example, for any orthogonal representation, R^3, RTR and RAR all reduce to uniform crossover [27], GNX reduces to N-point crossover, and BMM becomes simple gene-wise point mutation.

It is reassuring that algorithms defined using these formal operators reduce to commonly used search strategies in the relevant problem domains. To illustrate, the algorithm shown in Fig. 11.3 is instantiated in Fig. 11.4 for both the travelling sales-rep problem using the undirected-edge representation, and for a real-parameter function optimisation problem using the Dedekind representation. This results, on the one hand, on a strategy based on edge-recombination and sub-tour inversions, and, on the other, in one based on blend-crossover and Gaussian creep-mutation. Both of these strategies have been widely used in their respective domains, but it was not clear before now that they were exactly the same formal algorithm.

Again recall that the representation and algorithm described here are abstract constructions, used only to derive mathematically the search strategy that will be eventually implemented computationally. For example, the (formal) Dedekind representation for real numbers has (in the limit) an infinite number of genes, yet it

Problem domain:	**TSP**	**Real-parameter opt.**
Representation:	**Undirected-edges**	**Dedekind**
Representation type:	**Allelic**	**Genetic**
Choose initial population:	of random tours	of random vectors
Evaluate each solution:	by measuring tour length	using provided $f(\mathbf{x})$
Select two parents:	using binary-tournament selection	
Recombine parents using:	variant of edge-recomb.	BLX-0(blend crossover)
Mutate the child with:	binomial number of sub-tour inversions	Gaussian creep-mutation for each parameter
Evaluate, replace worst:	if the child does not exist in the population	
Repeat:	until termination criterion	

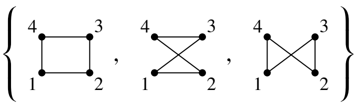
Fig. 11.4 Search strategy as algorithm plus representation: When the illustrative representation-independent algorithm of Fig. 11.3 is instantiated with characterisations of two diverse problem domains (TSP at *left*; real-parameter optimisation at *right*), we derive familiar, problem-specific search strategies

is a simple matter mathematically to derive forms of the various operators suitable for efficient (and finite!) implementation. (More recent work by Gong [9] has looked more at questions of generating physical representations and operators, rather than specifying them mathematically, as is the focus here.)

11.5.1 Travelling Sales-Rep Optimization

Here the problem domain, D, comprises all (symmetric) TSPs. Each problem instance consists of a particular collection of cities, so that the search space $\mathcal{S}$ is the set of all possible tours: all *non-equivalent* paths that visit every city exactly once. For example, given a set of four particular cities labelled 1–4, chosen for illustration to sit at the corners of a square,

$\mathcal{S} =$



(11.1)

In general, a TSP over n cities has an associated search space of size $n!/2n = (n-1)!/2$, arising from the $n!$ different permutations of the city labels and the $2n$ equivalent forms for any tour.

11.5.1.1 The Permutation Representation

One convenient way to represent a tour is by listing the sequence of city labels in the order in which they are visited, so that the second tour shown in the set might be represented by 1243. This is called the *permutation representation,* denoted π.

For any particular representation ρ, we distinguish between the representative of a solution under ρ (the formal chromosome, or genotype) and the solution itself (the phenotype). The set of all chromosomes in this representation will be denoted $\mathcal{C}^{\rho}$. Given a chromosome $\eta \in \mathcal{C}^{\rho}$, and the solution $x \in \mathcal{S}$ to which it corresponds, we write η^{ρ} to mean "the solution represented by η in representation ρ". Thus η^{ρ} is a member of $\mathcal{S}$, and in the current example $\eta^{\rho} = x$. For our four-city TSP, we can thus write

