

INTERNATIONAL SERIES IN OPERATIONS  
RESEARCH AND MANAGEMENT SCIENCE



# Handbook of Metaheuristics

Second Edition

*Edited by*

Michel Gendreau

Jean-Yves Potvin

 Springer

# **International Series in Operations Research & Management Science**

Volume 146

**Series Editor:**

Frederick S. Hillier  
Stanford University, CA, USA

**Special Editorial Consultant:**

Camille C. Price  
Stephen F. Austin State University, TX, USA

For further volumes:  
<http://www.springer.com/series/6161>



Michel Gendreau · Jean-Yves Potvin  
Editors

# Handbook of Metaheuristics

Second Edition



*Editors*

Michel Gendreau  
Département de mathématiques et de  
génie industriel  
École Polytechnique de Montréal, and  
Centre interuniversitaire de recherche  
sur les réseaux d'entreprise  
la logistique et le transport  
C.P. 6079, succ. Centre-ville  
Montréal, QC H3C 3A7, Canada  
michel.gendreau@cirrelt.ca

Jean-Yves Potvin  
Département d'informatique et de  
recherche opérationnelle  
Université de Montréal, and Centre  
interuniversitaire de recherche  
sur les réseaux d'entreprise  
la logistique et le transport  
C.P. 6128, succ. Centre-ville  
Montréal, QC H3C 3J7, Canada  
potvin@iro.umontreal.ca

ISBN 978-1-4419-1663-1      e-ISBN 978-1-4419-1665-5

DOI 10.1007/978-1-4419-1665-5

Springer New York Dordrecht Heidelberg London

Library of Congress Control Number: 2010933095

© Springer Science+Business Media, LLC 2003, 2010

All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher (Springer Science+Business Media, LLC, 233 Spring Street, New York, NY 10013, USA), except for brief excerpts in connection with reviews or scholarly analysis. Use in connection with any form of information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed is forbidden. The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

Printed on acid-free paper

Springer is part of Springer Science+Business Media ([www.springer.com](http://www.springer.com))

*À nos épouses Johanne et Lynne et à nos  
enfants Catherine, Laurent, Gabrielle,  
Stéphanie et Simon.*



# Preface

The first edition of the *Handbook of Metaheuristics* was published in 2003 under the editorship of Fred Glover and Gary A. Kochenberger. Given the numerous developments observed in the field of metaheuristics in recent years, it appeared that the time was ripe for a second edition of the *Handbook*. For different reasons, Fred and Gary were unable to accept Springer's invitation to prepare this second edition and they suggested that we should take over the editorship responsibility of the *Handbook*. We are deeply honored and grateful for their trust.

As stated in the first edition, metaheuristics are “solution methods that orchestrate an interaction between local improvement procedures and higher level strategies to create a process capable of escaping from local optima and performing a robust search of a solution space.” Although this broad characterization still holds today, many new and exciting developments and extensions have been observed in the last few years. We think in particular to hybrids, which take advantage of the strengths of each of their individual metaheuristic components to better explore the solution space. Hybrids of metaheuristics with other optimization techniques, like branch-and-bound, mathematical programming or constraint programming are also increasingly popular. On the front of applications, metaheuristics are now used to find high-quality solutions to an ever-growing number of complex, ill-defined real-world problems, in particular combinatorial ones.

This second edition of the *Handbook of Metaheuristics*, through its 21 chapters, is designed to provide a broad coverage of the concepts, implementations, and applications in this important field of optimization. We were glad to get a positive response from renowned experts for each chapter. They either accepted to revise and update their chapter from the first edition or to write brand new ones. The *Handbook* now includes updated chapters on the best known metaheuristics, including simulated annealing, tabu search, variable neighborhood search, scatter search and path relinking, genetic algorithms, memetic algorithms, genetic programming, ant colony optimization, multi-start methods, greedy randomized adaptive search procedure, guided local search, hyper-heuristics, and parallel metaheuristics. It also contains three new chapters on large neighborhood search, artificial immune systems, and hybrid metaheuristics. The last four chapters are devoted to more general issues

related to the field of metaheuristics, namely reactive search, stochastic search, fitness landscape analysis, and performance comparison. A few chapters from the first edition were discarded, as they appear to be less relevant.

We think that this *Handbook* will be a great reference for researchers and graduate students, as well as practitioners. Each presentation, although exhibiting inevitable stylistic differences, adheres to some common principles which results in stand-alone chapters that can be read individually.

We are grateful to all authors for taking the time to write the chapters that appear in this *Handbook*. We are also very grateful to Fred Hillier, Neil Levine, and Matthew Amboy of Springer for their encouragements, support, and patience at the different stages of production of this book.

Montreal, Canada  
March 2010

Michel Gendreau  
Jean-Yves Potvin

# Preface to First Edition

Metaheuristics, in their original definition, are solution methods that orchestrate an interaction between local improvement procedures and higher level strategies to create a process capable of escaping from local optima and performing a robust search of a solution space. Over time, these methods have also come to include any procedures that employ strategies for overcoming the trap of local optimality in complex solution spaces, especially those procedures that utilize one or more neighborhood structures as a means of defining admissible moves to transition from one solution to another, or to build or destroy solutions in constructive and destructive processes.

The degree to which neighborhoods are exploited varies according to the type of procedure. In the case of certain population-based procedures, such as genetic algorithms, neighborhoods are implicitly (and somewhat restrictively) defined by reference to replacing components of one solution with those of another, by variously chosen rules of exchange popularly given the name of “crossover.” In other population-based methods, based on the notion of path relinking, neighborhood structures are used in their full generality, including constructive and destructive neighborhoods as well as those for transitioning between (complete) solutions. Certain hybrids of classical evolutionary approaches, which link them with local search, also use neighborhood structures more fully, though apart from the combination process itself. Meanwhile, “single thread” solution approaches, which do not undertake to manipulate multiple solutions simultaneously, run a wide gamut that not only manipulate diverse neighborhoods but incorporate numerous forms of strategies ranging from thoroughly randomized to thoroughly deterministic, depending on the elements such as the phase of search or (in the case of memory-based methods) the history of the solution process.<sup>1</sup>

---

<sup>1</sup> Methods based on incorporating collections of memory-based strategies, invoking forms of memory more flexible and varied than those used in approaches such as tree search and branch and bound, are sometimes grouped under the name Adaptive Memory Programming. This term, which originated in the tabu search literature where such adaptive memory strategies were first introduced and continue to be the primary focus, is also sometimes used to encompass other methods that have more recently adopted memory-based elements.

A number of the tools and mechanisms that have emerged from the creation of metaheuristic methods have proved to be remarkably effective, so much so that metaheuristics have moved into the spotlight in recent years as the preferred line of attack for solving many types of complex problems, particularly those of a combinatorial nature. While metaheuristics are not able to certify the optimality of the solutions they find, exact procedures (which theoretically can provide such a certification, if allowed to run long enough)<sup>2</sup> have often proved incapable of finding solutions whose quality is close to that obtained by the leading metaheuristics—particularly for real-world problems, which often attain notably high levels of complexity. In addition, some of the more successful applications of exact methods have come about by incorporating metaheuristic strategies within them. These outcomes have motivated additional research and application of new and improved metaheuristic methodologies.

This handbook is designed to provide the reader with a broad coverage of the concepts, themes, and instrumentalities of this important and evolving area of optimization. In doing so, we hope to encourage an even wider adoption of metaheuristic methods for assisting in problem solving and to stimulate research that may lead to additional innovations in metaheuristic procedures.

The handbook consists of 19 chapters. Topics covered include scatter search, tabu search, genetic algorithms, genetic programming, memetic algorithms, variable neighborhood search, guided local search, GRASP, ant colony optimization, simulated annealing, iterated local search, multi-start methods, constraint programming, constraint satisfaction, neural network methods for optimization, hyper-heuristics, parallel strategies for metaheuristics, metaheuristic class libraries, and A-teams. This family of metaheuristic chapters, while not exhaustive of the many approaches that have sprung into existence in recent years, encompasses the critical strategic elements and their underlying ideas that represent the state of the art of modern metaheuristics.

This book is intended to provide the communities of both researchers and practitioners with a broadly applicable, up-to-date coverage of metaheuristic methodologies that have proven to be successful in a wide variety of problem settings and that hold particular promise for success in the future. The various chapters serve as stand-alone presentations giving both the necessary underpinnings as well as practical guides for implementation. The nature of metaheuristics invites an analyst to modify basic methods in response to problem characteristics, past experiences, and personal preferences, and the chapters in this handbook are designed to facilitate this process as well.

---

<sup>2</sup> Some types of problems seem quite amenable to exact methods, particularly to some of the methods embodied in the leading commercial software packages for mixed integer programming. Yet even by these approaches the “length of time” required to solve many problems exactly appears to exceed all reasonable measure, including in some cases measures of astronomical scale. It has been conjectured that metaheuristics succeed where exact methods fail because of their ability to use strategies of greater flexibility than permitted to assure that convergence will inevitably be obtained.

The authors who have contributed to this volume represent leading figures from the metaheuristic community and are responsible for pioneering contributions to the fields they write about. Their collective work has significantly enriched the field of optimization in general and combinatorial optimization in particular. We are especially grateful to them for agreeing to provide the first-rate chapters that appear in this handbook. We would also like to thank our graduate students, Gyung Yung and Rahul Patil, for their assistance. Finally, we would like to thank Gary Folven and Carolyn Ford of Kluwer Academic Publishers for their unwavering support and patience throughout this project.

Fred Glover  
Gary A. Kochenberger



# Contents

<b>Preface .....</b>	vii
<b>Preface to First Edition .....</b>	ix
<b>Contributors .....</b>	xv
<b>1 Simulated Annealing .....</b>	1
Alexander G. Nikolaev and Sheldon H. Jacobson	
<b>2 Tabu Search .....</b>	41
Michel Gendreau and Jean-Yves Potvin	
<b>3 Variable Neighborhood Search .....</b>	61
Pierre Hansen, Nenad Mladenović, Jack Brimberg and José A. Moreno Pérez	
<b>4 Scatter Search and Path-Re-linking: Fundamentals, Advances, and Applications .....</b>	87
Mauricio G.C. Resende, Celso C. Ribeiro, Fred Glover and Rafael Martí	
<b>5 Genetic Algorithms .....</b>	109
Colin R. Reeves	
<b>6 A Modern Introduction to Memetic Algorithms .....</b>	141
Pablo Moscato and Carlos Cotta	
<b>7 Genetic Programming .....</b>	185
William B. Langdon, Robert I. McKay and Lee Spector	
<b>8 Ant Colony Optimization: Overview and Recent Advances .....</b>	227
Marco Dorigo and Thomas Stützle	

<b>9</b>	<b>Advanced Multi-start Methods . . . . .</b>	265
	R. Martí, J. Marcos Moreno-Vega, and A. Duarte	
<b>10</b>	<b>Greedy Randomized Adaptive Search Procedures: Advances, Hybridizations, and Applications . . . . .</b>	283
	Mauricio G.C. Resende and Celso C. Ribeiro	
<b>11</b>	<b>Guided Local Search . . . . .</b>	321
	Christos Voudouris, Edward P.K. Tsang and Abdullah Alshedy	
<b>12</b>	<b>Iterated Local Search: Framework and Applications . . . . .</b>	363
	Helena R. Lourenço, Olivier C. Martin and Thomas Stützle	
<b>13</b>	<b>Large Neighborhood Search . . . . .</b>	399
	David Pisinger and Stefan Ropke	
<b>14</b>	<b>Artificial Immune Systems . . . . .</b>	421
	Julie Greensmith, Amanda Whitbrook and Uwe Aickelin	
<b>15</b>	<b>A Classification of Hyper-heuristic Approaches . . . . .</b>	449
	Edmund K. Burke, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and John R. Woodward	
<b>16</b>	<b>Metaheuristic Hybrids . . . . .</b>	469
	Günther R. Raidl, Jakob Puchinger and Christian Blum	
<b>17</b>	<b>Parallel Meta-heuristics . . . . .</b>	497
	Teodor Gabriel Crainic and Michel Toulouse	
<b>18</b>	<b>Reactive Search Optimization: Learning While Optimizing . . . . .</b>	543
	Roberto Battiti and Mauro Brunato	
<b>19</b>	<b>Stochastic Search in Metaheuristics . . . . .</b>	573
	Walter J. Gutjahr	
<b>20</b>	<b>An Introduction to Fitness Landscape Analysis and Cost Models for Local Search . . . . .</b>	599
	Jean-Paul Watson	
<b>21</b>	<b>Comparison of Metaheuristics . . . . .</b>	625
	John Silberholz and Bruce Golden	
	<b>Subject Index . . . . .</b>	641

# **Contributors**

**Uwe Aickelin**

The University of Nottingham, Nottingham, UK

e-mail: [uxa@cs.nott.ac.uk](mailto:uxa@cs.nott.ac.uk)

**Abdullah Alsheddy**

University of Essex, Colchester, UK

e-mail: [aalshe@essex.ac.uk](mailto:aalshe@essex.ac.uk)

**Roberto Battiti**

Università di Trento, Trento, Italy

e-mail: [roberto.battiti@unitn.it](mailto:roberto.battiti@unitn.it)

**Christian Blum**

Universitat Politècnica de Catalunya, Barcelona, Spain

e-mail: [cblum@lsi.upc.edu](mailto:cblum@lsi.upc.edu)

**Jack Brimberg**

Royal Military College of Canada, Kingston, ON, Canada

e-mail: [jack.brimberg@rmc.ca](mailto:jack.brimberg@rmc.ca)

**Mauro Brunato**

Università di Trento, Trento, Italy

e-mail: [mauro.brunato@unitn.it](mailto:mauro.brunato@unitn.it)

**Edmund K. Burke**

The University of Nottingham, Nottingham, UK

e-mail: [ekb@cs.nott.ac.uk](mailto:ekb@cs.nott.ac.uk)

**Carlos Cotta**

Universidad de Málaga, Málaga, Spain

e-mail: [ccottap@lcc.uma.es](mailto:ccottap@lcc.uma.es)

**Teodor Gabriel Crainic**

Université du Québec à Montréal and CIRRELT, Montréal, QC, Canada

e-mail: teodorgabriel.crainic@cirrelt.ca

**Marco Dorigo**

Université Libre de Bruxelles, Brussels, Belgium

e-mail: mdorigo@ulb.ac.be

**Abraham Duarte**

Universidad Rey Juan Carlos, Madrid, Spain

e-mail: abraham.duarte@urjc.es

**Michel Gendreau**

École Polytechnique de Montréal and CIRRELT, Montréal, QC, Canada

e-mail: michel.gendreau@cirrelt.ca

**Fred Glover**

University of Colorado and OptTek Systems, Inc., Boulder, CO, USA

e-mail: glover@colorado.edu

**Bruce Golden**

University of Maryland, College Park, MD, USA

e-mail: bgolden@rhsmith.umd.edu

**Julie Greensmith**

The University of Nottingham, Nottingham, UK

e-mail: jqg@cs.nott.ac.uk

**Walter J. Gutjahr**

University of Vienna, Vienna, Austria

e-mail: walter.gutjahr@univie.ac.at

**Pierre Hansen**

École des Hautes Études Commerciales and GERAD, Montréal, QC, Canada

e-mail: pierreh@crt.umontreal.ca

**Matthew Hyde**

The University of Nottingham, Nottingham, UK

e-mail: mvh@cs.nott.ac.uk

**Sheldon H. Jacobson**

University of Illinois, Urbana, IL, USA

e-mail: shj@illinois.edu

**Graham Kendall**

The University of Nottingham, Nottingham, UK

e-mail: gxk@cs.nott.ac.uk

**William B. Langdon**

University College London, London, UK

e-mail: w.langdon@cs.ucl.ac.uk

**Helena R. Lourenço**  
Universitat Pompeu Fabra, Barcelona, Spain  
e-mail: helena.ramalhinho@upf.edu

**J. Marcos Moreno-Vega**  
Universidad de La Laguna, La Laguna, Santa Cruz de Tenerife, Spain  
e-mail: jmmoreno@ull.es

**Rafael Martí**  
Universidad de Valencia, Valencia, Spain  
e-mail: rafael.marti@uv.es

**Olivier C. Martin**  
Université Paris-Sud, Orsay, France  
e-mail: olivier.martin@u-psud.fr

**Robert I. McKay**  
Seoul National University, Seoul, Korea  
e-mail: rim@cse.snu.ac.kr

**Nenad Mladenović**  
Brunel University-West London, Uxbridge, UK  
e-mail: nenad.mladenovic@brunel.ac.uk

**José A. Moreno Pérez**  
Universidad de La Laguna, La Laguna, Santa Cruz de Tenerife, Spain  
e-mail: jamoreno@ull.es

**Pablo Moscato**  
The University of Newcastle, Callaghan, NSW Australia  
e-mail: pablo.moscato@newcastle.edu.au

**Alexander G. Nikolaev**  
University at Buffalo, Buffalo, NY, USA  
e-mail: anikolae@buffalo.edu

**Gabriela Ochoa**  
The University of Nottingham, Nottingham, UK  
e-mail: gxo@cs.nott.ac.uk

**Ender Özcan**  
The University of Nottingham, Nottingham, UK  
e-mail: exo@cs.nott.ac.uk

**David Pisinger**  
Technical University of Denmark, Lyngby, Denmark  
e-mail: pisinger@man.dtu.dk

**Jean-Yves Potvin**  
Université de Montréal and CIRRELT, Montréal, QC, Canada  
e-mail: potvin@iro.umontreal.ca

Jakob Puchinger  
University of Melbourne, Melbourne, Australia  
e-mail: jakobp@csse.unimelb.edu.au

Günther R. Raidl  
Vienna University of Technology, Vienna, Austria  
e-mail: raidl@ads.tuwien.ac.at

Colin R. Reeves  
Coventry University, Coventry, UK  
e-mail: c.reeves@coventry.ac.uk

Mauricio G.C. Resende  
AT&T Labs Research, Florham Park, NJ, USA  
e-mail: mgcr@research.att.com

Celso C. Ribeiro  
Universidade Federal Fluminense, Niterói, RJ, Brazil  
e-mail: celso@ic.uff.br

Stefan Ropke  
Technical University of Denmark, Lyngby, Denmark  
e-mail: sr@transport.dtu.dk

John Silberholz  
University of Maryland, College Park, MD, USA  
e-mail: josilber@umd.edu

Lee Spector  
Hampshire College, Amherst, MA, USA  
e-mail: lspector@hampshire.edu

Thomas Stützle  
Université Libre de Bruxelles, Brussels, Belgium  
e-mail: stuetzle@ulb.ac.be

Michel Toulouse  
CIRRELT, Montréal, QC, Canada  
e-mail: michel.toulouse@cirrelt.ca

Edward P.K. Tsang  
University of Essex, Colchester, UK  
e-mail: edward@essex.ac.uk

Christos Voudouris  
BT Group plc, Ipswich, UK  
e-mail: chris.voudouris@bt.com

Jean-Paul Watson  
Sandia National Laboratories, Albuquerque, NM, USA  
e-mail: jwatson@sandia.gov

Amanda Whitbrook  
The University of Nottingham, Nottingham, UK  
e-mail: amw@cs.nott.ac.uk

John R. Woodward  
The University of Nottingham, Nottingham, UK  
e-mail: jrw@cs.nott.ac.uk



# Chapter 1

## Simulated Annealing

Alexander G. Nikolaev and Sheldon H. Jacobson

**Abstract** Simulated annealing is a well-studied local search metaheuristic used to address discrete and, to a lesser extent, continuous optimization problems. The key feature of simulated annealing is that it provides a mechanism to escape local optima by allowing hill-climbing moves (i.e., moves which worsen the objective function value) in hopes of finding a global optimum. A brief history of simulated annealing is presented, including a review of its application to discrete, continuous, and multi-objective optimization problems. Asymptotic convergence and finite-time performance theory for simulated annealing are reviewed. Other local search algorithms are discussed in terms of their relationship to simulated annealing. The chapter also presents practical guidelines for the implementation of simulated annealing in terms of cooling schedules, neighborhood functions, and appropriate applications.

### 1.1 Background Survey

Simulated annealing is a local search algorithm (metaheuristic) capable of escaping from local optima. Its ease of implementation and convergence properties and its use of hill-climbing moves to escape local optima have made it a popular technique over the past two decades. It is typically used to address discrete and, to a lesser extent, continuous optimization problems. Survey articles that provide a good overview of simulated annealing's theoretical development and domains of application include [46, 55, 75, 90, 120, 144]. Aarts and Korst [1] and van Laarhoven

---

Alexander G. Nikolaev  
Industrial and Systems Engineering, University at Buffalo, Buffalo, NY 14260-2050  
e-mail: anikolae@buffalo.edu

Sheldon H. Jacobson  
Department of Computer Science, University of Illinois, Urbana, IL, USA 61801-2302  
e-mail: shj@illinois.edu

and Aarts [155] devote entire books to the subject. Aarts and Lenstra [2] dedicate a chapter to simulated annealing in their book on local search algorithms for discrete optimization problems.

### 1.1.1 History and Motivation

Simulated annealing is so named because of its analogy to the process of physical annealing with solids, in which a crystalline solid is heated and then allowed to cool very slowly until it achieves its most regular possible crystal lattice configuration (i.e., its minimum lattice energy state), and thus is free of crystal defects. If the cooling schedule is sufficiently slow, the final configuration results in a solid with such superior structural integrity. Simulated annealing establishes the connection between this type of thermodynamic behavior and the search for global minima for a discrete optimization problem. Furthermore, it provides an algorithmic means for exploiting such a connection.

At each iteration of a simulated annealing algorithm applied to a discrete optimization problem, the values for two solutions (the current solution and a newly selected solution) are compared. Improving solutions are always accepted, while a fraction of non-improving (inferior) solutions are accepted in the hope of escaping local optima in search of global optima. The probability of accepting non-improving solutions depends on a temperature parameter, which is typically non-increasing with each iteration of the algorithm.

The key algorithmic feature of simulated annealing is that it provides a means to escape local optima by allowing *hill-climbing* moves (i.e., moves which worsen the objective function value). As the temperature parameter is decreased to zero, hill-climbing moves occur less frequently, and the solution distribution associated with the inhomogeneous Markov chain that models the behavior of the algorithm converges to a form in which all the probability is concentrated on the set of globally optimal solutions (provided that the algorithm is convergent; otherwise the algorithm will converge to a local optimum, which may or may not be globally optimal).

### 1.1.2 Definition of Terms

To describe the specific features of a simulated annealing algorithm for discrete optimization problems, several definitions are needed. Let  $\Omega$  be the *solution space* (i.e., the set of all possible solutions). Let  $f: \Omega \rightarrow \mathfrak{R}$  be an *objective function* defined on the solution space. The goal is to find a global minimum,  $\omega^*$  (i.e.,  $\omega^* \in \Omega$  such that  $f(\omega^*) \leq f(\omega)$  for all  $\omega \in \Omega$ ). The objective function must be bounded to ensure that  $\omega^*$  exists. Define  $N(\omega)$  to be the *neighborhood function* for  $\omega \in \Omega$ . Therefore, associated with every solution,  $\omega \in \Omega$ , are neighboring solutions,  $N(\omega)$ , that can be reached in a single iteration of a local search algorithm.

Simulated annealing starts with an initial solution  $\omega \in \Omega$ . A neighboring solution  $\omega' \in N(\omega)$  is then generated (either randomly or using some pre-specified rule). Simulated annealing is based on the Metropolis acceptance criterion [101], which models how a thermodynamic system moves from the current solution (state)  $\omega \in \Omega$  to a candidate solution  $\omega' \in N(\omega)$ , in which the energy content is being minimized. The candidate solution,  $\omega'$ , is accepted as the current solution based on the acceptance probability

$$P\{\text{Accept } \omega' \text{ as next solution}\} = \begin{cases} \exp[-(f(\omega') - f(\omega))/t_k] & \text{if } f(\omega') - f(\omega) > 0 \\ 1 & \text{if } f(\omega') - f(\omega) \leq 0. \end{cases} \quad (1.1)$$

Define  $t_k$  as the temperature parameter at (outer loop) iteration  $k$ , such that

$$t_k > 0 \text{ for all } k \text{ and } \lim_{k \rightarrow \infty} t_k = 0. \quad (1.2)$$

This acceptance probability is the basic element of the search mechanism in simulated annealing. If the temperature is reduced sufficiently slowly, then the system can reach an equilibrium (steady state) at each iteration  $k$ . Let  $f(\omega)$  and  $f(\omega')$  denote the energies (objective function values) associated with solutions  $\omega \in \Omega$  and  $\omega' \in N(\omega)$ , respectively. This equilibrium follows the Boltzmann distribution, which can be described as the probability of the system being in state  $\omega \in \Omega$  with energy  $f(\omega)$  at temperature  $T$  such that

$$P\{\text{System is in state } \omega \text{ at temperature } T\} = \frac{\exp(-f(\omega)/t_k)}{\sum_{\omega'' \in \Omega} \exp(-f(\omega'')/t_k)}. \quad (1.3)$$

If the probability of generating a candidate solution  $\omega'$  from the neighbors of solution  $\omega \in \Omega$  is  $g_k(\omega, \omega')$ , where

$$\sum_{\omega' \in N(\omega)} g_k(\omega, \omega') = 1, \text{ for all } \omega \in \Omega, k = 1, 2, \dots, \quad (1.4)$$

then a non-negative square stochastic matrix  $\mathbf{P}_k$  can be defined with transition probabilities

$$P_k(\omega, \omega') = \begin{cases} g_k(\omega, \omega') \exp(-\Delta_{\omega, \omega'}/t_k) & \omega' \in N(\omega), \quad \omega' \neq \omega \\ 0 & \omega' \notin N(\omega), \quad \omega' \neq \omega \\ 1 - \sum_{\omega'' \in N(\omega), \omega'' \neq \omega} P_k(\omega, \omega'') & \omega' = \omega \end{cases} \quad (1.5)$$

for all solutions  $\omega \in \Omega$  and all iterations  $k = 1, 2, \dots$  and with  $\Delta_{\omega, \omega'} \equiv f(\omega') - f(\omega)$ . These transition probabilities define a sequence of solutions generated from an inhomogeneous Markov chain [120]. Note that boldface type indicates matrix/vector notation, and all vectors are row vectors.

### 1.1.3 Statement of Algorithm

Simulated annealing is outlined in pseudo-code (see [46]).

```

Select an initial solution  $\omega \in \Omega$ 
Select the temperature change counter  $k = 0$ 
Select a temperature cooling schedule,  $t_k$ 
Select an initial temperature  $T = t_0 \geq 0$ 
Select a repetition schedule,  $M_k$ , that defines the number of iterations executed at
each temperature,  $t_k$ 
Repeat
    Set repetition counter  $m = 0$ 
    Repeat
        Generate a solution  $\omega' \in N(\omega)$ 
        Calculate  $\Delta_{\omega,\omega'} = f(\omega') - f(\omega)$ 
        If  $\Delta_{\omega,\omega'} \leq 0$ , then  $\omega \leftarrow \omega'$ 
        If  $\Delta_{\omega,\omega'} > 0$ , then  $\omega \leftarrow \omega'$  with probability  $\exp(-\Delta_{\omega,\omega'}/t_k)$ 
         $m \leftarrow m + 1$ 
    Until  $m = M_k$ 
     $k \leftarrow k + 1$ 
Until stopping criterion is met

```

This simulated annealing formulation results in  $M_0 + M_1 + \dots + M_k$  total iterations being executed, where  $k$  corresponds to the value for  $t_k$  at which some stopping criterion is met (for example, a pre-specified total number of iterations has been executed or a solution of a certain quality has been found). In addition, if  $M_k = 1$  for all  $k$ , then the temperature changes at each iteration.

### 1.1.4 Discrete Versus Continuous Problems

The majority of the theoretical developments and application work with simulated annealing has been for discrete optimization problems. However, simulated annealing has also been used as a tool to address problems in the continuous domain. There is considerable interest in using simulated annealing for global optimization over regions containing several local and global minima (due to an inherent non-linearity of objective functions). Fabian [48] studies the performance of simulated annealing methods for finding a global minimum of a given objective function. Bohachevsky et al. [15] propose a generalized simulated annealing algorithm for function optimization for use in statistical applications, and Locatelli [96] presents a proof of convergence for the algorithm. Optimization of continuous functions involves finding a candidate solution by picking a direction from the current (incumbent) solution and a step size to take in this direction and evaluating the function at the new (candidate) location. If the function value of this candidate location is an improvement over the function value of the incumbent location, then the candidate

becomes the incumbent. This migration through local minima in search of a global minimum continues until the global minimum is found or some termination criteria are reached. Belisle [12] presents a special simulated annealing algorithm for global optimization, which uses a heuristically motivated cooling schedule. This algorithm is easy to implement and provides a reasonable alternative to existing methods.

Belisle et al. [13] discuss convergence properties of simulated annealing algorithms applied to continuous functions and apply these results to hit-and-run algorithms used in global optimization. The presented convergence properties are consistent with those presented in [72] and provide a good contrast between convergence in probability and (the stronger) almost sure convergence. This work is further extended in [166] to an improved hit-and-run algorithm used for global optimization.

Fleischer and Jacobson [57] propose cybernetic optimization by simulated annealing as a method of parallel processing that accelerates the convergence of simulated annealing to the global optima. This theory is extended by Fleischer [56] into the continuous domain by applying probabilistic feedback control to the generation of candidate solutions. The probabilistic feedback control method of generating candidate solutions effectively accelerates convergence to a global optimum using parallel simulated annealing on continuous variable problems.

Locatelli [94] presents convergence properties for a class of simulated annealing algorithms for continuous global optimization by removing the restriction that the next candidate point must be generated according to a probability distribution whose support is the whole feasible set. A study on simulated annealing algorithms for globally minimizing functions of multiple continuous variables is conducted by Siarry et al. [131]. The study focuses on how high dimensionality can be addressed using variables discretization and addresses the design and implementation issues for several complementary stopping criteria. Convergence results and criteria for simulated annealing applied to continuous global optimization problems are also provided in [163] and [95]. A general-purpose simulated annealing algorithm that solves mixed integer linear programs is introduced by Kiatsupaibul and Smith [88]. The simulated annealing algorithm is constructed using a Markov chain sampling algorithm to generate uniformly distributed points on an arbitrary bounded region of a high-dimensional integer lattice. They show that the algorithm converges in probability to a global optimum. Romeijn et al. [119] study a simulated annealing algorithm that uses a reflection generator for mixed integer/continuous global optimization problems. Locatelli [96] establishes the convergence of simulated annealing algorithms for continuous global optimization and an upper bound for the expected first hitting time, i.e., the expected number of iterations before reaching the global optimum value within accuracy  $\varepsilon$ .

### **1.1.5 Single-objective Versus Multi-objective Problems**

Originally used as an optimization tool for combinatorial optimization problems, simulated annealing has recently been adapted to address multi-objective problems

(see [144]). Its framework is easy to implement and simulated annealing-based algorithms are capable of producing a Pareto set of solutions in a single run with very little computational cost. Additionally, its performance is not influenced by the shape of the Pareto set, which is a concern for mathematical programming techniques.

The first multi-objective version of simulated annealing has been proposed by Serafini [128, 129]. The method closely follows the guidelines of regular single-objective simulated annealing and uses a modification of the solution acceptance criteria in the original algorithm. Various alternative criteria have been investigated, with the objective to increase the probability of accepting non-dominated solutions. A special selection rule produced by the combination of several criteria has been proposed in order to concentrate the search almost exclusively on the non-dominated solutions. This idea has also been used by Ulungu and Teghem [152] and Serafini [130], with the latter utilizing a target-vector approach to solve a bi-objective optimization problem. Ulungu et al. [154] propose a complete MOSA algorithm, where a weighted aggregating function is used to evaluate and compare the obtained solutions. The MOSA algorithm works with only one current solution but keeps a record of the population of non-dominated solutions found during the search. A further improved, interactive version of MOSA is presented in [153] and is referred to as the UMOSA method. Suppapatnern and Parks [145] propose a different simulated annealing-based approach proposed to tackle multi-objective problems (SMOSA method). At each iteration, the algorithm does the search based on one solution and the annealing process adjusts the temperature adaptively, using the objective function value of the obtained solution in each of the multiple objectives. An archive is created to store all the non-dominated solutions.

The idea of introducing the “new-acceptance” probability formulation based on an annealing schedule with multiple temperatures (one for each objective) has also been proposed. The acceptance probability of a “new solution” depends on whether or not it is added to the set of potentially Pareto-optimal solutions. If it is added to the Pareto set, then it is accepted as the current solution with probability equal to 1. Otherwise, a multi-objective acceptance rule is used. Czyzak et al. [36] and Czyzak and Jaszkiewicz [37] propose another way to adapt simulated annealing to a multi-objective context, which combines the ideas of unicriterion simulated annealing and genetic algorithms to provide efficient solutions for a multi-criteria shortest path problem. A classical neighborhood search has been used to explore the population of solutions, with the weight for each objective function adjusted in each iteration. This technique increases the probability of escaping local optima, in a way similar to multi-objective tabu search.

Suman [141–143] proposes different simulated annealing-based approaches to tackle constrained multi-objective optimization problems. In [142], a comparative analysis of five simulated annealing algorithms is conducted for the system reliability optimization problem. The goal of these methods is to generate a set of solutions that are a good approximation to the entire set of efficient (non-dominated or Pareto-optimal) solutions over a fixed time period.

Villalobos-Arias et al. [159, 160] study asymptotic convergence of simulated annealing algorithms for multi-objective optimization problems in comparison with other algorithms such as an Artificial Immune System and a General Evolutionary Algorithm. Tekinalp and Karsli [146] present a simulated annealing algorithm for continuous multi-objective optimization that has an adaptive cooling schedule and uses a population of fitness functions to accurately generate the Pareto front. Whenever an improvement with a fitness function is encountered, the trial point is accepted and the temperature parameters associated with the improving fitness functions are cooled down. In addition to well-known linear fitness functions, special elliptic and ellipsoidal fitness functions, suitable for the generation on non-convex fronts, are used.

## 1.2 Convergence Results

### 1.2.1 Asymptotic Performance

Asymptotic convergence results for simulated annealing have typically taken one of two directions: the algorithm has been modeled either as a sequence of homogeneous Markov chains or as a single inhomogeneous Markov chain.

#### 1.2.1.1 Homogeneous Markov Chain Approach

The homogeneous Markov chain approach [3, 49, 70, 84, 85, 97, 104, 123] assumes that each temperature  $t_k$  is held constant for a sufficient number of iterations  $m$  such that the stochastic matrix  $\mathbf{P}_k$  can reach its stationary (steady-state) distribution  $\pi_k$ . Note that in the interest of simplifying notation, the inner loop index  $m$  is suppressed. However, the index  $k$  should be interpreted as the double index  $k, m$ , where a sequence of  $m = 1, 2, \dots, M_k$  simulated annealing iterations occur for each fixed  $k$ . The existence of a stationary distribution at each iteration  $k$  follows from Theorem 1. (Note: To ensure that Theorem 1 is consistent with the simulated annealing algorithm depicted in Section 1.1.3, without loss of generality, let  $t_k$  be a function only of each outer loop iteration  $k$ , and let the respective number of inner loop iterations  $M_k$  and outer loop iterations  $k$  each approach infinity).

**Theorem 1** *Let  $\mathbf{P}_k(\omega, \omega')$  be the probability of moving from solution  $\omega$  to solution  $\omega'$  in one inner iteration at outer loop  $k$ , and let  $\mathbf{P}_k^{(m)}(\omega, \omega')$  be the probability of going from solution  $\omega$  to solution  $\omega'$  in  $m$  inner loops. If the Markov chain associated with  $\mathbf{P}_k^{(m)}(\omega, \omega')$  is irreducible and aperiodic with finitely many solutions, then  $\lim_{m \rightarrow \infty} \mathbf{P}_k^{(m)}(\omega, \omega') = \pi_k(\omega')$  exists for all  $\omega, \omega' \in \Omega$  and iterations  $k$ . Furthermore,  $\pi_k(\omega')$  is the unique strictly positive solution of*

$$\pi_k(\omega') = \sum_{\omega \in \Omega} \pi_k(\omega) \mathbf{P}_k(\omega, \omega'), \text{ for all } \omega' \in \Omega, \quad (1.6)$$

and

$$\sum_{\omega \in \Omega} \pi_k(\omega) = 1. \quad (1.7)$$

*Proof* See [33].

The key requirements for the existence of the stationary distributions and for the convergence of the sequence of  $\pi_k$  vectors include the following:

1. transition matrix irreducibility (for every finite outer loop  $k$ , the transition matrix can assign a path of non-zero probability between any two solutions  $\omega, \omega' \in \Omega$ ),
2. aperiodicity (starting at solution  $\omega' \in \Omega$ ), it is possible to return to  $\omega, \omega'$  with period 1; see [78],
3. A non-zero stationary probability distribution, as the number of outer loops  $k$  approaches infinity.