$$\mathcal{S} = \{1234^{\pi}, 1243^{\pi}, 1324^{\pi}\}. \tag{11.2}$$

The permutation representation is a formal genetic representation, where the ith gene identifies the ith city in the tour. Here we have characterised the problem domain using basic equivalence relations that group together tours in which a specified city is visited at a prescribed position in the tour. As noted above, this characterisation results more from the natural way in which tours are written down rather than from any underlying belief about the structure of the cost function. Composition of these basic equivalence relations lead to formae that contain tours in which particular cities are visited at specified positions in the tour (possibly familiar to most readers as the o-schemata of Goldberg and Lingle [6]). For example, in our four-city example, the forma ξ containing all tours that have city 1 and 2 in the first and second positions respectively would be given by

$$\xi = \{1234^{\pi}, 1243^{\pi}\} \tag{11.3}$$

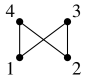
corresponding to the o-schema $12\square\square$ (where $\square$ is a wildcard or "don't care" indicator).

The representation-independent RAR$^{\pi}$ and GNX$^{\pi}$ operators are straightforward to implement for this representation, while the definition of BMM$^{\pi}$ depends on the observation that the minimal mutation of a permutation is generated by exchanging the positions of two cities [20]. However, we find that the resulting domain-specific operators are not particularly effective, precisely because the permutation-based characterization does not capture very useful information about solution structure and fitness.

11.5.1.2 The Undirected Edge Representation

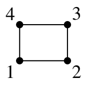
When we instead consider the structure of the cost function for the TSP, it is clear that it is the connections between cities—the edges contained in the tour—that determine its cost. This suggests that a characterisation of the problem domain based on edges might be more effective than the permutation representation. In the *undirected edge representation*, denoted u, we choose the characterisation χ_u that identifies subsets of tours having a link (edge) between any two specified cities. For example, the tour 1324^π contains the (undirected) edges $\overline{13}$, $\overline{23}$, $\overline{24}$ and $\overline{14}$, so that

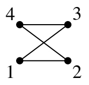
$$1324^\pi = \{\overline{13}, \overline{23}, \overline{24}, \overline{14}\}^u = \quad . \tag{11.4}$$



Any solution can be represented by the set of undirected edges that it contains, as χ_u generates formae comprising all tours containing a specific set of edges. For example, the forma ξ consisting of those tours that contain the $\overline{12}$ edge is

$$\xi = \{ \quad , \quad \}. \tag{11.5}$$





Note that in this case χ_u does not generate equivalence relations: we call the representation *allelic* because genes are not directly defined, and a solution is simply a collection of alleles. (Although we could induce a *genetic* representation by defining a basic equivalence relation for each of the $n(n-1)/2$ edges a tour could contain—generating a binary chromosome with $n(n-1)/2$ genes—the actions of the move operators would then tend to be dominated by the many edges *not* contained by the tour. See also [5] where this representation is employed.)

RAR^u, while somewhat harder to implement than for the permutation representation, follows straightforwardly from its definition, becoming a variant of edge recombination by Whitley et al. [29] and R^3 reduces to a weaker version of the same operator [20].

To instantiate BMM^u, we note that the minimal mutation for this representation is the reversal of a subtour. For example,

$$\{\overline{14}, \overline{34}, \overline{35}, \overline{25}, \overline{26}, \overline{16}\}^u = 143526^\pi \overset{\text{BMM}^u}{\longmapsto} 125346^\pi = \{\overline{12}, \overline{34}, \overline{35}, \overline{25}, \overline{46}, \overline{16}\}^u, \tag{11.6}$$

reversing the section from positions 2–5 inclusive. Minimal mutations are thus at distance 2 from their parents in this representation (as sub-tour reversal breaks two edges), allowing us to implement BMM^u directly.

Thus instantiating the representation-independent algorithm shown in Sect. 11.4 results in the domain-specific search strategy shown at the start of this section.