Note that all simulated annealing proofs of convergence in the literature based on homogeneous Markov chain theory, either explicitly or implicitly, use the sufficient condition of reversibility (also called detailed balance; see [122]) defined as

$$\pi_k(\omega) \mathbf{P}_k(\omega, \omega') = \pi_k(\omega') \mathbf{P}_k(\omega', \omega), \text{ for all } \omega, \omega' \in \Omega, \text{ and all iterations } k. \quad (1.8)$$

Reversibility is a sufficient condition for a unique solution to exist for Equations (1.6) and (1.7) at each outer loop iteration  $k$ . A necessary condition for reversibility is multiplicativity. That is, for any three solutions  $\omega, \omega', \omega'' \in \Omega$  such that  $f(\omega) \leq f(\omega') \leq f(\omega'')$  and for all iterations  $k$ ,

$$\kappa_k(\Delta_{\omega, \omega''}) = \kappa_k(\Delta_{\omega, \omega'}) \kappa_k(\Delta_{\omega', \omega''}), \quad (1.9)$$

where  $\kappa_k(\Delta_{\omega, \omega'})$  is the probability of accepting the transition from solution  $\omega$  to solution  $\omega'$  at outer loop iteration  $k$ . Reversibility is enforced by assuming conditions of symmetry on the solution generation probabilities  $g_k$  and either by directly expressing the acceptance probability using an exponential form or by requiring the multiplicative condition in Equation (1.9).

The homogeneous Markov chain proofs of convergence in the literature (implicitly or explicitly) require the condition in Equation (1.9) to hold for the acceptance function and then address the sufficient conditions on the solution generation matrix  $\mathbf{P}_k$ . For example, the original homogeneous proofs of convergence [3, 97] require the multiplicative condition for the acceptance function, and then assume that the solution generation matrix is symmetric and constant for all outer loop iterations  $k$ . Rossier et al. [123] partition the solution space into blocks composed of neighboring solutions of equal objective function value and then require that only the solution generation probabilities be symmetric between these blocks. Rossier et al. then express the acceptance function as a ratio of the stationary distribution probabilities. Faigle and Schrader [51] and Faigle and Kern [49] use a graph theoretic approach to relax the solution generation matrix symmetry condition.

However, they require that the solution acceptance probability function satisfies Equation (1.9).

Granville et al. [70] propose a simulated annealing procedure for filtering binary images, where the acceptance function is based on the probability of the current solution, instead of the change in objective function value. The probability function that Granville et al. [70] present for accepting a candidate solution at (outer loop) iteration  $k$  is based on the ratio of the stationary probability of the incumbent solution from iteration  $k - 1$  versus the stationary probability of an initial solution (which is based on a maximum likelihood estimate). The acceptance probability is

$$\xi_k = q\pi_0(\omega)/\pi_{k-1}^{\phi(k)}(\omega') \quad (1.10)$$

where  $q = \inf_{\omega \in \Omega} \pi(\omega)/\sup_{\omega' \in \Omega} \pi(\omega')$  ( $q$  must also be estimated), and  $\phi(k)$  is a slowly increasing function. Therefore, the probability of a solution transition does not consider the objective function value of the candidate solution. Granville et al. [70] provide a proof of asymptotic convergence of this approach, but note that the proof methodology does not show that the set of globally optimal solutions are asymptotically uniformly distributed.

Simulated annealing and the homogeneous convergence theory are based on the work of Metropolis et al. [101], which addresses problems in equilibrium statistical mechanics [74]. To see this relationship, consider a system in thermal equilibrium with its surroundings, in solution (state)  $S$  with energy  $F(S)$ . The probability density in phase space of the point representing  $S$  is proportional to

$$\exp(-F(S)/bT), \quad (1.11)$$

where  $b$  is the Boltzmann constant, and  $T$  is the absolute temperature of the surroundings. Therefore the proportion of time that the system spends in solution  $S$  is proportional to Equation (1.11) (see [74]), hence the equilibrium probability density for all  $S \in \Omega$  is

$$\pi_s = \frac{\exp(-F(S)/bT)}{\int \exp(-F(S)/bT) dS}. \quad (1.12)$$

The expectation of any valid solution function  $f(S)$  is thus

$$E[f] = \frac{\int f(S) \exp(-F(S)/bT) dS}{\int \exp(-F(S)/bT) dS}. \quad (1.13)$$

Unfortunately, for many solution functions, Equation (1.13) cannot be evaluated analytically. Hammersley and Handscomb [74] note that one could theoretically use naive Monte Carlo techniques to estimate the value of the two integrals in Equation (1.13). However, this often fails in practice since the exponential factor means that a significant portion of the integrals is concentrated in a very small region of the solution space  $\Omega$ . This problem can be overcome using importance sampling (see [18], Chapter 2), by generating solutions with probability density Equation (1.12). This approach would also seem to fail, because of the integral in

the denominator of Equation (1.12). However, Metropolis et al. [101] solve this problem by first discretizing the solution space, such that the integrals in Equations (1.12) and (1.13) are replaced by summations over the set of discrete solutions  $\omega' \in \Omega$ , and then by constructing an irreducible, aperiodic Markov chain with transition probabilities  $\mathbf{P}(\omega, \omega')$  such that

$$\pi(\omega') = \sum_{\omega \in \Omega} \pi(\omega) \mathbf{P}(\omega, \omega') \quad \text{for all } \omega' \in \Omega, \quad (1.14)$$

where

$$\pi(\omega') = \frac{\exp(-F(\omega')/bT)}{\sum_{\omega \in \Omega} \exp(-F(\omega)/bT)dS} \quad \text{for all } \omega' \in \Omega. \quad (1.15)$$

Note that to compute the equilibrium distribution  $\pi$ , the denominator of Equation (1.13) (a normalizing constant) does not need to be calculated. Instead, the ratios  $\pi(\omega')/\pi(\omega)$  need only be computed and a transition matrix  $\mathbf{P}$  defined that satisfies Equation (1.14). Hammersley and Handscomb [74] show that Metropolis et al. [101] accomplish this by defining  $\mathbf{P}$  as the product of symmetric solution generation probabilities  $g(\omega, \omega')$ , and the equilibrium ratios  $\pi(\omega')/\pi(\omega)$ ,

$$\mathbf{P}(\omega, \omega') = \begin{cases} g(\omega, \omega') \pi(\omega')/\pi(\omega) & \text{if } \pi(\omega')/\pi(\omega) < 1, \quad \omega' \neq \omega \\ g(\omega, \omega') & \text{if } \pi(\omega')/\pi(\omega) \geq 1, \quad \omega' \neq \omega \\ g(\omega, \omega') + \Delta & \text{if } \omega' = \omega \end{cases} \quad (1.16)$$

with  $\Delta = \sum_{\omega'' \in \Omega, \pi(\omega'') < \pi(\omega)} g(\omega, \omega'') (1 - (\pi(\omega'')/\pi(\omega)))$ , where

$$g(\omega, \omega') \geq 0, \quad \sum_{\omega' \in \Omega} g(\omega, \omega') = 1, \quad \text{and } g(\omega, \omega') = g(\omega', \omega) \quad \text{for all } \omega, \omega' \in \Omega. \quad (1.17)$$

The use of stationary probability ratios to define the solution acceptance probabilities, combined with symmetric solution generation probabilities, enables Metropolis et al. [101] to use the reversibility condition in Equation (1.8) to show that Equations (1.16) and (1.17) satisfy Equation (1.14).

Homogeneous proofs of convergence for simulated annealing become more difficult to establish when the reversibility condition is not satisfied. Note that the existence of a unique stationary distribution for each outer loop iteration  $k$  is easily shown by specifying that each transition matrix  $\mathbf{P}_k$  be irreducible and aperiodic. On the other hand, it becomes very difficult to derive an explicit closed-form expression for each stationary distribution  $\pi_k$  that remains analytically tractable as the problem's solution space becomes large. One can no longer use Equation (1.8) to describe each stationary distribution, since, in general, the multiplicative condition is not met. Instead, one must directly solve the system of equations formed with Equations (1.6) and (1.7). For example, in [38], Davis attempts to obtain a closed-form expression for  $\pi_k$  by using Cramer's rule and rewriting Equation (1.6) and (1.7) as

$$\pi_k(\mathbf{I} - \mathbf{P}_k) = 0 \quad (1.18)$$

and

$$\pi_k \mathbf{e}^T = 1, \quad (1.19)$$

respectively, where boldface type indicates vector/matrix notation,  $\mathbf{I}$  is the identity matrix, and  $\mathbf{e}^T$  is a column vector of ones. Note that the  $\text{card}(\Omega) \times \text{card}(\Omega)$  transition matrix  $\mathbf{P}_k$  associated with Equation (1.18) is of rank  $\text{card}(\Omega) - 1$  [33]. Therefore, by deleting any one equation from Equation (1.18) and substituting Equation (1.19), the result is the set of  $\text{card}(\Omega)$  linearly independent equations

$$\pi_k(\mathbf{I} - \mathbf{P}_k)^{[i]} = \mathbf{e}_i, \quad (1.20)$$

where the square matrix  $(\mathbf{I} - \mathbf{P}_k)^{[i]}$  is obtained by substituting the  $i$ th column of matrix  $(\mathbf{I} - \mathbf{P}_k)$  with a column vector of ones. The vector  $\mathbf{e}_i$  is a row vector of zeroes, except for a one in the  $i$ th position. Since  $(\mathbf{I} - \mathbf{P}_k)^{[i]}$  is of full rank, then its determinant (written as  $\det((\mathbf{I} - \mathbf{P}_k)^{[i]})$ ) is non-zero. Define  $(\mathbf{I} - \mathbf{P}_k)^\omega$  to be the same matrix as  $(\mathbf{I} - \mathbf{P}_k)$  except that the elements of the  $\omega$ th row of  $(\mathbf{I} - \mathbf{P}_k)$  are replaced by the vector  $\mathbf{e}_\omega$ . Therefore, for all iterations  $k$ ,

$$\pi_k(\omega) = \frac{\det((\mathbf{I} - \mathbf{P}_k)^{[i]\omega})}{\det((\mathbf{I} - \mathbf{P}_k)^{[i]}), \text{ for all } \omega \in \Omega. \quad (1.21)}$$

In [38], an attempt is made to solve Equation (1.21) for each  $\omega \in \Omega$  via a multivariate Taylor series expansion of each determinant, but the method failed to produce a closed-form analytical expression.

Overall, the difficulty of explicitly expressing the stationary distributions for large solution spaces, combined with bounding the transition matrix condition number for large  $k$ , suggests that it is very difficult to prove asymptotic convergence of the simulated annealing algorithm by treating Equations (1.5) and (1.6) as a linear algebra problem.

Lundy and Mees [97] note that for each fixed outer loop iteration  $k$ , convergence to the solution equilibrium probability distribution vector  $\pi_k$  (in terms of the Euclidean distance between  $P_k^{(m)}$  and  $\pi_k$ , as  $m \rightarrow +\infty$ ) is geometric since the solution space is finite, and the convergence factor is given by the second largest eigenvalue of the transition matrix  $\mathbf{P}_k$ . This result is based on a standard convergence theorem for irreducible, aperiodic homogeneous Markov chains (see [33]). Note that a large solution space precludes practical calculation of this eigenvalue. Lundy and Mees [97] conjecture that when the temperature  $t_k$  is near zero, the second largest eigenvalue will be close to one for problems with local optima, and thus convergence to the equilibrium distribution will be very slow (recall that the dominant eigenvalue for  $\mathbf{P}_k$  is 1, with algebraic multiplicity 1 [78]). Lundy and Mees [97] use the conjecture to justify why simulated annealing should be initiated with a relatively high temperature. For an overview of current methods for assessing non-asymptotic rates of convergence for general homogeneous Markov chains, see [121].

The assumption of stationarity for each outer loop iteration  $k$  limits practical application of homogeneous Markov chain theory. Romeo and Sangiovanni-Vincentelli [120] show that if equilibrium (for a Markov chain that satisfies the

reversibility condition) is reached in a finite number of steps, then it can be achieved in one step. Thus, Romeo and Sangiovanni-Vincentelli [120] conjecture that there is essentially no hope for the most used versions of simulated annealing to reach equilibrium in a finite number of iterations.

### 1.2.1.2 Inhomogeneous Markov Chain Approach

The second convergence approach for simulated annealing is based on inhomogeneous Markov chain theory [10, 65, 104]. In this approach, the Markov chain need not reach a stationary distribution (e.g., the simulated annealing inner loop need not be infinitely long) for each outer loop  $k$ . On the other hand, an infinite sequence of (outer loop) iterations  $k$  must still be examined, with the condition that the temperature parameter  $t_k$  cool sufficiently slowly. The proof given by Mitra et al. [104] is based on satisfying the inhomogeneous Markov chain conditions of weak and strong ergodicity [78, 127]. The proof requires four conditions:

1. The inhomogeneous simulated annealing Markov chain must be weakly ergodic (i.e., dependence on the initial solution vanishes in the limit).
2. An eigenvector  $\pi_k$  with eigenvalue 1 must exist such that Equations (1.6) and (1.7) hold for every iteration  $k$ .
3. The Markov chain must be strongly ergodic (i.e., the Markov chain must be weakly ergodic and the sequence of eigenvectors  $\pi_k$  must converge to a limiting form), i.e.,

$$\sum_{k=0}^{\infty} \|\pi_k - \pi_{k+1}\| < +\infty. \quad (1.22)$$

4. The sequence of eigenvectors must converge to a form where all probability mass is concentrated on the set of globally optimal solutions  $\omega^*$ . Therefore,

$$\lim_{k \rightarrow \infty} \pi_k = \pi^{\text{opt}}, \quad (1.23)$$

where  $\pi^{\text{opt}}$  is the equilibrium distribution with only global optima having probabilities greater than 0. (Note that weak and strong ergodicity are equivalent for homogeneous Markov chain theory.)

Mitra et al. [104] satisfy condition 1 (weak ergodicity) by first forming a lower bound on the probability of reaching any solution from any local minimum and then showing that this bound does not approach zero too quickly. For example, they define the lower bound for the simulated annealing transition probabilities in Equation (1.5) as

$$P^{(m)}(\omega, \omega') \geq w^m \exp(-m\Delta_L/t_{km-1}), \quad (1.24)$$

for any integer  $k$  greater than or equal to some fixed integer  $k_0$ , where  $m$  is the number of transitions needed to reach any solution from any solution of non-maximal objective function value,  $w > 0$  is a lower bound on the one-step solution generation probabilities,  $\Delta_L$  is the maximum one-step increase in objective function value

between any two solutions, and  $t_{km-1}$  is a temperature at iteration  $km - 1$ . Mitra et al. [104] show that the Markov chain is weakly ergodic if for any fixed integer  $k_0$

$$\sum_{k=k_0}^{\infty} \exp(-m\Delta_L/t_{km-1}) = +\infty. \quad (1.25)$$

Therefore, weak ergodicity is obtained if the temperature  $t_k$  is reduced sufficiently slowly to zero such that Equation (1.25) is satisfied. In general, the (infinite) sequence of temperatures  $\{t_k\}$ ,  $k = 1, 2, \dots$ , must satisfy

$$t_k \geq \frac{\beta}{\log(k)}, \quad (1.26)$$

where  $\lim_{k \rightarrow \infty} t_k = 0$ ,  $\beta$  is a problem-dependent constant and  $k$  is the number of iterations. Mitra et al. [104] show that conditions (1.2), (1.3), and (1.4) are satisfied by using the homogeneous Markov chain theory developed for the transition probabilities Equation (1.5), provided that the solution generation function is symmetric.

Romeo and Sangiovanni-Vincentelli [120] note that while the logarithmic cooling schedule in Equation (1.26) is a sufficient convergence condition, there are only a few values for  $\beta$  which make the logarithmic rule also necessary. Furthermore, there exists a unique choice for  $\beta$  which makes the logarithmic rule both necessary and sufficient for the convergence of simulated annealing to the set of global optima. In [72], Hajek was the first to show that the logarithmic cooling schedule (Equation (1.26)) is both necessary and sufficient, by developing a tight lower bound for  $\beta$ , namely the depth of the deepest local minimum which is not a global minimum, under a weak reversibility assumption (note that Hajek requires the depth of global optima to be infinitely large). Hajek defines a Markov chain to be weakly reversible if, for any pair of solutions  $\omega, \omega' \in \Omega$  and for any non-negative real number  $h$ ,  $\omega$  is reachable at height  $h$  from  $\omega'$  if and only if  $\omega'$  is reachable at height  $h$  from  $\omega$ . Note that Hajek [72] does not attempt to satisfy the conditions of weak and strong ergodicity, but rather uses a more general probabilistic approach to develop a lower bound on the probability of escaping local, but not global, optima. Connors and Kumar [35] substantiate the necessary and sufficient conditions in Hajek [72] using the orders of recurrence,

$$B_i \equiv \sup \left\{ x \geq 0 : \sum_{k=0}^{\infty} \exp(-x/t_k) \pi_k(\omega) = +\infty \right\} \text{ for all } i \in \Omega. \quad (1.27)$$

Connors and Kumar [35] show that these orders of recurrence quantify the asymptotic behavior of each solution's probability in the solution distribution. The key result is that the simulated annealing inhomogeneous Markov chain converges in a Cesaro sense to the set of solutions having the largest recurrence orders. Borkar [16] improves this convergence result by using a convergence/oscillation dichotomy result for martingales. Tsitsiklis [150] uses bounds and estimates for singularly perturbed, approximately stationary Markov chains to develop a convergence theory

that subsumes the condition of weak reversibility in [72]. Note that Tsitsiklis [150] defines  $N(h) \subset \Omega$  as the set of all local minima (in terms of objective function value) of depth  $h + 1$  or more. Therefore  $\beta$  is the smallest  $h$  such that all local (but not global) minima have depth  $h$  or less. Tsitsiklis conjectures that without some form of reversibility, there does not exist any  $h$  such that the global optima are contained in the set of local optima. Note that in [16, 28, 30, 35, 72, 104], the multiplicative condition (1.9) is required (either explicitly or implicitly) for the proofs of convergence.

Anily and Federgruen [10] use perturbation analysis techniques (e.g., see [102]) to prove convergence of a particular stochastic hill-climbing algorithm by bounding the deviations of the sequence of stationary distributions of the particular hill-climbing algorithm against the sequence of known stationary distributions corresponding to a simulated annealing algorithm. In general, this convergence proof approach is only useful for a restrictive class of simulated annealing algorithms, since the transition matrix condition number grows exponentially as the number of iterations  $k$  becomes large.