11.5.2 Real-Parameter Optimization

Many optimisation problems are formulated as a search for vectors of real-valued parameters that form extrema of some function. A variety of both local and global techniques with varying degrees of specialisation have been proposed for tackling such problems. Evolutionary algorithms have also been regularly applied in these domains, a particular attraction being that they require only the ability to evaluate the function at any point. Indeed, *evolution strategies* have been primarily focused on real-parameter optimisation, with theoretical results specialised to this domain. Traditional genetic algorithms, on the other hand, were historically applied to such problems by mapping to a canonical representation space of binary strings for which simple operators are defined. Typical "practical" genetic algorithms specialise these operators by considering the phenotypic effects of the moves they generate in the search domain of real parameters.

We show that by explicitly designing representations that capture beliefs about the structure of the search domain of real parameters (such as the importance of locality and continuity), we can instantiate representation-independent mutation and recombination operators to derive commonly used operators such as blend crossover, line recombination and Gaussian mutation.

Although real-parameter optimization is conceptually based on a continuous search-space, we can extend our work on formal construction of representations in discrete (typically combinatorial) search problems by constructing a limiting sequence of discrete representations that form an increasingly accurate approximation to the continuous space [24]. In this approach, we approximate each real parameter x by an integer $i \in \mathbb{Z}_n$ (the integers, modulo n) with $n \to \infty$, which extends naturally to higher dimensions (multiple real parameters).

11.5.2.1 Traditional Binary Coding

Historical approaches to binary coding of real parameters were based (if on any explicit foundation!) on the belief that more schemata are better than fewer (the notion of implicit parallelism, giving rise to the principle of minimal alphabets). This has repeatedly been shown to be little more than statistical sleight of hand: one sample is one sample, not many. There is also perhaps some idea that because binary coding "chops up the search space" in many different ways, it lets the algorithm discover useful patterns [7,11] but research has shown that is only true if the problem happens to coincide with the particular scaling and location captured in the binary coding. For example, Eshelman and Schaffer [4] found that simply rescaling or shifting the coordinate axes could dramatically affect performance. (It is reasonable, however, to speculate about constructing a formal representation based on beliefs about periodicity in the objective function—or, indeed, on any other feature thought to be relevant—but this has not as yet been achieved, although an early attempt was made by Radcliffe [16].)

11.5.2.2 The Dedekind Representation

In generic real-parameter optimisation, it would seem that a more desirable characterization of the problem domain would capture the idea that small changes in the

Table 11.2 This table compares the Dedekind representation with traditional binary-coding for a single real-valued parameter. The real parameter is first approximated by mapping to $\mathbb{Z}_n$ (with $n \to \infty$ in the limit). Although the Dedekind representation is highly non-orthogonal with $n - 1$ genes, compared to the $\lceil \log_2 n \rceil$ required for traditional binary coding, we never physically store or manipulate the chromosome in this form; it is simply a mathematical device used to derive appropriate domain-specific operators that encapsulate our beliefs about the problem structure

$x \in [0, 1)$	$i \in \mathbb{Z}_8$	Dedekind	Traditional binary
0.000	0	0000000	000
0.125	1	1000000	001
0.250	2	1100000	010
0.375	3	1110000	011
0.500	4	1111000	100

parameters lead to small changes in the observed function values (e.g. see [32]). That is to say, we might believe that neighbouring solutions in the search space are likely to have related performance. In evolution strategies, this belief is termed *the principle of strong causality*, and in real analysis functions with such a property are termed *Hölder continuous* or *Lipschitz*.

In order to quantify this belief, we identify groups of solutions (formae) based on locality using the idea of *Dedekind cuts*[3] on each parameter (approximated as an integer in $\mathbb{Z}_n$). Here, two solutions are equivalent with respect to the ith cut for a given parameter if they lie on the same side of i on the $\mathbb{Z}_n$ number line. Table 11.2 compares the Dedekind and traditional binary-coded representation for a single parameter with $n = 8$.