Anily and Federgruen [10] also present a proof of convergence for simulated annealing with general acceptance probability functions. Using inhomogeneous Markov chain theory, they prove convergence under the following necessary and sufficient conditions:

1. The acceptance probability function must, for any iteration  $k$ , allow any hill-climbing transition to occur with positive probability.
2. The acceptance probability function must be bounded and asymptotically monotone, with limit zero for hill-climbing solution transitions.
3. In the limit, the stationary probability distribution must have 0 probability mass for every non-globally optimal solution.
4. The probability of escaping from any locally (but not globally) optimal solution must not approach 0 too quickly.

Anily and Federgruen [10] use condition (3) to relax the acceptance function multiplicative condition (1.9). However, in practice, condition (3) would be very difficult to check without assuming that Equation (1.9) holds. Condition (4) provides the necessary condition for the rate that the probability of hill-climbing transitions approaches 0. Condition (4) is expressed quantitatively as follows: let  $t_k$  be defined by Equation (1.2) and define the minimum one-step acceptance probability as

$$\underline{a}_k = \min_{\omega \in \Omega, \omega' \in N(\omega)} a_{t_k}(\omega, \omega'). \quad (1.28)$$

Define the set of local optima  $L \subset \Omega$  such that  $\omega \in L$  implies that  $f(\omega) \leq f(\omega')$  for all  $\omega' \in N(\omega)$ , and let

$$\bar{a}_k = \min_{\omega \in L, \omega' \in N(\omega) \setminus L} a_{t_k}(\omega, \omega'). \quad (1.29)$$

Finally, let any solution  $\omega' \in \Omega$  be reachable from any solution  $\omega \in \Omega$  in  $q$  transitions or less. If (non-globally) locally optimal solutions exist,

$$\sum_{k=1}^{\infty} (\underline{a}_k)^q = +\infty, \quad (1.30)$$

and conditions (1), (2), and (3) hold, then the simulated annealing algorithm will asymptotically converge to the set of global optima with probability 1. However, if (non-globally) locally optimal solutions exist and

$$\sum_{k=1}^{\infty} \bar{a}_k < +\infty, \quad (1.31)$$

then the simulated annealing algorithm will not always converge to the set of global optima with probability 1. Johnson and Jacobson [85] relax the sufficient conditions found in [10] by using a path argument between global optima and local (but not global) optima.

Yao and Li [165] and Yao [164] also discuss simulated annealing algorithms with general acceptance probabilities, though their primary contribution is with respect to general neighborhood generation distributions. In [126], Schuur provides a description of acceptance functions ensuring the convergence of the associated simulated annealing algorithm to the set of global optima.

The inhomogeneous proof concept is stronger than the homogeneous approach in that it provides necessary conditions for the rate of convergence, but its asymptotic nature suggests that practical implementation may not be feasible. Romeo and Sangiovanni-Vincentelli [120] note that “there is no reason to believe that truncating the logarithmic temperature sequence would yield a good configuration, since the tail of the sequence is the essential ingredient in the proof.” In addition, the logarithmic cooling schedule dictates a very slow rate of convergence. Therefore, most recent work has focused on methods of improving simulated annealing’s finite-time behavior and modifying or blending the algorithm with other search methods such as genetic algorithms [92], tabu search [66], or both [59].

### 1.2.2 Finite-Time Performance

Over the past decade, a growing body of work has been devoted to finite-time behavior of simulated annealing. Jacobson and Yucesan [82] present necessary and sufficient (asymptotic) convergence conditions for generalized hill climbing algorithms that include simulated annealing as a special case. They also introduce new performance measures that can be used to evaluate and compare both convergent and non-convergent generalized hill-climbing algorithms with random restart local search [79]. Such a comparison provides insights into both asymptotic and finite-time performance of discrete optimization algorithms. For example, they use the global visit probability to evaluate the performance of simulated annealing using random restart local search as a benchmark. These results suggest that random restart local search may outperform simulated annealing provided that a sufficiently large number of

restarts are executed. In [60], Fox notes that a similar result that compares random restart local search with simulated annealing can be true but only if both the number of accepted and rejected moves are counted. A clever example is also provided in [60] to illustrate this point, which illustrates that comparing random restart local search and simulating annealing may not be prudent. In [59] and [61], Fox presents modifications of simulated annealing that circumvent the counting issue described in [60], hence yielding superior performing simulated annealing algorithm implementations. The primary value of using simulated annealing may therefore be for finite-time executions that obtain near-optimal solutions reasonably quickly. This, in turn, suggests that studying the finite-time behavior of simulated annealing is equally important as its asymptotic convergence.

Chiang and Chow [29] and Mazza [100] investigate the statistical properties of the first visit time to a global optimum, which provides insight into asymptotic properties of the algorithm as the outer loop counter  $k \rightarrow +\infty$ . In [20], Catoni investigates optimizing a finite-horizon cooling schedule to maximize the number of visits to a global optimum after a finite number of iterations. In [44], Desai focuses on finite-time performance by incorporating size-asymptotic information supplied by certain eigenvalues associated with the transition matrix. Desai [44] also provides some quantitative and qualitative information about the performance of simulated annealing after a finite number of steps, by observing the quality of solutions is related to the number of steps that the algorithm has taken.

Srichander [133] examines the finite-time performance of simulated annealing using spectral decomposition of matrices. He proposes that an annealing schedule on the temperature is not necessary for the final solution of the simulated annealing algorithm to converge to the global minimum with probability 1. Srichander shows that initiating the simulated annealing algorithm with high initial temperatures produces an inefficient algorithm in the sense that the number of function evaluations required to obtain a global minima is very large. A modified simulated annealing algorithm is presented with a low initial temperature and an iterative schedule on the size of the neighborhood sets that leads to a more efficient algorithm. The new algorithm is applied to a real-world example and computational performance is reported.

Fleischer and Jacobson [58] use a reverse approach to establish theoretical relationships between the finite-time performance of an algorithm and the characteristics of problem instances. They observe that the configuration space created by an instance of a discrete optimization problem determines the efficiency of simulated annealing when applied to that problem. The entropy of the Markov chain embodying simulated annealing is introduced as a measure that captures the entire topology of the configuration space associated with the particular instance of the discrete optimization problem. By observing the expected value of the final state in a simulated annealing algorithm as it relates to the entropy value of the underlying Markov chain, they present measures of performance that determine how well the simulated annealing algorithm performs in finite time. Their computational results suggest that superior finite time performance of a simulated annealing algorithm are associated with higher entropy measures.

Nolte and Schrader [111] give a proof of the convergence of simulated annealing by applying results about rapidly mixing Markov chains. With this proof technique, it is possible to obtain better bounds for the finite-time behavior of simulated annealing than previously known.

To evaluate the expected run-time required by a simulated annealing algorithm to reach solution of a pre-specified quality, Wood et al. [161] present an approach to model and analyze a generic stochastic global optimization algorithm using a sequence of stochastic processes, culminating in a backtracking adaptive search process. The theory developed for this backtracking adaptive search procedure is then used to analyze the classic simulated annealing algorithm.

In [118], Rajasekaran presents an analysis of simulated annealing that provides a time bound for convergence with very high probability. Convergence of simulated annealing in the limit then follows as a corollary to the established finite-time performance results.

## 1.3 Relationship to Other Local Search Algorithms

The hill-climbing strategy inherent in simulated annealing has lead to the formulation of other such algorithms (e.g., threshold accepting, the noising method). Moreover, though different in how they traverse the solution space, both tabu search and genetic algorithms share with simulated annealing the objective of using local information to find global optima over solution spaces with multiple local optima.

### 1.3.1 Threshold Accepting

Questioning the very need for a randomized acceptance function, Dueck and Scheuer [45] and, independently, Moscato and Fontanari [106] propose the threshold accepting algorithm, where the acceptance probability function is

$$a_k(\Delta_{\omega, \omega'}) = \begin{cases} 1 & \text{if } Q_k \geq \Delta_{\omega, \omega'} \\ 0 & \text{otherwise} \end{cases},$$

with  $Q_k$  defined as the threshold value at iteration  $k$ .  $Q_k$  is typically set to be a deterministic, non-increasing step function in  $k$ . Dueck and Scheuer [45] report computational results that suggest dramatic improvements in traveling salesman problem solution quality and algorithm run-time over basic simulated annealing. Moscato and Fontanari [106] report more conservative results—they conjecture that simulated annealing’s probabilistic acceptance function does not play a major role in the search for near-optimal solutions.

Althofer and Koschnick [8] develop a convergence theory for threshold accepting based on the concept that simulated annealing belongs to the convex hull of

threshold accepting. The idea presented in [8] is that (for a finite  $Q_k$  threshold sequence) there can exist only finitely many threshold accepting transition matrices, but simulated annealing can have infinitely many transition matrices because of the real-valued nature of the temperature at each iteration. However, every simulated annealing transition matrix for a given problem can be represented as a convex combination of the finitely many threshold accepting transition matrices. Althofer and Koschnick [8] are unable to prove that threshold accepting will asymptotically reach a global minimum, but it does prove the existence of threshold schedules that provide convergence to within an  $\varepsilon$ -neighborhood of the optimal solutions. Jacobson and Yucesan [81] prove that if the threshold value approaches 0 as  $k$  approaches infinity, then the algorithm does not converge in probability to the set of globally optimal solutions.

Hu et al. [77] modify threshold accepting to include a non-monotonic, self-tuning threshold schedule in the hope of improving the algorithm's finite-time performance. Hu et al. allow the threshold  $Q_k$  to change dynamically (either up or down), based on the perceived likelihood of being near a local minimum. These changes are accomplished using a principle they call *dwindling expectation*—when the algorithm fails to move to neighboring solutions, the threshold  $Q_k$  is gradually increased, in the hope of eventually escaping a local optimum. Conversely, when solution transitions are successful, the threshold is reduced, in order to explore local optima. The experimental results based on two traveling salesman problems presented in [77] showed that the proposed algorithm outperformed previous hill-climbing methods in terms of finding good solutions earlier in the optimization process.

Threshold accepting's advantages over simulated annealing lie in its ease of implementation and its generally faster execution time, due to the reduced computational effort in avoiding acceptance probability computations and the generation of random numbers [106]. However, compared to simulated annealing, relatively few threshold accepting applications are reported in the literature [93, 110, 125].

### 1.3.2 Noising Method

Charon and Hudry [23] advocate a simple descent algorithm called the *noising method*. The algorithm first perturbs the solution space by adding random noise to the problem's objective function values. The noise is gradually reduced to 0 during the algorithm's execution, allowing the original problem structure to reappear. Charon and Hudry provide computational results, but do not prove that the algorithm will asymptotically converge to the set of globally optimal solutions. Charon and Hudry [24] show how the noising method is a generalization of simulated annealing and threshold accepting.

Storer et al. [136] propose an optimization strategy for sequencing problems, by integrating fast, problem-specific heuristics with local search. Its key contribution is to base the definition of the search neighborhood on a heuristic problem pair  $(h, p)$ , where  $h$  is a fast, known, problem-specific heuristic and  $p$  represents the problem data. By perturbing the heuristic, the problem, or both, a neighborhood of solutions

is developed. This neighborhood then forms the basis for local search. The hope is that the perturbations will cluster good solutions close together, thus making it easier to perform local search.

### 1.3.3 Tabu Search

Tabu search [66] is a general framework for a variety of iterative local search strategies for discrete optimization. Tabu search uses the concept of *memory* by controlling the algorithm's execution via a dynamic list of forbidden moves. This allows the tabu search algorithm to intensify or diversify its search of a given problem's solution space in an effort to avoid entrapment in local optima. See [67] for a discussion on the convergence of tabu search algorithms.

Given that simulated annealing is completely memoryless (i.e., simulated annealing disregards all historical information gathered during the algorithm's execution), tabu search provides an alternative mechanism to hill-climb and escape local optima. Faigle and Kern [50] propose a particular tabu search algorithm called *probabilistic tabu search* as a meta-heuristic to help guide simulated annealing. Probabilistic tabu search attempts to capitalize on both the asymptotic optimality of simulated annealing and the memory feature of tabu search. In probabilistic tabu search, the probabilities of generating and accepting each candidate solution are set as functions of both a temperature parameter (as in simulated annealing) and information gained in previous iterations (as in tabu search). Faigle and Kern [50] are then able to prove asymptotic convergence of their particular tabu search algorithm by using methods developed for simulated annealing [49, 52]. Note that the results in [50] build upon work by Glover [67] where probabilistic tabu search was first introduced and contrasted with simulated annealing.

Vaughan and Jacobson [158] develop a framework, termed tabu guided generalized hill climbing, that uses a tabu release parameter that probabilistically accepts solutions currently on the tabu list. The presented algorithms are modeled as a set of stationary Markov chains, where the tabu list is fixed for each outer loop iteration. This framework provides practitioners with guidelines for developing tabu search strategies to use in conjunction with generalized hill-climbing algorithms that preserve some of the algorithms' known performance properties. Sufficient conditions are obtained that indicate how to design iterations for problem-specific tabu search strategies.

### 1.3.4 Genetic Algorithms

Genetic algorithms [92] emulate the evolutionary behavior of biological systems. They generate a sequence of populations of candidate solutions to the underlying optimization problem by using a set of genetically inspired stochastic solution transition operators to transform each population of candidate solutions into a

descendent population. The three most popular transition operators are reproduction, cross-over, and mutation [38]. Davis and Principe [39] and Rudolph [124] attempt to use homogeneous finite Markov chain techniques to prove convergence of genetic algorithms [21], but are unable to develop a theory comparable in scope to that of simulated annealing.

Zolfaghari and Liang [167] undertake a comparative study of simulated annealing, genetic algorithms, and tabu search for solving binary (considering only machines and part types) machine-grouping problems of varying types (involving machine/part types, processing times, lot sizes, and machine capacities). To test the performance of the three metaheuristics, two binary performance indices and two generalized performance indices are used for binary and comprehensive machine/part grouping problems, respectively. The comparisons are made in terms of solution quality, search convergence behavior, and pre-search effort. The results indicate that simulated annealing outperforms both genetic algorithm and tabu search, particularly for large problems.

In [107], Muhlenbein presents a theoretical analysis of genetic algorithms based on population genetics. He counters the popular notion that models that mimic natural phenomenon are superior to other models. The article argues that evolutionary algorithms can be inspired by nature, but do not necessarily have to copy a natural phenomenon. He addresses the behavior of transition operators and designs new genetic operators that are not necessarily related to events in nature, yet still perform well in practice.

One criticism of simulated annealing is the slow speed at which it converges. In [41], Delpot combines simulated annealing with evolutionary algorithms to improve performance in terms of speed and solution quality. The benefit of this hybrid system of simulated annealing and evolutionary selection is due to the adjustments in the cooling schedule based on fast recognition of the thermal equilibrium in terms of selection intensity, which results in much faster convergence of the algorithm.

Sullivan and Jacobson [139, 140] link genetic algorithms with simulated annealing using generalized hill-climbing algorithms [80]. They first link genetic algorithms to ordinal hill-climbing algorithms, which can then be used, through its formulation within the generalized hill-climbing algorithm framework, to form a bridge with simulated annealing. Though genetic algorithms have proven to be effective for addressing intractable discrete optimization problems and can be classified as a type of hill-climbing approach, its link with generalized hill-climbing algorithms (through the ordinal hill-climbing formulation) provides a means to establish well-defined relationships with other generalized hill-climbing algorithms (like simulated annealing and threshold accepting). They also present two formulations of genetic algorithms that provide a first step toward developing a bridge between genetic algorithms and other local search strategies like simulated annealing.

### ***1.3.5 Generalized Hill-Climbing Algorithms***

Generalized hill-climbing algorithms (GHC) (see [80]) provide a framework for modeling local search algorithms used to address intractable discrete optimization

problems. All generalized hill-climbing algorithms have the same basic structure, but can be tailored to a specific instance of a problem by changing the hill-climbing random variable (which is used to accept or reject inferior solutions) and neighborhood functions. Generalized hill-climbing algorithms are described in pseudo-code form:

Select an initial solution  $\omega \in \Omega$

Set the outer loop counter bound  $K$  and the inner loop counter bounds  $M_k$ ,  $k = 1, 2, \dots, K$

Define a set of hill-climbing (random) variables  $R_k : \Omega \times \Omega \rightarrow (-\infty, +\infty)$ ,  $k = 1, 2, \dots, K$

Set the iteration indices  $k = m = 1$

Repeat while  $k \leq K$

Repeat while  $m \leq M_k$

Generate a solution  $\omega' \in N(\omega)$

Calculate  $\Delta_{\omega, \omega'} = f(\omega') - f(\omega)$

If  $R_k(\omega, \omega') \geq \Delta_{\omega, \omega'}$ , then  $\omega \leftarrow \omega'$

$m \leftarrow m + 1$

Until  $m = M_k$

$m \leftarrow 1, k \leftarrow k + 1$

Until  $k = K$

Note that the outer and inner loop bounds,  $K$  and  $M_k$ ,  $k = 1, 2, \dots, K$ , respectively, may all be fixed, or  $K$  can be fixed with the  $M_k$ ,  $k = 1, 2, \dots, K$ , defined as random variables whose values are determined by the solution at the end of each set of inner loop iterations satisfying some property (e.g., the solution is a local optima).

Generalized hill-climbing algorithms can be viewed as sampling procedures over the solution space  $\Omega$ . The key distinction between different generalized hill-climbing algorithm formulations is in how the sampling is performed. For example, simulated annealing produces biased samples, guided by the neighborhood function, the objective function, and the temperature parameter. More specifically, simulated annealing can be described as a generalized hill-climbing algorithm by setting the hill-climbing random variable,  $R_k(\omega, \omega') = -t_k \ln(u_k)$ ,  $\omega \in \Omega$ ,  $\omega' \in N(\omega)$ ,  $k = 1, 2, \dots, K$ , with the  $\{u_k\}$  independently and identically distributed  $U(0, 1)$  random variables. To formulate Monte Carlo search as a generalized hill-climbing algorithm, set  $R_k(\omega, \omega') = +\infty$ ,  $\omega \in \Omega$ ,  $\omega' \in N(\omega)$ ,  $k = 1, 2, \dots, K$ . Deterministic local search, which accepts only neighbors of improving (lower) objective function value, can be expressed as a generalized hill-climbing algorithm with  $R_k(\omega, \omega') = 0$ ,  $\omega \in \Omega$ ,  $\omega' \in N(\omega)$ ,  $k = 1, 2, \dots, K$ . Other algorithms that can be described using the generalized hill-climbing framework include threshold accepting, some simple forms of tabu search, and Weibull accepting. For detailed discussions of these algorithms and a description of how they fit into the generalized hill-climbing algorithm framework, see [80, 85, 139].