Note that, formally, the Dedekind representation has $n - 1$ highly nonorthogonal (constrained) binary genes coding a single approximated parameter, instead of the $k = \lceil \log_2 n \rceil$ orthogonal genes of traditional integer coding or Gray coding. However, we will see that the operators we derive from this representation and their limiting behaviour as we increase the fidelity of approximation ($n \to \infty$) are much more natural for the problem domain.

While it is somewhat ironic that this formalism of the real representation utilises binary genes, we emphasise once again that we do *not* propose to store or manipulate solutions in this form, but only to apply our design principles to (mathematically rather than computationally) develop and analyse our operators. Indeed, we strongly advised *against* storing the solutions as physically described in by the Dedekind representations; in our own code, we store real parameters as floating native point values in our chosen implementation language.

It is now straightforward to derive forms of the representation-independent genetic operators described in Sect. 11.4. For the traditional binary coding, the

[3]A Dedekind cut is a partitioning of the rational numbers into two non-empty sets, such that all the members of one are less than all those of the other. For example, the irrationals are formally defined as Dedekind cuts on the rationals (e.g. $\sqrt{2} \triangleq \langle \{x \mid x^2 > 2\}, \{x \mid x^2 < 2\} \rangle$).

generalised operators reduce to "standard" forms, since the representations are orthogonal (all combination of allele values are legal). Thus, RAR, RTR and R^3 reduce to uniform crossover, GNX reduces to N-point crossover, and BMM reduces to bitwise point mutation [26].

For the Dedekind representation, it is clear that both R^3 and RTR require that the child be uniformly selected to lie in the box determined by the parents. It is clear that in the limit of $n \to \infty$, this is equivalent to blend crossover with parameter $\alpha = 0$ (BLX-0), as defined by Eshelman and Schaffer [4], and widely used in evolution strategies [1].[4] For this representation, it is not difficult to see that RAR is also equivalent to BLX-0 (it is easy to show that the child must lie in the interval defined by the parents, and only slightly more difficult to demonstrate that the likelihood is uniform over the interval).

Turning to mutation, if we analyse our representation-independent mutation operator BMM with the Dedekind representation, it is clear that a minimal mutation involves flipping the value of one of the two bits forming the transition from ones to zeros in the genome. Thus a fixed-length sequence of minimal mutations is equivalent to a random walk away from the original transition point. We can show that this reduces in the limit of $n \to \infty$ to standard Gaussian creep mutation with width parameterised by the gene-wise mutation probability [26].

The Dedekind representation results from treating each real parameter (dimension) of a multiparameter search space independently. By instead considering all possible axis orientations simultaneously we derive the Isodedekind ("Isotropic Dedekind") representation [26]. The operators we derive from this representation again reduce to forms familiar from evolution strategies.

Once again, we have demonstrated that we can formally derive a domain-specific search strategy—this time for real-parameter optimization—from a representation-independent algorithm. This allows us to study directly the impact of representation on algorithm performance. Indeed Fig. 11.5 illustrates how exactly the same formal algorithm can exhibit very different characteristics for a specific optimization problem when representation is the only variable.

11.6 Summary

This paper has presented a more formal approach to evolutionary search, by separating a search strategy into a representation and an algorithm. We have introduced a disciplined methodology for attacking new problem domains—instead of simply using evolutionary "ideas" to invent new operators, one need only provide a *characterisation* of the problem that explicitly captures beliefs about its structure, and then instantiate an existing algorithm with the derived representation. This applies equally to problems with nonorthogonal representations where

[4]BLX-0 is perhaps now more commonly known as *box crossover.*

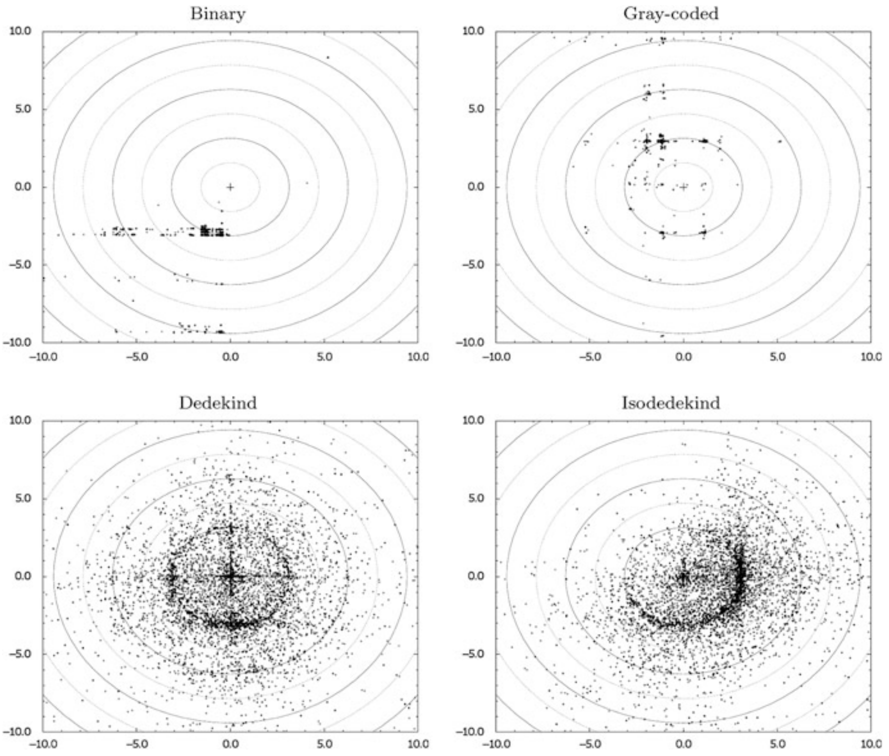


Fig. 11.5 In the spirit of the work presented in [4] by Eshelman and Schaffer, this figure illustrates the results of applying an identical formal algorithm instantiated with the four different real-parameter representations discussed by Surry [24] to Schaffer's F6 function. The two-dimensional function is radially symmetric with global maximum at $(0, 0)$, local maxima on circles of radius $\pi, 2\pi, \ldots$, and local minima on circles of radius $\pi/2, 3\pi/2, \ldots$ [4]. The search domain is $[-100, 100] \times [-100, 100]$ of which the figures show the central region. Each figure shows all of the points sampled in one typical run of a fixed algorithm based on the R^3 and BMM operators. Each algorithm sampled approximately 9,000 points in the central region during a run of 100 generations with population size 100. The binary representation (*top-left*) exhibits extremely poor coverage and an extremely striated sampling pattern based on the relative periodicity in the function and the representation. A *Gray-coded* representation (*top-right*) typically shows better coverage, as mutation is more effective, but the sampling pattern is clearly biased. The Dedekind representation (*bottom-left*) shows much better coverage, but since R^3 reduces to BLX-0 there is still a tendency to favour the axis directions, and an inward bias on the population. The algorithm based on the Isodedekind representation (*bottom-right*) still shows the inward population bias since R^3 reduces to line recombination, but the axial skew is removed

traditional evolutionary algorithms are inapplicable. We have demonstrated, by way of example, that identical algorithms can be applied to both the TSP and real parameter optimisation, yielding familiar (but apparently quite different) concrete search strategies.

Because these formal algorithms are independent of any particular representation, it is possible to transfer them to arbitrary problem domains, and to make meaningful comparisons between them. By making the role of domain knowledge more explicit we are also directed to more reasoned investigation of what makes a good representation for a given problem. Further investigations will build on these ideas to construct a more complete taxonomy of representations, and to investigate issues of algorithmic performance and quality of representation.