Measures for assessing the finite-time performance of generalized hill-climbing algorithms have been developed, including the expected number of iterations to visit a predetermined objective function value level. Jacobson et al. [83] introduce the cyclical simulated annealing algorithm and describe a procedure to estimate

lower and upper bounds for the expected number of iterations to visit a near-optimal solution for this algorithm. Computational results with four traveling salesman problem instances are reported.

## 1.4 Practical Guidelines

Implementation issues for simulated annealing can follow one of two paths—that of specifying problem-specific choices (objective function and neighborhood) and that of specifying generic choices (generation and acceptance probability functions and the cooling schedule) (see [46]). These choices often depend on the domain of each specific problem. The principal shortcoming of simulated annealing is that it often requires extensive computer time. Implementation modifications generally strive to retain simulated annealing’s asymptotic convergence character, but at reduced computer run-time. The methods discussed here are mostly heuristic.

### 1.4.1 Problem-Specific Choices

#### 1.4.1.1 Objective Functions

One problem-specific choice involves the objective function specification. In [135], Stern recommends a heuristic temperature-dependent penalty function as a substitute for the actual objective function for problems where low cost solutions have neighbors of much higher cost or in cases of degeneracy (i.e., large neighborhoods of solutions of equal, but high costs). The original objective function surfaces, as the penalty and the temperature are gradually reduced to 0. This technique is similar to the noising method presented by Charon and Hudrey in [23], where the penalty function is described as noise and is reduced at each outer loop iteration of the algorithm. One speed-up technique is to evaluate only the difference in objective functions,  $\Delta_{\omega,\omega'}$ , instead of calculating both  $f(\omega)$  and  $f(\omega')$ . In [148], Tovey suggests several methods of approximating  $\Delta_{\omega,\omega'}$  by using surrogate functions (that are faster to evaluate than  $\Delta_{\omega,\omega'}$ , but not as accurate) probabilistically for cases when evaluation of  $\Delta_{\omega,\omega'}$  is expensive; this technique is referred to as the *surrogate function swindle*.

Straub et al. [137] improve the performance of simulated annealing on problems in chemical physics by using the continuous normal density distribution instead of the molecular dynamics of single point particles to describe the potential energy landscape. Ma and Straub [98] report that using this distribution has the effect of smoothing the energy landscape by reducing both the number and depth of local minima.

Yan and Mukai [162] consider the case when a closed-form formula for the objective function is not available. They use a probabilistic simulation (termed the

stochastic ruler method) to generate a sample objective function value for an input solution and then accept the solution if the sample objective function value falls within a predetermined bound. They also provide a proof of asymptotic convergence by extrapolating the convergence proofs for simulated annealing and analyze the rate of convergence.

#### 1.4.1.2 Neighborhoods

A key problem-specific choice concerns the neighborhood function definition. The efficiency of simulated annealing is highly influenced by the neighborhood function used [105]. The choice of neighborhood serves to enforce a topology. In [46], Eglese reports that “a neighborhood structure which imposes a ‘smooth’ topology where the local minima are shallow is preferred to a ‘bumpy’ topology where there are many deep local minima.” Solla et al. [132] and Fleischer and Jacobson [58] report similar conclusions. This also supports the result in [72] that shows that asymptotic convergence to the set of global optima depends on the depth of the local minima.

Another factor to consider when choosing neighborhood functions is the neighborhood size. No theoretical results are available, other than the necessity of reachability (in a finite number of steps) from any solution to any other solution. Cheh et al. [25] report that small neighborhoods are best, while Ogbu and Smith [113] provide evidence that larger neighborhoods result in better simulated annealing performance. Goldstein and Waterman [68] conjecture that if the neighborhood size is small compared to the total solution space cardinality, then the Markov chain cannot move around the solution space fast enough to find the minimum in a reasonable time. On the other hand, a very large neighborhood has the algorithm merely sampling randomly from a large portion of the solution space and thus is unable to focus on specific areas of the solution space. It is reasonable to believe that neighborhood size is heavily problem specific. For example, problems where the smoothness of its solution space topology is moderately insensitive to different neighborhood definitions may benefit from larger neighborhood sizes.

Concepts from information theory are used in [54] and [58] to show that the neighborhood structure can affect the *information rate* or total uncertainty associated with simulated annealing. In [54], Fleischer shows that simulated annealing tends to perform better as the entropy level of the associated Markov chain increases and thus conjectures that an entropy measure could be useful for predicting when simulated annealing would perform well on a given problem. However, efficient ways of estimating the entropy are needed to transform this result into a practical tool.

Triki et al. [151] present an empirical study on the efficiency of the simulated annealing algorithm, as impacted by the landscape and the choice of the neighborhood function. The experiments they conducted follow the observation that it is possible to compute the exact probability for the algorithm to reach any point in the landscape, provided that the number of solutions and the number of neighbors per solution are sufficiently small. A developed computational tool allows to study the

influence of the tuning of all the main parameters of simulated annealing, as well as theoretical concepts such as thermodynamic equilibrium and optimal temperature decrement rules.

Bouffard and Ferland [17] propose a method to improve the simulated annealing algorithm with a variable neighborhood search to solve the resource-constrained scheduling problem. The method is compared numerically with other neighborhood search techniques: threshold accepting methods and tabu search. Furthermore, these techniques are combined with multi-start diversification strategies. The numerical results indicate that using a variable neighborhood search technique indeed improves the performance.

Another issue on neighborhood function definition addresses the solution space itself. Chardaïre et al. [22] propose a method for addressing 0 – 1 optimization problems, in which the solution space is progressively reduced by fixing the value of *strongly persistent* variables (which have the same value in all optimal solutions). They isolate the persistent variables during simulated annealing's execution by periodically estimating the expectation of the random variable (a vector of binary elements) that describes the current solution and fixing the value of those elements in the random variable that meet threshold criteria.

## 1.4.2 Generic Choices

### 1.4.2.1 Generation Probability Functions

Generation probability functions are usually chosen as uniform distributions with probabilities proportional to the size of the neighborhood. The generation probability function is typically not temperature dependent. In [59], Fox suggests that instead of blindly generating neighbors uniformly, adopt an intelligent generation mechanism that modifies the neighborhood and its probability distribution to accommodate search intensification or diversification, in the same spirit of the tabu search metaheuristic. Fox also notes that simulated annealing convergence theory does not preclude this idea. In [148], Tovey suggests an approach with a similar effect, called the *neighborhood prejudice swindle*.

### 1.4.2.2 Acceptance Probability Functions

The literature reports considerable experimentation with acceptance probability functions for hill-climbing transitions. The most popular is the exponential form (1.1). Ogbu and Smith [113] consider replacing the basic simulated annealing acceptance function  $a_k(\Delta_{\omega, \omega'})$  with a geometrically decreasing form that is independent of the change in objective function value. They adopt a *probabilistic-exhaustive* heuristic technique in which randomly chosen neighbors of a solution are examined and all solutions that are accepted are noted, but only the last solution accepted

becomes the new incumbent. The hope is that this scheme will explore a broader area of the problem solution space. Their acceptance probability function is defined for all solutions  $\omega, \omega' \in \Omega$  and for  $k = 1, 2, \dots, K$  as

$$a_k(\Delta_{\omega, \omega'}) = a_k = \begin{cases} a_1 x^{k-1} & \text{if } f(\omega') > f(\omega) \\ 1 & \text{otherwise} \end{cases},$$

where  $a_1$  is the initial acceptance probability value,  $x \in (0, 1)$  is a reducing factor, and  $K$  is the number of stages (equivalent to a temperature cooling schedule). They also experiment with this method (and a neighborhood of large cardinality) on a permutation flow shop problem and report that its approach found comparable solutions to the basic simulated annealing algorithm in one-third the computation time.

#### 1.4.2.3 Cooling Schedules

The simulated annealing cooling schedule is fully defined by an initial temperature, a schedule for reducing/changing the temperature, and a stopping criterion. Romeo and Sangiovanni-Vincentelli [120] note that an effective cooling schedule is essential to reducing the amount of time required by the algorithm to find an optimal solution. Therefore, much of the literature on cooling schedules (see [19, 34, 62, 112]) is devoted to this efficiency issue.

Homogeneous simulated annealing convergence theory has been used to design effective cooling schedules. Romeo and Sangiovanni-Vincentelli [120] suggest the following procedure for designing a cooling schedule:

1. Start with an initial temperature  $t_0$  for which a good approximation of the stationary distribution  $\pi_{t_0}$  is quickly reached.
2. Reduce  $t_0$  by an amount  $\delta(t)$  small enough such that  $\pi_{t_0}$  is a good starting point to approximate  $\pi_{t_0 - \delta(t)}$ .
3. Fix the temperature at a constant value during the iterations needed for the solution distribution to approximate  $\pi_{t_0 - \delta(t)}$ .

Repeat the above process of cooling and iterating until no further improvement seems possible.

Generally, the initial temperature is set such that the acceptance ratio of bad moves is equal to a certain value. In [14], Ben-Ameur proposes an algorithm to compute a temperature which is compatible with a given acceptance ratio. The acceptance probability function is shown to be convex for low temperatures and concave for high temperatures, and a lower bound is provided for the number of required temperature changes based on a geometric cooling schedule.

Cooling schedules are grouped into two classes: static schedules, which must be completely specified before the algorithm begins; and *adaptive* schedules, which adjust the temperature's rate of decrease from information obtained during the algorithm's execution. Cooling schedules are almost always heuristic; they seek to balance moderate execution time with simulated annealing's dependence on asymptotic behavior.

Strenski and Kirkpatrick [138] present an exact (non-heuristic) characterization of finite-length annealing schedules. They consider extremely small problems that represent features (local optima and smooth/hilly topologies) and determine probability distribution of outcomes of annealing process in a finite number of iterations to gain insights into some popular assumptions and intuition behind cooling schedules. Their experiments suggest that optimal cooling schedules are not monotone decreasing in temperature. They also show that for the test problem (a white noise surface), geometric and linear cooling schedules perform better than inverse logarithmic cooling schedules, when sufficient computing effort is allowed. Moreover, their experiments do not show measurable performance differences between linear and geometric cooling schedules. They also observe that geometric cooling schedules are not greatly affected by excessively high initial temperatures. The results presented suggest that even the most robust adaptive cooling schedule “produces annealing trajectories which are never in equilibrium” [138]. However, they also conclude that the transition acceptance rate is not sensitive to the degree of closeness to the equilibrium distribution.

Christoph and Hoffmann [31] also attempt to characterize optimal cooling schedules. They derive a relationship between a finite sequence of optimal temperature values (i.e., outer loops) and the number of iterations (i.e., inner loops) at each respective temperature for several small test problems to reach optimality (i.e., the minimal mean final energy). They find that this *scaling behavior* is of the form

$$x_m = a_m v^{-b_m}, \quad (1.32)$$

where  $a$  and  $b$  are scaling coefficients,  $x_m = \exp(-1/t_k)$  is referred to as the temperature,  $v$  is the number of inner loop iterations at temperature  $x_m$ , and  $m$  is the number of outer loops at which the temperature  $x_m$  is reduced. The proposed approach is to solve for the coefficients  $a$  and  $b$  based on known temperature and iteration parameter values for an optimal schedule based on several replications of the algorithm using  $(m \times v)$  iterations for each replication, and then use Equation (1.32) to interpolate the optimal cooling schedule for intermediate iterations. They however do not make any suggestions on how to efficiently solve for the necessary optimal cooling schedules for a (typically large) problem instance.

Romeo and Sangiovanni-Vincentelli [120] present a theoretical framework for evaluating the performance of the simulated annealing algorithm. They discuss annealing schedules in terms of initial temperature  $T = t_0$ , the number of inner loops for each value of  $t_k$ , the decrease rate of the temperature (i.e., cooling schedule), and the criteria for stopping the algorithm. They conclude that the theoretical results obtained thus far have not been able to explain why simulated annealing is so successful even when a diverse collection of static cooling schedule heuristics is used. Many heuristic methods are available in the literature to find optimal cooling schedules, but the effectiveness of these schedules can only be compared through experimentation. They conjecture that the neighborhood and the corresponding topology of the objective function are responsible for the behavior of the algorithm.

Cohn and Fielding [34] conduct a detailed analysis of various cooling schedules and how they affect the performance of simulated annealing. Convergent simulated

annealing algorithms are often too slow in practice, whereas a number of non-convergent algorithms may be preferred for good finite-time performance. They analyze various cooling schedules and present cases where repeated independent runs using a non-convergent cooling schedule provide acceptable results in practice. They provide examples of when it is both practically and theoretically justified to use a very high, fixed temperature, or even fast cooling schedules which have a small probability of reaching global minima and apply these cooling schedules to traveling salesman problems of various sizes. Fielding [53] computationally studies fixed temperature cooling schedules for the traveling salesman problem, the quadratic assignment problem and the graph partitioning problem and demonstrates that a fixed temperature cooling schedule can yield superior results in locating optimal and near-optimal solutions. Orosz and Jacobson [115, 116] present finite-time performance measures for simulated annealing with fixed temperature cooling schedules. They illustrate their measures using randomly generated instances of the traveling salesman problem.

Another approach to increasing the speed of simulated annealing is to implement a two-staged simulated annealing algorithm. In two-staged simulated annealing algorithms, a fast heuristic is used to replace simulated annealing at higher temperatures, with a traditional simulated annealing algorithm implemented at lower temperatures to improve on the fast heuristic solution. In addition to implementing an intelligent cooling schedule, finding the initial temperature  $t_0$  to initialize the traditional simulated annealing algorithm is important to the success of the two-staged algorithm. Varanelli and Cohoon [157] propose a method for determining an initial temperature  $t_0$  for two-staged simulated annealing algorithms using traditional cooling schedules. They note that if  $t_0$  is too low at the beginning of the traditional simulated annealing phase, the algorithm can get trapped in an inferior solution, while if the initial temperature  $t_0$  is too high, the algorithm can waste too many iterations (hence computing time) by accepting too many hill-climbing moves.

Azizi and Zolfaghari [11] propose two variations of simulated annealing, tested on the minimum makespan job shop scheduling problems. In the conventional simulated annealing, the temperature declines monotonically, providing the search with a higher transition probability in the beginning of the search and lower probability toward the end of the search. In [11], an adaptive temperature control scheme with a tabu list is used that changes temperature based on the number of consecutive improving moves, resulting in an improved algorithm performance.

In [69], a sample adaptive simulated annealing algorithm, inspired by the idea of Metropolis algorithm, is constructed on a finite state space. The algorithm can be viewed as a substitute of the annealing of iterative stochastic schemes. In [108], the optimal cooling schedule for simulated annealing is formulated to derive a differential equation for the time-dependent temperature  $T(t)$ . Based on this equation, the long-term behavior of  $T(t)$ , entropy production, and the Kullback–Leibler entropy are studied. For some simple examples, such as a many-level system and the small scale traveling salesman problem, the explicit time dependence of the temperature is obtained.

### 1.4.3 Domains—Types of Problems with Examples

Over the past decade, simulated annealing has developed into a popular optimization tool. It has been used to address numerous discrete optimization problems as well as continuous variable problems. Several application articles and surveys have been published on simulated annealing. Johnson et al. [86, 87] present a series of articles on simulated annealing applied to certain well-studied discrete optimization problems. The first in the series of articles uses the graph partitioning problem to illustrate simulated annealing and highlight the effectiveness of several modifications to the basic simulated annealing algorithm. The second in the series focuses on applying lessons learned from the first article to the graph coloring and number partitioning problems. Local optimization techniques were previously thought to be unacceptable approaches to these two problems. In [87], it is also observed that for long run lengths, simulated annealing outperforms the traditional techniques used to solve graph coloring problems. However, simulated annealing did not compare well with traditional techniques on the number partitioning problem except for small problem instances. The third article in the series (not yet published) uses simulated annealing to approach the well-known traveling salesman problem.

Koulamas et al. [90] focus on simulated annealing applied to applications in production/operations management and operations research. They discuss traditional problems such as single machine, flow shop and job shop scheduling, lot sizing, and traveling salesman problems as well as non-traditional problems to include graph coloring and number partitioning. They conclude that simulated annealing is an effective tool for solving many problems in operations research and that the degree of accuracy that the algorithm achieves can be controlled by the practitioner, in terms of number of iterations and neighborhood functions (i.e., an increased number of iterations (outer loops) combined with increased number of searches at each iteration (inner loops) can result in solutions with a higher probability of converging to the optimal solution). In [55], Fleischer discusses simulated annealing from a historical and evolutionary point of view in terms of solving difficult optimization problems. Fleischer also summarizes ongoing research and presents an application of simulated annealing to graph problems.

Steinhofel et al. [134] also address flow shop scheduling problems by applying logarithmic cooling schedules of simulated annealing-based algorithms to flow shop scheduling. In the considered problem setting, the objective is to minimize the overall completion time (called the makespan). A lower bound is derived for the number of steps that are needed to approach an optimum solution with a certain probability, based on the maximum escape depth  $\Gamma$  from local minima of the underlying energy landscape.

The simulated annealing algorithm has proved to be a good technique for solving difficult discrete optimization problems. In engineering optimization, simulated annealing has emerged as an alternative tool to address problems that are difficult to solve by conventional mathematical programming techniques. The algorithm's major disadvantage is that solving a complex system problem may be an extremely

slow, albeit convergent process, using much more processor time than conventional algorithms. Consequently, simulated annealing has not been widely embraced as an optimization algorithm for engineering problems. Attempts have been made to improve the performance of the algorithm either by reducing the annealing length or changing the generation and the acceptance mechanisms. However, these faster schemes, in general, do not inherit the property of escaping local minima. A more efficient way to reduce the processor time and make simulated annealing a more attractive alternative for engineering problems is to add parallelism (see [73]). However, the implementation and efficiency of parallel simulated annealing algorithms are typically problem dependent. Leite et al. [91] consider the evaluation of parallel schemes for engineering problems where the solution spaces may be very complex and highly constrained, with function evaluations varying from medium to high cost. In addition, they provide guidelines for selecting appropriate schemes for engineering problems. They also present an engineering problem with relatively low fitness evaluation cost and strong time constraints to demonstrate the lower bounds of applicability of parallel schemes.

Many signal processing applications create optimization problems with multi-modal and non-smooth cost functions. Gradient methods are ineffective in these situations because of multiple local minima and the requirement to compute gradients. Chen and Luk [27] propose an adaptive simulated annealing algorithm as a viable optimization tool for addressing such difficult non-linear optimization problems. The adaptive simulated annealing algorithm maintains the advantages of simulated annealing, but converges faster. Chen and Luk demonstrate the effectiveness of adaptive simulated annealing with three signal processing applications: maximum likelihood joint channel and data estimation, infinite-impulse-response filter design, and evaluation of minimum symbol-error-rate decision feedback equalizer. They conclude that the adaptive simulated annealing algorithm is a powerful global optimization tool for solving such signal processing problems.