11.7 Glossary

Allele. If genes are defined, the alleles for a gene are its possible values. For example, in the directed edge representation for the TSP, there is a gene for each (originating) city, and the alleles are the possible destinations that the tour can next visit. If genes are not defined, we use the term allele to refer to the smallest specifiable values that together make up a chromosome, usually as a set. For example, in the undirected edge representation of the TSP, a tour is defined simply as a set of (undirected) edges, e.g. $\{\overline{12}, \overline{23}, \overline{34}, \overline{41}\}$. We refer to these edges as alleles, but the alleles do not correspond to any gene (something true geneticists would probably regard as an abuse).

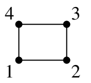
Basic Equivalence Relation. A basis in linear algebra is a set of vectors from which others may be formed as linear combinations. Similarly, a set of equivalence relations can have a basis, from which others may be formed by conjunction (logical and). Basic equivalence relations play the role of formal genes in forma analysis.

Basic Formae; Basic Equivalence Classes. The equivalence classes induced by basic equivalence relations ("genes"). Basic equivalence classes play the rôle of formal alleles in forma analysis.

Chromosome. See *genotype.*

Characterisation. A characterisation of a *problem domain* is a mathematical procedure for constructing *a formal representation* for any *problem instance* in that domain. It captures our assumptions about the relationship between the *fitness function* and the structure of the *search space.*

Degeneracy. A degenerate representation is one in which two or more chromosomes represent the same solution. For example, in the permutation representation for the symmetric TSP, 1234, 2341 and 4321 (among others) all represent the same solution, namely



.

Equivalence Relation. An equivalence relation $\sim$ is a relation between members of a set that exhibits reflexivity ($x \sim x$), symmetry ($x \sim y \Longrightarrow y \sim x$) and transitivity ($x \sim y$ and $y \sim z \Longrightarrow x \sim z$). Real biological genes can be thought of as examples of equivalence relations (e.g. eye colour). Equivalence relations partition spaces into sets of equivalent solutions called *equivalence classes;* the equivalance classes for eye colour might be "green", "blue" and "brown".

Fitness Function. The objective function, or quality measure, for solutions.

Forma. A forma is a subset of the search space, often an equivalence class induced by an equivalence relation. A forma is a generalization of the notion of a schema.

Formae. The plural of forma is formae.

Formal Algorithm. A formal (search) algorithm is a well-specified mathematical procedure for sampling a *search space,* based on *representation-independent move operators* and the *fitness* values observed for candidate solutions. It can be thought of as parameterized by a *formal representation*: once the (formal) algorithm and representation are chosen, an implementable *search strategy* can be derived.

Formal Representation. A formal representation is a conceptual data structure that allows (formal) operators to be defined. The representation is often derived using a *characterisation* of a *problem domain.* We typically distinguish between *genetic* and *allelic* representations.

Gene. A gene is a variable in a representation, which can take on some well-defined range of values and which has a well-defined meaning. For example, the *eye-colour gene* specifies eye colour, and might have legal values (alleles) green, blue and brown.

Genotype. The chosen representation of the problem, usually as some kind of string of "genes". This is also known as a chromosome.

Genotype–Phenotype Mapping. See *growth function.*

Growth Function. A function that transforms a *genotype* (or *chromosome*) into a solution (or *phenotype*). Also known as the *genotype-phenotype mapping.*

Move Operator. A move operator is a prescription for generating a new candidate *solution* to a *problem instance* from one or more existing solutions and (sometimes) their *fitness* values. Examples include recombination, mutation and hill-climbing operators. We distinguish *representation-independent* move operators used in *formal algorithms*, which can be instantiated mathematically for any *formal representation*, from the resulting problem-specific move operators employed in *search strategies.*

Orthogonal Representation. A representation is orthogonal if each of its parameters may be set independent of the value of any of the others, i.e. if there are no "invalid" chromosomes.

Partitions. A partitioning of a set is a collection of disjoint subsets (partitions) covering the set.

Phenotype. The candidate *solution* to the problem. Phenotypes are represented and specified by a given *genotype* or *chromosome* but are distinct from them. In nature, an actual organism is a phenotype, whereas the organism's genotype is its DNA.