Abramson et al. [4] describe the use of simulated annealing for solving the school timetabling problem. They use the scheduling problem to highlight the performance of six different cooling schedules: the basic geometric cooling schedule, a scheme that uses multiple cooling rates, geometric reheating, enhanced geometric reheating, non-monotonic cooling, and reheating as a function of cost. The basic geometric cooling schedule found in [156] is used as the baseline schedule for comparison purposes. Experimental results suggest that using multiple cooling rates for a given problem yields better quality solutions in less time than the solutions produced by a single cooling schedule. The conclusion in [4] is that the cooling scheme that uses the phase transition temperature (i.e., when sub-parts of the combinatorial optimization problem are solved) in combination with the best solution to date produces the best results.

Emden-Weinert and Proksch [47] present a study of a simulated annealing algorithm for the flight pairing subproblem in crew scheduling, which models the matching of flight segments as a preliminary phase to crew rostering. It is revealed that the algorithm run-time can be decreased and solution quality can be improved by using a problem-specific initial solution, relaxing constraints, combining simulated

annealing with a problem-specific local improvement heuristic, and conducting multiple independent runs.

There is no question that simulated annealing can demand significant computational time to reach global minima. Recent attempts to use parallel computing schemes to speed up simulated annealing have provided promising results. Chu et al. [32] present a new, efficient, and highly general-purpose parallel optimization method based on simulated annealing that does not depend on the structure of the optimization problem being addressed. Their algorithm is used to analyze a network of interacting genes that control embryonic development and other fundamental biological processes. They use a two-stage procedure which monitors and pools performance statistics obtained simultaneously from all processors and then mixes states at intervals to maintain a Boltzmann-like distribution of costs. They demonstrate that their parallel simulated annealing approach leads to nearly optimal parallel efficiency for certain optimization problems. In particular, the approach is appropriate when the relative effort required to compute the cost function is large compared to the relative communication effort between parallel machines for pooling statistics and mixing states.

Chen et al. [26] implement five variants of simulated annealing algorithm from sequential-to-parallel forms on high-performance computers and applied them to a set of standard function optimization problems in order to test their performances. The experimental results indicate that the traditional approach to parallelizing simulated annealing, namely executing algorithm runs simultaneously on multiple communicating processors, does not enjoy much success in solving hard problem instances. Divide-and-conquer decomposition strategy used to traverse the search space sometimes might find the global optimum function value, but frequently results in high computing times as the problem size increases. A hybrid version of a genetic algorithm combined with simulated annealing has proven to be most efficient.

Alrefaei and Andradottir [7] present a modified simulated annealing algorithm with a constant temperature to address discrete optimization problems and use two approaches to estimate an optimal solution to the problem. One approach estimates the optimal solution based on the state most visited versus the state last visited, while the other approach uses the best average estimated objective function value to estimate the optimal solution. Both approaches are guaranteed to converge almost surely to the set of global optimal solutions under mild conditions. They compare the performance of the modified simulated annealing algorithm to other forms of simulated annealing used to solve discrete optimization problems.

Creating effective neighborhood functions or neighborhood generation mechanisms is a critical element in designing efficient and effective simulated annealing algorithms for discrete optimization problems. Tian et al. [147] investigate the application of simulated annealing to discrete optimization problems with a permutation property, such as the traveling salesman problem, the flow shop scheduling problem, and the quadratic assignment problems. They focus on the neighborhood function of the discrete optimization problem and, in particular, the generation mechanism for the algorithm used to address the problem. They introduce six types of perturbation

schemes for generating random permutation solutions and prove that each scheme satisfies asymptotic convergence requirements. The results of the experimental evaluations on the traveling salesman problem, the flow shop scheduling problem, and the quadratic assignment problem suggest that the efficiencies of the perturbation schemes are different for each problem type and solution space. Tian et al. conclude that with the proper perturbation scheme, simulated annealing can produce efficient solutions to different discrete optimization problems that possess a permutation property.

In [5], Ahmed proposes a modification of the simulated annealing algorithm for solving discrete stochastic optimization problems where the objective function is stochastic and can be evaluated only through Monte Carlo simulation (i.e., the objective function cannot be computed exactly or such an evaluation is computationally expensive). In this modification, the temperature is held constant, and the Metropolis criterion depends on whether the objective function values indicate a statistically significant difference at each iteration. The obtained algorithm compares favorably with three previously proposed methods (see [6, 63, 71]).

Research also continues on the application of simulated annealing to optimization of continuous functions. Continuous global optimization is defined as the problem of finding points on a bounded subset of  $\Re^n$  where some real-valued function  $f$  assumes its optimal (maximal or minimal) value. The application of simulated annealing to continuous optimization generally falls into two classes. The first approach closely follows the original idea presented by Kirkpatrick et al. [89], where the algorithm mimics the physical annealing process. The second approach describes the annealing process with Langevin equations, where the global minimum is found by solving a set of stochastic differential equations (see [9]). Gemen and Hwang [64] prove that continuous optimization algorithms based on Langevin equations converge to the global optima. Dekkers and Aarts [40] propose a third stochastic approach to address global optimization based on simulated annealing, which is similar to the formulation of simulated annealing applied to discrete optimization problems. They extend the mathematical formulation of simulated annealing to continuous optimization problems and prove asymptotic convergence to the set of global optima based on the equilibrium distribution of Markov chains. They also discuss an implementation of the proposed algorithm and compare its performance with other well-known algorithms on a standard set of test functions from the literature.

Tsallis and Stariolo [149] discuss and illustrate a new stochastic algorithm (generalized simulated annealing) for computationally finding the global minimum of a given (not necessarily convex) energy/cost function defined on a continuous multi-dimensional space. This algorithm covers, as particular cases, the so-called classical (Boltzmann machine) and fast (Cauchy machine) simulated annealings and turns out to be quicker than both. This method, which has been widely used in many fields as a global optimization tool, is composed of three parts: visiting distribution, accepting rule, and cooling schedule. The most complicated of these is the visiting distribution. Although Tsallis and Stariolo [149] did provide a heuristic algorithm to generate a random number for the visiting distribution, empirical simulations

have shown that it is inappropriate. Deng et al. [43] propose an alternative method of generating random numbers based on the results from [99]. Nishimori and Inoue [109] prove the weak ergodicity of the inhomogeneous Markov process generated by the generalized transition probability of Tsallis and Stariolo [149] under power-law decay of the temperature. Del Moral and Miclo [42] study the convergence of the generalized simulated annealing with time-inhomogeneous communication cost functions. This study is based on the use of log-Sobolev inequalities and semigroup techniques in the spirit of a previous article by one of the authors.

## 1.5 Summary

Simulated annealing optimization algorithms have been well studied in the literature. These algorithms can be used to optimize single as well as multi-objective optimization problems. They have been applied in various fields like process system engineering, operations research, and smart materials. Recent work on simulated annealing primarily involves ad hoc techniques, adaptive cooling schedules, and the development of hybrid algorithms.

The popularity and flexibility of simulated annealing has spawned several new annealing algorithms. Pepper et al. [117] introduce demon algorithms and test them on the traveling salesman problem. Ohlmann et al. [114] introduce another variant of simulated annealing termed compressed annealing. They incorporate the concepts of pressure and volume, in addition to temperature, to address discrete optimization problems with relaxed constraints. They also introduce a primal/dual metaheuristic by simultaneously adjusting temperature and pressure in the algorithm.

In [76], Herault presents rescaled simulated annealing, which is designed for combinatorial problems where the available computational effort is limited. This generalization performs the rescaling of the energies of the states that are candidates for a transition, before applying the Metropolis criterion. A direct consequence of this rescaling is an acceleration of the algorithm's convergence, by avoiding dives and escaping from high-energy local minima. Mingjun and Huanwen [103] propose chaos simulated annealing with chaotic initialization and chaotic sequences replacing the Gaussian distribution. These features improve the rate of convergence and are efficient and easy to implement.

Once a very popular approach to solving hard combinatorial problems, simulated annealing is now taking a backseat and giving way to new algorithms and heuristics designed to better exploit the unique properties and features of problems. However, simulated annealing continues to be widely used, given its simplicity and ease of implementation. Moreover, its simple structure is often incorporated and blended with other metaheuristics. It also remains one of the most analyzed metaheuristics, which underlines the importance and usefulness of its basic idea.

**Acknowledgments** This work is supported in part by the Air Force Office of Scientific Research (FA9550-07-1-0232). The authors wish to thank the anonymous referees for their feedback on this chapter.

## References

1. Aarts, E.H.L., Korst, J.: *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*. Wiley, Chichester (1989)
2. Aarts, E.H.L., Lenstra, J.K.: *Local Search in Combinatorial Optimization*. Wiley, Chichester (1997)
3. Aarts, E.H.L., van Laarhoven, P.J.M.: Statistical cooling: A general approach to combinatorial optimization problems. *Phillips J. Res.* **40**, 193–226 (1985)
4. Abramson, D., Krishnamoorthy, M., Dang, H.: Simulated annealing cooling schedules for the school timetabling problem. *Asia-Pac. J. Oper. Res.* **16**, 1–22 (1999)
5. Ahmed, M.A.: A modification of the simulated annealing algorithm for discrete stochastic optimization. *Eng. Optim.* **39**(6), 701–714 (2007)
6. Alkhamis, T.M., Ahmed, M.A., Tuan, V.K.: Simulated annealing for discrete optimization with estimation. *Eur. J. Oper. Res.* **116**, 530–544 (1999)
7. Alrefaei, M.H., Andradottir, S.: A simulated annealing algorithm with constant temperature for discrete stochastic optimization. *Manage. Sci.* **45**, 748–764 (1999)
8. Althofer, I., Koschnick, K.U.: On the convergence of threshold accepting. *Appl. Math. Optim.* **24**, 183–195 (1991)
9. Aluffi-Pentini, F., Parisi, V., Zirilli, F.: Global optimization and stochastic differential equations. *J. Optim. Theory Appl.* **47**, 1–16 (1985)
10. Anily, S., Federgruen, A.: Simulated annealing methods with general acceptance probabilities. *J. Appl. Probab.* **24**, 657–667 (1987)
11. Azizi, N., Zolfaghari, S.: Adaptive temperature control for simulated annealing: A comparative study. *Comput. Oper. Res.* **31**(14), 2439–2451 (2004)
12. Belisle, C.J.P.: Convergence theorems for a class of simulated annealing algorithms on RD. *J. Appl. Probab.* **29**, 885–895 (1992)
13. Belisle, C.J.P., Romeijn, H.E., Smith, R.L.: Hit-and-run algorithms for generating multivariate distributions. *Math. Oper. Res.* **18**, 255–266 (1993)
14. Ben-Ameur, W.: Computing the initial temperature of simulated annealing. *Comput. Optim. Appl.* **29**, 369–385 (2004)
15. Bohachevsky, I.O., Johnson, M.E., Stein, M.L.: Generalized simulated annealing for function optimization. *Technometrics* **28**, 209–217 (1986)
16. Borkar, V.S.: Pathwise recurrence orders and simulated annealing. *J. Appl. Probab.* **29**, 472–476 (1992)
17. Bouffard, V., Ferland, J.: Improving simulated annealing with variable neighborhood search to solve resource-constrained scheduling problem. *J. Sched.* **10**, 375–386 (2007)
18. Bratley, P., Fox, B.L., Schrage, L.: *A guide to simulation*. Springer, New York, NY (1987)
19. Cardoso, M.F., Salcedo, R.L., de Azevedo, S.F.: Nonequilibrium simulated annealing: A faster approach to combinatorial minimization. *Ind. Eng. Chem. Res.* **33**, 1908–1918 (1994)
20. Catoni, O.: Metropolis, simulated annealing, and Iterated energy transformation algorithms: Theory and experiments. *J. Complex.* **12**, 595–623 (1996)
21. Cerf, R.: Asymptotic convergence of genetic algorithms. *Adv. Appl. Probab.* **30**, 521–550 (1998)
22. Chardaire, P., Lutton, J.L., Sutter, A.: Thermostatistical persistency: A powerful improving concept for simulated annealing algorithms. *Eur. J. Oper. Res.* **86**, 565–579 (1995)
23. Charon, I., Hudry, O.: The noising method - a new method for combinatorial optimization. *Oper. Res. Lett.* **14**, 133–137 (1993)
24. Charon, I., Hudry, O.: The Noising Methods - a generalization of some metaheuristics. *Eur. J. Oper. Res.* **135**, 86–101 (2001)
25. Cheh, K.M., Goldberg, J.B., Askin, R.G.: A note on the effect of neighborhood-structure in simulated annealing. *Comput. Oper. Res.* **18**, 537–547 (1991)
26. Chen, D., Lee, C., Park, C., Mendes, P.: Parallelizing simulated annealing algorithms based on high-performance computer. *J. Global Optim.* **39**, 261–289 (2007)

27. Chen, S., Luk, B.L.: Adaptive simulated annealing for optimization in signal processing applications. *Signal Process.* **79**, 117–128 (1999)
28. Chiang, T.S., Chow, Y.S.: On the convergence rate of annealing processes. *SIAM J. Control Optim.* **26**, 1455–1470 (1988)
29. Chiang, T.S., Chow, Y.Y.: A limit-theorem for a class of inhomogeneous markov-processes. *Ann. Probab.* **17**, 1483–1502 (1989)
30. Chiang, T.S., Chow, Y.Y.: The asymptotic-behavior of simulated annealing processes with absorption. *SIAM J. Control. Optim.* **32**, 1247–1265 (1994)
31. Christoph, M., Hoffmann, K.H.: Scaling behavior of optimal simulated annealing schedules. *J. Phys. A - Math. Gen.* **26**, 3267–3277 (1993)
32. Chu, K.W., Deng, Y.F., Reinitz, J.: Parallel simulated annealing by mixing of states. *J. Comput. Phys.* **148**, 646–662 (1999)
33. Cinlar, E.: Introduction to Stochastic Processes. Prentice-Hall, Englewood Cliffs, NJ (1974)
34. Cohn, H., Fielding, M.: Simulated annealing: Searching for an optimal temperature schedule. *SIAM J. Optim.* **9**, 779–802 (1999)
35. Connors, D.P., Kumar, P.R.: Simulated annealing type markov-chains and their order balance-equations. *SIAM J. Control. Optim.* **27**, 1440–1461 (1989)
36. Czyzak, P., Hapke, M., Jaszkiewicz, A.: Application of the Pareto-Simulated Annealing to the Multiple Criteria Shortest Path Problem, Technical Report, Politechnika Poznanska Instytut Informatyki, Poland (1994)
37. Czyzak, P., Jaszkiewicz, A.: Pareto simulated annealing a metaheuristic technique for multiple-objective combinatorial optimization. *J. Multicriteria Decis. Anal.* **7**, 34–47 (1998)
38. Davis, T.E.: Toward an extrapolation of the simulated annealing convergence theory onto the simple genetic algorithm. Doctoral Dissertation, University of Florida, Gainesville, FL (1991)
39. Davis, T.E., Principe, J.C.: A simulated annealing like convergence theory for the simple genetic algorithm. Proceedings of the Fourth International Conference on Genetic Algorithms in San Diego, CA, pp. 174–181. Morgan Kaufmann, San Francisco, CA (1991)
40. Dekkers, A., Aarts, E.: Global Optimization and Simulated Annealing, *Math. Program.* **50**, 367–393 (1991)
41. Delport, V.: Parallel simulated annealing and evolutionary selection for combinatorial optimisation. *Electron. Lett.* **34**, 758–759 (1998)
42. Del Moral, P., Miclo, L.: On the convergence and applications of generalized simulated annealing. *SIAM J. Control. Optim.* **37**(4), 1222–1250 (1999)
43. Deng, J., Chen, H., Chang, C., Yang, Z.: A superior random number generator for visiting distribution in GSA. *Int. J. Comput. Math.* **81**(1), 103–120 (2004)
44. Desai, M.P.: Some results characterizing the finite time behaviour of the simulated annealing algorithm. *Sadhana-Acad. Proc. Eng. Sci.* **24**, 317–337 (1999)
45. Dueck, G., Scheuer, T.: Threshold accepting - a general-purpose optimization algorithm appearing superior to simulated annealing. *J. Comput. Phys.* **90**, 161–175 (1990)
46. Eglese, R.W.: Simulated annealing: A tool for operational research. *Eur. J. Oper. Res.* **46**, 271–281 (1990)
47. Emden-Weinert, T., Proksch, M.: Best practice simulated annealing for the airline crew scheduling problem. *J. Heuristics* **5**, 419–436 (1999)
48. Fabian, V.: Simulated annealing simulated. *Comput. Math. Appl.* **33**, 81–94 (1997)
49. Faigle, U., Kern, W.: Note on the convergence of simulated annealing algorithms. *SIAM J. Control. Optim.* **29**, 153–159 (1991)
50. Faigle, U., Kern, W.: Some convergence results for probabilistic tabu search. *ORSA J. Comput.* **4**, 32–37 (1992)
51. Faigle, U., Schrader, R.: On the convergence of stationary distributions in simulated annealing algorithms. *Inf. Process. Lett.* **27**, 189–194 (1988)
52. Faigle, U., Schrader, R.: Simulated annealing - a case-study. *Angew. Inform.* **30**(6), 259–263 (1988)

53. Fielding, M.: Simulated annealing with an optimal fixed temperature. *SIAM J. Optim.* **11**, 289–307 (2000)
54. Fleischer, M.A.: Assessing the performance of the simulated annealing algorithm using information theory. Doctoral Dissertation, Department of Operations Research, Case Western Reserve University, Cleveland, Ohio (1993)
55. Fleischer, M.A.: Simulated annealing: Past, present, and future. In: Alexopoulos, C., Kang, K., Lilegdon, W.R., Goldsman, D., (eds.) *Proceedings of the 1995 Winter Simulation Conference*, pp. 155–161. IEEE Press, Arlington, Virginia (1995)
56. Fleischer, M.A.: Generalized cybernetic optimization: Solving continuous variable problems, In: Voss, S., Martello, S., Roucairol, C., Ibrahim, H., Osman, I.H., (eds.) *Meta-heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pp. 403–418. Kluwer (1999)
57. Fleischer, M.A., Jacobson, S.H.: Cybernetic optimization by simulated annealing: An implementation of parallel processing using probabilistic feedback control, In: Osman, I.H., Kelly, J.P., (eds.) *Meta-heuristics: Theory and applications*, pp. 249–264. Kluwer (1996)
58. Fleischer, M.A., Jacobson, S.H.: Information theory and the finite-time behavior of the simulated annealing algorithm: Experimental results. *INFORMS J. Comput.* **11**, 35–43 (1999)
59. Fox, B.L.: Integrating and accelerating tabu search, simulated annealing, and genetic algorithms. *Ann. Oper. Res.* **41**, 47–67 (1993)
60. Fox, B.L.: Random restarting versus simulated annealing. *Comput. Math. Appl.* **27**, 33–35 (1994)
61. Fox, B.L.: Faster Simulated Annealing. *SIAM J. Optim.* **5**, 485–505 (1995)
62. Fox, B.L., Heine, G.W.: Simulated Annealing with Overrides, Technical, Department of Mathematics, University of Colorado, Denver, Colorado (1993)
63. Gelfand, S.B., Mitter, S.K.: Simulated annealing with noisy or imprecise energy measurements. *J. Optim. Theory Appl.* **62**, 49–62 (1989)
64. Geman, S., Hwang, C.R.: Diffusions for global optimization. *SIAM J. Control. Optim.* **24**, 1031–1043 (1986)
65. Gidas, B.: Nonstationary Markov Chains and Convergence of the Annealing Algorithm, *J. Stat. Phys.* **39**, 73–131 (1985)
66. Glover, F.: Tabu search for nonlinear and parametric optimization (with Links to Genetic Algorithms). *Discrete Appl. Math.* **49**, 231–255 (1994)
67. Glover, F., Hanafi, S.: Tabu search and finite convergence. *Discrete Appl. Math.* **119**(1–2), 3–36 (2002)
68. Goldstein, L., Waterman, M.: Neighborhood size in the simulated annealing algorithm. *Am. J. Math. Manage. Sci.* **8**, 409–423 (1988)
69. Gong, G., Liu, Y., Quin, M.: An adaptive simulated annealing algorithm. *Stoch. Processes. Appl.* **94**, 95–103 (2001)
70. Granville, V., Krivanek, M., Rasson, J.P.: Simulated annealing - a proof of convergence. *IEEE Trans. Pattern Anal. Mach. Intell.* **16**, 652–656 (1994)
71. Gutjahr, W.J., Pflug, G.C.: Simulated annealing for noisy cost functions. *J. Global Optim.* **8**, 1–13 (1996)
72. Hajek, B.: Cooling schedules for optimal annealing. *Math. Oper. Res.* **13**, 311–329 (1988)
73. Hamma, B., Viitanen, S., Torn, A.: Parallel continuous simulated annealing for global optimization. *Optim. Methods. Softw.* **13**, 95–116 (2000)
74. Hammersley, J.M., Handscomb, D.C.: Monte Carlo Methods, Methuen, Wiley, London, New York (1964)
75. Henderson, D., Jacobson, S.H., Johnson, A.W.: Handbook of Metaheuristics, Kluwer, Boston, MA (2003)
76. Herault, L.: Rescaled simulated annealing - accelerating convergence of simulated annealing by rescaling the state energies. *J. Heuristics*, **6**, 215–252 (2000)
77. Hu, T.C., Kahing, A.B., Tsao, C.W.A.: Old bachelor acceptance: A new class of non-monotone threshold accepting methods. *ORSA J. Comput.* **7**, 417–425 (1995)

78. Isaacson, D.L., Madsen, R.W.: *Markov Chains, Theory and Applications*, Wiley, New York (1976)
79. Jacobson, S.H.: Analyzing the performance of local search algorithms using generalized hill climbing algorithms. In: Hansen, P., Ribeiro C.C. (eds.) Chapter 20 in *Essays and Surveys on Metaheuristics*, pp. 441–467. Kluwer, Norwell, MA (2002)
80. Jacobson, S.H., Sullivan, K.A., Johnson, A.W.: Discrete manufacturing process design optimization using computer simulation and generalized hill climbing algorithms. *Eng. Optim.* **31**, 247–260 (1998)
81. Jacobson, S.H., Yucesan, E.: Global optimization performance measures for generalized hill climbing algorithms. *J. Global Optim.* **29**(2), 173–190 (2004)
82. Jacobson, S.H., Yucesan, E.: Analyzing the performance of generalized hill climbing algorithms. *J. Heuristics* **10**(4), 387–405 (2004)
83. Jacobson, S.H., Hall, S.N., McLay, L.A., Orosz, J.E.: Performance analysis of cyclical simulated annealing algorithms. *Methodol. Comput. Appl. Probab.* **7**, 183–201 (2005)
84. Johnson, A.W., Jacobson, S.H.: A class of convergent generalized hill climbing algorithms. *Appl. Math. Comput.* **125**(2–3), 359–373 (2002a)
85. Johnson, A.W., Jacobson, S.H.: On the convergence of generalized hill climbing algorithms. *Discrete Appl. Math.*, **119**(1–2), 37–57 (2002b)
86. Johnson, D.S., Aragon, C.R., McGeoch, L.A., Schevon, C.: Optimization by simulated annealing - an experimental evaluation; Part 1, graph partitioning. *Oper. Res.*, **37**, 865–892 (1989)
87. Johnson, D.S., Aragon, C.R., McGeoch, L.A., Schevon, C.: Optimization by simulated annealing - an experimental evaluation; Part 2, graph-coloring and number partitioning. *Oper. Res.*, **39**, 378–406 (1991)
88. Kiatsupaibul, S., Smith, R.L.: A General Purpose Simulated Annealing Algorithm for Integer Linear Programming, Technical Report, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, Michigan (2000)
89. Kirkpatrick, S., Gelatt, Jr., C.D., Vecchi, M.P.: Optimization by simulated annealing. *Science*, **220**, 671–680 (1983)
90. Koulamas, C., Antony, S.R., Jaen, R.: A survey of simulated annealing applications to operations- research problems. *OMEGA-Int. J. Manage. Sci.* **22**, 41–56 (1994)
91. Leite, J.P.B., Topping, B.H.V.: Parallel simulated annealing for structural optimization. *Comput. Struct.* **73**, 545–564 (1999)
92. Liepins, G.E., Hilliard, M.R.: Genetic algorithms: Foundations and applications. *Ann. Oper. Res.* **21**, 31–58 (1989)
93. Lin, C.K.Y., Haley, K.B., Sparks, C.: A comparative study of both standard and adaptive versions of threshold accepting and simulated annealing algorithms in three scheduling problems. *Eur. J. Oper. Res.* **83**, 330–346 (1995)
94. Locatelli, M.: Convergence properties of simulated annealing for continuous global optimization. *J. Appl. Probab.* **33**, 1127–1140 (1996)
95. Locatelli, M.: Simulated annealing algorithms for continuous global optimization: Convergence conditions. *J. Optim. Theory. Appl.* **104**, 121–133 (2000)
96. Locatelli, M.: Convergence and first hitting time of simulated annealing algorithms for continuous global optimization. *Math. Methods. Oper. Res.* **54**, 171–199 (2001)
97. Lundy, M., Mees, A.: Convergence of an annealing algorithm. *Math. Program.* **34**, 111–124 (1986)
98. Ma, J., Straub, J.E.: Simulated annealing using the classical density distribution. *J. Chem. Phys.* **101**, 533–541 (1994)
99. Mantegna, R.N.: Fast, accurate algorithm for numerical simulation of Levy stable stochastic processes. *Phys. Rev. E*, **49**(5), 4677–4683 (1994)
100. Mazza, C.: Parallel simulated annealing, *Random Struct. Algorithms*, **3**, 139–148 (1992)
101. Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., Teller, E.: Equation of state calculations by fast computing machines. *J. Chem. Phys.*, **21**, 1087–1092 (1953)

102. Meyer, C.D.: The condition of a finite markov chain and perturbation bounds for the limiting probabilities. *SIAM J. Algebraic. Discrete Methods* **1**, 273–283 (1980)
103. Mingjun, J., Huanwen, T.: Application of chaos in simulated annealing. *Chaos, Solitons. Fractals* **21**, 933–941 (2003)
104. Mitra, D., Romeo, F., Sangiovanni-Vincentelli, A.L.: Convergence and finite time behavior of simulated annealing. *Adv. Appl. Probab.* **18**, 747–771 (1986)
105. Moscato, P.: An introduction to population approaches for optimization and hierarchical objective functions: A discussion on the role of tabu search. *Ann. Oper. Res.* **41**, 85–121 (1993)
106. Moscato, P., Fontanari, J.F.: Convergence and finite-time behavior of simulated annealing. *Adv. Appl. Probab.* **18**, 747–771 (1990)
107. Muhlenbein, H.: Genetic algorithms, In: Aarts, E., Lenstra, J.K., (eds.) *Local search in combinatorial optimization*, pp. 137–172. Wiley, New York, NY (1997)
108. Munakata, T., Nakamura, Y.: Temperature control for simulated annealing. *Phys. Rev. E Stat. Nonlin. and Soft Matter Phys.* **64**(4II), 461271–461275 (2001)
109. Nishimori, H., Inoue, J.: Convergence of simulated annealing using the generalized transition probability. *J. Phys. A*, **31**, 5661–5672 (1998)
110. Nissen, V., Paul, H.: A modification of threshold accepting and its application to the quadratic assignment problem. *OR Spektrum* **17**, 205–210 (1995)
111. Nolte, A., Schrader R.: A note on finite time behavior of simulated annealing. *Math. Oper. Res.* **25**(3), 476–484 (2000)
112. Nourani, Y., Andresen, B.: A comparison of simulated annealing cooling strategies. *J. Phys. A-Math. Gen.* **31**, 8373–8385 (1998)
113. Ogbu, F.A., Smith, D.K.: The application of the simulated annealing algorithm to the solution of the N/M/Cmax flowshop problem. *Comput. Oper. Res.* **17**, 243–253 (1990)
114. Ohlmann, J.W., Bean, J.C., Henderson, S.G.: Convergence in probability of compressed annealing. *Math. Oper. Res.* **29**(4), 837–860 (2004)
115. Orosz, J.E., Jacobson, S.H.: Finite-time performance analysis of static simulated annealing algorithms. *Comput. Optim. Appl.* **21**, 21–53 (2002a)
116. Orosz, J.E., Jacobson, S.H.: Analysis of static simulated annealing algorithms. *J. Optim. Theory. Appl.* **115**(1), 165–182 (2002b)
117. Pepper, J.W., Golden, B.L., Wasil, E.A.: Solving the traveling salesman problem with annealing-based Heuristics: A computational study. *IEEE Trans. Syst. Manufacturing and Cybernetics, Part A: Syst. Humans*, **32**(1), 72–77 (2002)
118. Rajasekaran, S.: On simulated annealing and nested annealing. *J. Global Optim.* **16**, 43–56 (2000)
119. Romeijn, H.E., Zabinsky, Z.B., Graesser, D.L., Noegi, S.: New reflection generator for simulated annealing in mixed-integer/continuous global optimization. *J. Optim. Theory. Appl.* **101**, 403–427 (1999)
120. Romeo, F., Sangiovanni-Vincentelli, A.: A theoretical framework for simulated annealing. *Algorithmica* **6**, 302–345 (1991)
121. Rosenthal, J.S.: Convergence rates for markov chains. *SIAM Rev.* **37**, 387–405 (1995)
122. Ross, S.M.: *Stochastic processes*, J Wiley, New York, NY (1996)
123. Rossier, Y., Troyon, M., Liebling, T.M.: Probabilistic exchange algorithms and euclidean traveling salesman problems. *OR Spektrum* **8**, 151–164 (1986)
124. Rudolph, G.: Convergence analysis of cononical genetic algorithms. *IEEE Trans. Neural Net. Special Issue on Evolutional Computing*, **5**, 96–101 (1994)
125. Scheermesser, T., Bryngdahl, O.: Threshold accepting for constrained half-toning. *Opt. Commun.* **115**, 13–18 (1995)
126. Schuur, P.C.: Classification of acceptance criteria for the simulated annealing algorithm. *Math. Oper. Res.* **22**, 266–275 (1997)
127. Seneta, E.: *Non-Negative Matrices and Markov Chains*, Springer, New York, NY (1981)

128. Serafini, P.: Mathematics of Multiobjective Optimization, p. 289. CISM Courses and Lectures, Springer, Berlin (1985)
129. Serafini, P.: Simulated Annealing for Multiple Objective Optimization Problems, Proceedings of the Tenth International Conference on Multiple Criteria Decision Making, pp. 87–96, Taipei (1992)
130. Serafini, P.: Simulated Annealing for Multiple Objective Optimization Problems, Multiple Criteria Decision Making. Expand and Enrich the Domains of Thinking and Application pp. 283–292, Springer, Berlin, (1994)
131. Siarry, P., Berthiau, G., Durbin, F., Haussy, J.: Enhanced simulated annealing for globally minimizing functions of many-continuous variables. *ACM Trans. Math. Softw.* **23**, 209–228 (1997)
132. Solla, S.A., Sorkin, G.B., White, S.R.: Configuration space analysis for optimization problems. In: Bienenstock, E., Fogelmanouli, F., Weisbuch, G. (eds.) *Disordered Systems and Biological Organization*, pp. 283–292. Springer, New York (1986)
133. Srichander, R.: Efficient schedules for simulated annealing. *Eng. Optim.* **24**, 161–176 (1995)
134. Steinhofel, K., Albrecht, A., Wong, C.K.: The convergence of stochastic algorithms solving flow shop scheduling. *Theor. Comput. Sci.* **285**, 101–117 (2002)
135. Stern, J.M.: Simulated annealing with a temperature dependent penalty function, *ORSA J. Comput.* **4**, 311–319 (1992)
136. Storer, R.H., Wu, S.D., Vaccari, R.: New Search Spaces for Sequencing Problems with Application to Job Shop Scheduling, *Manage. Sci.* **38**, 1495–1509 (1992)
137. Straub, J.E., Ma, J., Amara, P.: Simulated annealing using coarse grained classical dynamics: Smouuchowski Dynamics in the Gaussian Density Approximation. *J. Chem. Phys.* **103**, 1574–1581 (1995)
138. Streński, P.N., Kirkpatrick, S.: Analysis of finite length annealing schedules. *Algorithmica*, **6**, 346–366 (1991)
139. Sullivan, K.A., Jacobson, S.H.: Ordinal hill climbing algorithms for discrete manufacturing process design optimization problems. *Discrete Event Dyn. Syst.* **10**, 307–324 (2000)
140. Sullivan, K.A., Jacobson, S.H.: A convergence analysis of generalized hill climbing algorithms. *IEEE Trans. Automatic Control* **46**, 1288–1293 (2001)
141. Suman, B.: Multiobjective simulated annealing a metaheuristic technique for multiobjective optimization of a constrained problem. *Found. Comput. Decis. Sci.*, **27**, 171–191 (2002)
142. Suman, B.: Simulated annealing based multiobjective algorithm and their application for system reliability. *Eng. Optim.*, **35**, 391–416 (2003)
143. Suman, B.: Self-stopping PDMOSA and performance measure in simulated annealing based multiobjective optimization algorithms. *Comput. Chem. Eng.* **29**, 1131–1147 (2005)
144. Suman, B., Kumar, P.: A survey of simulated annealing as a tool for single and multiobjective optimization. *J. Oper. Res. Soc.* **57**, 1143–1160 (2006)
145. Suprapitnarm, A., Parks, T.: Simulated Annealing: An Alternative Approach to True Multiobjective Optimization, Genetic and Evolutionary Computation Conference, Conference Workshop Program pp. 406–407, Orlando, FL (1999)
146. Tekinalp, O., Karsli, G.: A new multiobjective simulated annealing algorithm. *J. Global Optim.* **39**, 49–77 (2007)
147. Tian, P., Ma, J., Zhang, D.M.: Application of the simulated annealing algorithm to the combinatorial optimisation problem with permutation property: An investigation of generation mechanism. *Eur. J. Oper. Res.* **118**, 81–94 (1999)
148. Tovey, C.A.: Simulated simulated annealing. *Am. J. Math. Manage. Sci.*, **8**, 389–407 (1988)
149. Tsallis, C., Stariolo, D.A.: Generalized simulated annealing. *Physica A*, **233**, 395–406 (1996)
150. Tsitsiklis, J.N.: Markov chains with rare transitions and simulated annealing. *Math. Oper. Res.* **14**, 70–90 (1989)
151. Triki, E., Collette, Y., Siarry, P.: A theoretical study on the behavior of simulated annealing leading to a new cooling schedule. *Eur. J. Oper. Res.* **166**, 77–92 (2005)

152. Ulungu, L.E., Teghem, J.: Multiobjective combinatorial optimization problems: A survey. *J. Multicriteria Decis. Anal.* **3**, 83–104 (1994)
153. Ulungu, L.E., Teghem, J., Ost, C.: Interactive simulated annealing in a multiobjective framework: Application to an industrial problem. *J. Oper. Res. Soc.* **49**, 1044–1050 (1998)
154. Ulungu, L.E., Teghem, J., Fortemps, P.H., Tuyttens, D.: MOSA method: A tool for solving multiobjective combinatorial optimization problems. *J. Multicriteria Decis. Anal.* **8**, 221–236 (1999)
155. van Laarhoven, P.J.M.: Theoretical and Computational Aspects of Simulated Annealing, Centrum voor Wiskunde en Informatica, Amsterdam, Netherlands (1988)
156. van Laarhoven, P.J.M., Aarts, E.H.L.: Simulated annealing: Theory and applications, D. Reidel: Kluwer, Dordrecht, Boston, Norwell, MA (1987)
157. Varanelli, J.M., Cohoon, J.P.: A fast method for generalized starting temperature determination in homogeneous two-stage simulated annealing systems. *Comput. Oper. Res.* **26**, 481–503 (1999)
158. Vaughan, D., Jacobson, S.H.: Tabu guided generalized hill climbing algorithms. *Methodol. Comput. Appl. Probab.* **6**, 343–354 (2004)
159. Villalobos-Arias, M., Coello, C.A.C., Hernandez-Lerma, O.: Foundations of genetic algorithms. *Lecture Notes in Comput. Sci.* 3469, 95–111 (2005)
160. Villalobos-Arias, M., Coello, C.A.C., Hernandez-Lerma, O.: Asymptotic Convergence of a Simulated Annealing Algorithm for Multiobjective Optimization Problems, *Math. Methods. Oper. Res.*, **64**, 353–362 (2006)
161. Wood, G.R., Alexander, D.L.J., Bulger, D.W.: *J. Global Optim.* **22**, 271–284 (2002)
162. Yan, D., Mukai, H.: Stochastic discrete optimization. *SIAM J. Control Optim.*, **30**, 594–612 (1992)
163. Yang, R.L.: Convergence of the simulated annealing algorithm for continuous global optimization. *J. Optim. Theory Appl.* **104**, 691–716 (2000)
164. Yao, X.: A new simulated annealing algorithm. *Int. J. Comput. Math.* **56**, 161–168 (1995)
165. Yao, X., Li, G.: General simulated annealing. *J. Comput. Sci. Tech.* **6**, 329–338 (1991)
166. Zabinsky, Z.B., Smith, R.L., McDonald, J.F., Romeijn, H.E., Kaufman, D.E.: Improving hit-and-run for global optimization. *J. Global Optim.* **3**, 171–192 (1993)
167. Zolfaghari, S., Liang, M.: Comparative study of simulated annealing, genetic algorithms and tabu Search for solving binary and comprehensive machine-grouping problems. *Int. J. Prod. Resour.* **40**(9), 2141–2158 (2002)



# Chapter 2

## Tabu Search

Michel Gendreau and Jean-Yves Potvin

**Abstract** This chapter presents the fundamental concepts of tabu search (TS) in a tutorial fashion. Special emphasis is put on showing the relationships with classical local search methods and on the basic elements of any TS heuristic, namely the definition of the search space, the neighborhood structure, and the search memory. Other sections cover other important concepts such as search intensification and diversification and provide references to significant work on TS. Recent advances in TS are also briefly discussed.