Problem Domain. A problem domain is a collection of related *search problems,* for example symmetric travelling sales-rep problems (TSP).

Problem Instance. A problem instance is a specific example from a *problem domain*, consisting of a *search space* and *fitness function.* For example, a set of $n(n-1)/2$ intercity distances for a TSP.

Redundancy. A representation is redundant if the value of one gene can be deduced from the values of others, or equivalently, if not all of the gene values are required to specify a solution uniquely. Redundant representations cannot be orthogonal, but nonorthogonal representations do not have to be redundant.

Representation. The genotype of a natural organism specifies a corresponding phenotype (physical organism) that can be constructed by following a well-defined set of steps starting from the DNA; in this sense, the genotype may be said to "represent" the organism. In a similar way, we typically manipulate and work with some kind of an encoding or *representation* of the actual objects in the search space when we use evolutionary algorithms.

Search Algorithm. See *formal algorithm.*

Search Problem. A search problem is the task of finding (near) optimal *solutions* to a *problem instance.*

Search Space. The set of candidate solutions (*phenotypes*) to a given *search problem.*

Search Strategy. A *search strategy* is a concrete computational approach to finding high *fitness* solutions from a given *problem instance.* This paper demonstrates that search strategies can be derived mathematically given a *characterisation* of the *problem domain* and a *formal algorithm.*

Solution. A (candidate) solution to the problem; a member of the *search space;* a *phenotype.*

References

1. T. Bäck, F. Hoffmeister, H.-P. Schwefel, A survey of evolution strategies, in *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Diego (Morgan Kaufmann, San Mateo, 1991), pp. 2–9
2. C. Cotta, J. Troya, Genetic forma recombination in permutation flowshop problems. Evol. Comput. **6**, 25–44 (1998)

3. C. Cotta, J. Troya, On the influence of the representation granularity in heuristic forma recombination, in ed. by J. Carroll, E. Damiani, H. Haddad, D. Oppenheim, *ACM Symposium on Applied Computing 2000*, Villa Olmo (ACM, 2000) pp. 433–439
4. L.J. Eshelman, D.J. Schaffer, Real-coded genetic algorithms and interval schemata, in ed. by D. Whitley, *Foundations of Genetic Algorithms 2* (Morgan Kaufmann, San Mateo, 1992) pp. 187–202
5. B.R. Fox, M.B. McMahon, Genetic operators for sequencing problems, in ed. by G.J.E. Rawlins, *Foundations of Genetic Algorithms* (Morgan Kaufmann, San Mateo, 1991)
6. D.E. Goldberg, R. Lingle Jr, Alleles, loci and the traveling salesman problem, in *Proceedings of an International Conference on Genetic Algorithms*, Pittsburgh (Lawrence Erlbaum Associates, Hillsdale, 1985)
7. D.E. Goldberg, *Genetic Algorithms in Search, Optimization & Machine Learning* (Addison-Wesley, Reading, 1989)
8. D.E. Goldberg, Real-coded genetic algorithms, virtual alphabets, and blocking. Technical Report IlliGAL Report No. 90001, Department of General Engineering, University of Illinois at Urbana-Champaign, 1990
9. T. Gong, Principled Design of Nature Inspired Optimizers—Generalizing a Formal Design Methodology, PhD thesis, City University, London, 2008
10. J.J. Grefenstette, GENESIS: a system for using genetic search procedures, in *Proceedings of the 1984 Conference on Intelligent Systems and Machines*, Rochester, 1984, pp. 161–165
11. J.H. Holland, *Adaptation in Natural and Artificial Systems* (University of Michigan Press, Ann Arbor, 1975)
12. T.C. Jones, Evolutionary Algorithms, Fitness Landscapes and Search, PhD thesis, University of New Mexico, 1995
13. Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs* (Springer, Berlin, 1992)
14. A. Moraglio, Towards a Geometric Unification of Evolutionary Algorithms, PhD thesis, University of Essex, 2007
15. I.M. Oliver, D.J. Smith, J.R.C. Holland, A study of permutation crossover operators on the travelling salesman problem, in *Proceedings of the Third International Conference on Genetic Algorithms*, George Mason University, Washington, DC (Morgan Kaufmann, San Mateo, 1987)
16. N.J. Radcliffe, Equivalence class analysis of genetic algorithms. Complex Syst. **5**(2), 183–205 (1991)
17. N.J. Radcliffe, Forma analysis and random respectful recombination, in *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Diego (Morgan Kaufmann, San Mateo, 1991), pp. 222–229
18. N.J. Radcliffe, Genetic set recombination, in ed. by D. Whitley, *Foundations of Genetic Algorithms 2* (Morgan Kaufmann, San Mateo, 1992)
19. N.J. Radcliffe, The algebra of genetic algorithms. Ann. Maths Artif. Intell. **10**, 339–384 (1992)
20. N.J. Radcliffe, P.D. Surry, Fitness variance of formae and performance prediction, in *Foundations of Genetic Algorithms III*,ed. by L.D. Whitley, M.D. Vose (Morgan Kaufmann, San Mateo, 1994) pp. 51–72
21. N.J. Radcliffe, P.D. Surry, Formal memetic algorithms, in ed. by T.C. Fogarty, *Evolutionary Computing: AISB Workshop*, Leeds, Apr 1994, Lecture Notes in Computer Science 865 (Springer, Berlin/New York, 1994) pp. 1–16
22. N.J. Radcliffe, P.D. Surry, Fundamental limitations on search algorithms: evolutionary computing in perspective, in *Computer Science Today: Recent Trends and Developments*, ed. by J. van Leeuwen. Lecture Notes in Computer Science, vol. 1000 (Springer, New York, 1995), pp. 275–291
23. J.E. Rowe, M.D. Vose, A.H. Wright, Group properties of crossover and mutation. Evol. Comput. **10**(2), 151–184 (2002)
24. P.D. Surry, A Prescriptive Formalism for Constructing Domain-specific Evolutionary Algorithms, PhD thesis, University of Edinburgh, 1998

25. P.D. Surry, N.J. Radcliffe, Formal algorithms + formal representations = search strategies, in *Parallel Problem Solving from Nature IV*, Berlin, ed. by H.-M. Voigt, W. Ebeling, I. Rechenberg, H. Schwefel (Springer, LNCS 1141, 1996), pp. 366–375
26. P.D. Surry, N.J. Radcliffe, Real representations, in *Foundations of Genetic Algorithms IV*, ed. by R.K. Belew, M.D. Vose (Morgan Kaufmann, San Mateo, 1996)
27. G. Syswerda, Uniform crossover in genetic algorithms, in *Proceedings of the Third International Conference on Genetic Algorithms*, Fairfax (Morgan Kaufmann, San Mateo, 1989)
28. M.D. Vose, G.E. Liepins, Schema disruption, in *Proceedings of the Fourth International Conference on Genetic Algorithms*, San Diego (Morgan Kaufmann, San Mateo, 1991), pp. 237–243
29. D. Whitley, T. Starkweather, D. Fuquay, Scheduling problems and traveling salesmen: the genetic edge recombination operator, in *Proceedings of the Third International Conference on Genetic Algorithms*, Fairfax (Morgan Kaufmann, San Mateo, 1989)
30. N. Wirth, *Algorithms + Data Structures = Programs* (Prentice-Hall, Englewood Cliffs, 1976)
31. D.H. Wolpert, W.G. Macready, No free lunch theorems for search, Technical Report, SFI–TR–95–02–010, Santa Fe Institute, 1995
32. A.A. Zhigljavsky, *Theory of Global Random Search* (Kluwer Academic, Dordrecht/Boston, 1991)