### 2.1 Introduction

Over the last 20 years, hundreds of papers presenting applications of tabu search (TS), a heuristic method originally proposed by Glover in 1986 [29], to various combinatorial problems have appeared in the operations research literature. In several cases, the methods described provide solutions very close to optimality and are among the most effective, if not the best, to tackle the difficult problems at hand. These successes have made TS extremely popular among those interested in finding good solutions to the large combinatorial problems encountered in many practical settings. Several papers, book chapters, special issues, and books have surveyed

---

Michel Gendreau

Département de mathématiques et de génie industriel, École Polytechnique de Montréal, and Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport, Montréal, QC, Canada  
email: michel.gendreau@cirrelt.ca

Jean-Yves Potvin

Département d'informatique et de recherche opérationnelle, Université de Montréal, and Centre interuniversitaire de recherche sur les réseaux d'entreprise, la logistique et le transport, Montréal, QC, Canada  
email: potvin@iro.umontreal.ca

the rich TS literature (a list of some of the most important references is provided in a later section). In spite of this abundant literature, there still seem to be many researchers who, while they are eager to apply TS to new problem settings, find it difficult to properly grasp the fundamental concepts of the method, its strengths and its limitations, and to come up with effective implementations. The purpose of this chapter is to address this situation by providing an introduction in the form of a tutorial focusing on the fundamental concepts of TS. Throughout the chapter, a relatively straightforward, yet challenging and relevant, problem will be used to illustrate these concepts: the classical vehicle routing problem (CVRP). This problem will be introduced in the following section. The remainder of the chapter is organized as follows. The basic concepts of TS, like the search space, neighborhood structure, and short-term tabu lists, are described and illustrated in Section 2.3. Intermediate, yet critical, concepts, such as intensification and diversification, are described in Section 2.4. This is followed in Section 2.5 by a brief discussion of advanced topics and recent trends in TS, and in Section 2.6 by a short list of key references on TS and its applications. Section 2.7 provides practical tips for newcomers struggling with unforeseen problems as they first try to apply TS to their favorite problem. Section 2.8 concludes the chapter with some general advice on the application of TS to combinatorial problems.

## 2.2 The Classical Vehicle Routing Problem

Vehicle routing problems have very important applications in the area of distribution management. As a consequence, they have become some of the most studied problems in the combinatorial optimization literature and a large number of papers and books (see [62], for example) deal with the numerous procedures that have been proposed to solve them. These include several TS implementations that currently rank among the most effective. The CVRP is the basic variant in that class of problems. It can formally be defined as follows. Let  $G = (V, A)$  be a graph where  $V$  is the vertex set and  $A$  is the arc set. One of the vertices represents the depot at which a fleet of  $m$  identical vehicles of capacity  $Q$  is based, and the other vertices represent customers that need to be serviced. With each customer vertex  $v_i$  are associated a demand  $q_i$  and a service time  $t_i$ . With each arc  $(v_i, v_j)$  of  $A$  are associated a cost  $c_{ij}$  and a travel time  $t_{ij}$ . The CVRP consists in finding a set of routes such that

- Each route begins and ends at the depot;
- Each customer is visited exactly once by exactly one route;
- The total demand of the customers assigned to each route does not exceed  $Q$ ;
- The total duration of each route (including travel and service times) does not exceed a specified value  $L$ ;
- The total cost of the routes is minimized.

A feasible solution for the problem thus consists in a partition of the customers into  $m$  groups, each of total demand no larger than  $Q$ , that are sequenced to yield

routes (starting and ending at the depot) of duration no larger than  $L$ . This problem will be used in the following to illustrate how various TS concepts can be applied in practice.

## 2.3 Basic Concepts

Before introducing the basic concepts of TS, the next section first goes back in time to try to better understand the genesis of the method and how it relates to previous work.

### 2.3.1 Historical Background

Heuristics, i.e., approximate solution techniques, have been used since the beginnings of operations research to tackle difficult combinatorial problems. With the development of complexity theory in the early 1970s, it became clear that, since most of these problems were NP-hard, there was little hope of ever finding efficient exact solution procedures for them. This realization emphasized the role of heuristics for solving the combinatorial problems that were encountered in real-life applications and that needed to be tackled, whether or not they were NP-hard. While many different approaches were proposed and experimented with, the most popular one was based on local search (LS) improvement techniques. LS can be roughly summarized as an iterative search procedure that, starting from an initial feasible solution, progressively improves it by applying a series of local modifications (or moves). At each iteration, the search moves to an improving feasible solution that differs only slightly from the current one (in fact, the difference between the previous and the new solutions amounts to one of the local modifications mentioned above). The search terminates when it encounters a local optimum with respect to the transformations that it considers, an important limitation of the method: unless one is extremely lucky, this local optimum is often a fairly mediocre solution. In LS, the quality of the solution obtained and computing times are usually highly dependent on the richness of the set of transformations (moves) considered at each iteration of the heuristic.

In 1983, the world of combinatorial optimization was shattered by the appearance of a paper [43] where it was shown that a new heuristic approach called simulated annealing (SA) could converge to an optimal solution of a combinatorial problem, albeit in infinite computing time. Based on an analogy with statistical mechanics, SA can be interpreted as a form of controlled random walk in the space of feasible solutions. The emergence of SA indicated that one could look for other ways to tackle combinatorial optimization problems and spurred the interest of the research community. In the following years, many other new approaches were proposed, mostly based on analogies with natural phenomena (like TS, ant colony optimization,

particle swarm optimization, artificial immune systems) which, together with some older ones, such as genetic algorithms [40], gained an increasing popularity. Now collectively known under the name of metaheuristics (a term originally coined by Glover in [29]), these methods have become over the last 20 years the leading edge of heuristic approaches for solving combinatorial optimization problems.

### 2.3.2 Tabu Search

Building upon some of his previous work, Fred Glover proposed a new approach, which he called tabu search, to allow local search methods to overcome local optima [29]. In fact, many elements of this first TS proposal and some elements of later TS elaborations were introduced in [28], including short-term memory to prevent the reversal of recent moves, and longer term frequency memory to reinforce attractive components. The basic principle of TS is to pursue LS whenever it encounters a local optimum by allowing non-improving moves; cycling back to previously visited solutions is prevented by the use of memories, called tabu lists, that record the recent history of the search, a key idea that can be linked to artificial intelligence concepts. It is also important to remark that Glover did not see TS as a proper heuristic, but rather as a metaheuristic, i.e., a general strategy for guiding and controlling inner heuristics specifically tailored to the problems at hand.

### 2.3.3 Search Space and Neighborhood Structure

As we just mentioned, TS is an extension of classical LS methods. In fact, a basic TS can be seen as simply the combination of LS with short-term memories. It follows that the two first basic elements of any TS heuristic are the definition of its search space and its neighborhood structure.

The search space of an LS or TS heuristic is simply the space of all possible solutions that can be considered (visited) during the search. For instance, in the CVRP example described in Section 2.2, the search space could simply be the set of feasible solutions to the problem, where each point in the search space corresponds to a set of vehicles routes satisfying all the specified constraints. While in that case the definition of the search space seems quite natural, it is not always so. In the capacitated plant location problem (CPLP), for instance, customers must be served from plants located in a subset of potential sites. In this context, one could use the full feasible search space made of binary location variables (a site is open or closed) and continuous flow variables. A more attractive search space, though, is obtained by restricting the search space to the binary location variables, from which the complete solution can be obtained by solving the associated transportation problem to get the optimal flow variables. One could also decide to search for the extreme points of the

set of feasible flow variable vectors, retrieving the associated location variables by noting that a plant must be open whenever some flow is allocated to it [13]. It is also important to note that it is not always a good idea to restrict the search space to feasible solutions; in many cases, allowing the search to move to infeasible solutions is desirable and sometimes necessary (see Section 2.4.3 for further details).

Closely linked to the definition of the search space is that of the neighborhood structure. At each iteration of LS or TS, the local transformations that can be applied to the current solution, denoted  $S$ , define a set of neighboring solutions in the search space, denoted  $N(S)$  (the neighborhood of  $S$ ). Formally,  $N(S)$  is a subset of the search space made of all solutions obtained by applying a single local transformation to  $S$ . In general, for any specific problem at hand, there are many more possible (and even, attractive) neighborhood structures than search space definitions. This follows from the fact that there may be several plausible neighborhood structures for a given definition of the search space. This is easily illustrated on our CVRP example that has been the object of several TS implementations. To simplify the discussion, we suppose in the following that the search space is the feasible space. Simple neighborhood structures for the CVRP involve moving at each iteration a single customer from its current route; the selected customer is inserted in the same route or in another route with sufficient residual capacity. An important feature of these neighborhood structures is the way in which insertions are performed: one could use random insertion or insertion at the best position in the target route; alternately, one could use more complex insertion schemes that involve a partial re-optimization of the target route, such as GENI insertions [24]. Before proceeding any further it is important to stress that while we say that these neighborhood structures involve moving a single customer, the neighborhoods they define contain all the feasible route configurations that can be obtained from the current solution by moving any customer and inserting it in the stated fashion. Examining the neighborhood can thus be fairly demanding.

More complex neighborhood structures for the CVRP, such as the  $\lambda$ -interchange [47], are obtained by allowing simultaneously the movement of customers to different routes and the swapping of customers between routes. In [50], moves are defined by ejection chains that are sequences of coordinated movements of customers from one route to another; for instance, an ejection chain of length 3 would involve moving a customer  $v_1$  from route  $R_1$  to route  $R_2$ , a customer  $v_2$  from  $R_2$  to route  $R_3$ , and a customer  $v_3$  from  $R_3$  to route  $R_4$ . Other neighborhood structures involve the swapping of sequences of several customers between routes, as in the cross-exchange [60]. These types of neighborhoods have seldom been used for the CVRP, but are common in TS heuristics for its time-windows extension, where customers must be visited within a pre-specified time interval. We refer the interested reader to [6, 25] for a more detailed discussion of TS implementations for the CVRP and the vehicle routing problem with time windows.

When different definitions of the search space are considered for a given problem, neighborhood structures will inevitably differ to a considerable degree. In the case of the CPLP, alluded to above, if the search space corresponds to the location variables only, one could use operators to change the status of these variables

(from open to closed and conversely). If, however, the search space is made of the extreme points of the set of feasible flow variable vectors, one could instead consider moves defined by the application of pivots to the linear programming formulation of the transportation problem to move the current solution to an adjacent extreme point. Thus, choosing a search space and a neighborhood structure is by far the most critical step in the design of any TS heuristic. It is at this step that one must make the best use of the understanding and knowledge he/she has of the problem at hand.

### 2.3.4 Tabus

Tabus are one of the distinctive elements of TS when compared to LS. As we already mentioned, tabus are used to prevent cycling when moving away from local optima through non-improving moves. The key realization here is that when this situation occurs, something needs to be done to prevent the search from tracing back its steps to where it came from. This is achieved by declaring tabu (disallowing) moves that reverse the effect of recent moves. For instance, in the CVRP example, if customer  $v_1$  has just been moved from route  $R_1$  to route  $R_2$ , one could declare tabu moving back  $v_1$  from  $R_2$  to  $R_1$  for some number of iterations (this number is called the tabu tenure of the move). Tabus are also useful to help the search move away from previously visited portions of the search space and thus perform more extensive exploration.

Tabus are stored in a short-term memory of the search (the tabu list) and usually only a fixed and fairly limited quantity of information is recorded. In any given context, there are several possibilities regarding the specific information that is recorded. One could record complete solutions, but this requires a lot of storage and makes it expensive to check whether a potential move is tabu or not; it is therefore seldom used. The most commonly used tabus involve recording the last few transformations performed on the current solution and prohibiting reverse transformations (as in the example above); others are based on key characteristics of the solutions themselves or of the moves.

To better understand how tabus work, let us go back to our reference problem. In the CVRP, one could define tabus in several ways. To continue our example where customer  $v_1$  has just been moved from route  $R_1$  to route  $R_2$ , one could declare tabu specifically moving back  $v_1$  from  $R_2$  to  $R_1$  and record this in the short-term memory as the triplet  $(v_1, R_2, R_1)$ . Note that this type of tabu will not constrain the search much and that cycling may occur if  $v_1$  is then moved to another route  $R_3$  and then from  $R_3$  to  $R_1$ . A stronger tabu would involve prohibiting moving back  $v_1$  to  $R_1$ , without consideration for its current route and be recorded as  $(v_1, R_1)$ . An even stronger tabu would be to disallow moving  $v_1$  to any other route and would simply be noted as  $(v_1)$ .

Multiple tabu lists can be used simultaneously and are sometimes advisable. For example, when different types of moves are used to generate the neighborhood,

it might be a good idea to keep a separate tabu list for each type. Standard tabu lists are usually implemented as circular lists of fixed length. It has been shown, however, that fixed-length tabus cannot always prevent cycling, and some authors have proposed varying the tabu list length during the search [30, 31, 55, 58, 59]. Another solution is to randomly generate the tabu tenure of each move within some specified interval; using this approach requires a somewhat different scheme for recording tabus that are then usually stored as tags in an array (the entries in this array will usually record the iteration number until which a move is tabu; see [24], for more details).

### 2.3.5 Aspiration Criteria

While central to TS, tabus are sometimes too powerful: they may prohibit attractive moves, even when there is no danger of cycling, or they may lead to an overall stagnation of the searching process. It is thus necessary to use algorithmic devices that will allow one to revoke (cancel) tabus. These are called aspiration criteria. The simplest and most commonly used aspiration criterion, which is found in almost all TS implementations, consists in allowing a move, even if it is tabu, if it results in a solution with an objective value better than that of the current best-known solution (since the new solution has obviously not been previously visited). Much more complicated aspiration criteria have been proposed and successfully implemented (see, for instance [39, 65]), but they are rarely used. The key rule in this respect is that if cycling cannot occur, tabus can be disregarded.

### 2.3.6 A Template for Simple Tabu Search

We are now in the position to give a general template for TS, integrating the elements we have seen so far. We suppose that we are trying to minimize a function  $f(S)$  over some domain and we apply the so-called best improvement version of TS, i.e., the version in which one chooses at each iteration the best available move (this is the most commonly used version of TS).

#### *Notation*

- $S$ , the current solution,
- $S^*$ , the best-known solution,
- $f^*$ , the value of  $S^*$ ,
- $N(S)$ , the neighborhood of  $S$ ,
- $\tilde{N}(S)$ , the admissible subset of  $N(S)$  (i.e., non-tabu or allowed by aspiration),
- $T$ , the tabu list.

### *Initialization*

Choose (construct) an initial solution  $S_0$ .  
Set  $S \leftarrow S_0$ ,  $f^* \leftarrow f(S_0)$ ,  $S^* \leftarrow S_0$ ,  $T \leftarrow \emptyset$ .

### *Search*

While termination criterion not satisfied do

select  $S$  in  $\operatorname{argmin}_{S' \in \tilde{N}(S)} [f(S')]$ ;  
if  $f(S) < f^*$ , then set  $f^* \leftarrow f(S)$ ,  $S^* \leftarrow S$ ;  
record tabu for the current move in  $T$  (delete oldest entry if necessary).

### **2.3.7 Termination Criteria**

One may have noticed that we have not specified in our template above a termination criterion. In theory, the search could go on forever, unless the optimal value of the problem at hand is known beforehand. In practice, obviously, the search has to be stopped at some point. The most commonly used stopping criteria in TS are as follows:

- after a fixed number of iterations (or a fixed amount of CPU time);
- after some number of iterations without an improvement in the objective function value (the criterion used in most implementations);
- when the objective reaches a pre-specified threshold value.

In complex tabu schemes, the search is usually stopped after completing a sequence of phases, the duration of each phase being determined by one of the above criteria.

### **2.3.8 Probabilistic TS and Candidate Lists**

In regular TS, one must evaluate the objective for every element of the neighborhood  $N(S)$  of the current solution. This can prove extremely expensive from the computational standpoint. An alternative is to instead consider only a random sample  $N'(S)$  of  $N(S)$ , thus reducing significantly the computational burden. Another attractive feature of this alternative is that the added randomness can act as an anti-cycling mechanism; this allows one to use shorter tabu lists than would be necessary if a full exploration of the neighborhood was performed. One the negative side, it must be noted that, in that case, one may miss excellent solutions (more on this topic in Section 2.7.3). Probabilities may also be applied to activating tabu criteria.

Another way to control the number of moves examined is by means of candidate list strategies, which provide more strategic ways of generating a useful subset  $N'(S)$  of  $N(S)$  (the probabilistic approach can be considered to be one instance of a candidate list strategy and may also be used to modify such a strategy). Failure to adequately address the issues involved in creating effective candidate lists is one of the more conspicuous shortcomings that differentiates a naive TS implementation from one that is more solidly grounded. Relevant designs for candidate list strategies are discussed in [34]. We also discuss a useful type of candidate generation approach in Section 2.4.4. Another interesting approach for the CVRP is the granular TS [63], where only arcs that are likely to be found in good solutions (i.e., short ones) are considered, thus reducing the size of the underlying graph.

## 2.4 Intermediate Concepts

Simple TS as described above can sometimes successfully solve difficult problems, but in most cases, additional elements have to be included in the search strategy to make it fully effective. We now briefly review the most important of these.

### 2.4.1 Intensification

The idea behind the concept of search intensification is that, as an intelligent human being would probably do, one should explore more thoroughly the portions of the search space that seem promising to make sure that the best solutions in these areas are indeed found. From time to time, one would thus stop the normal searching process to perform an intensification phase. In general, intensification is based on some intermediate-term memory, such as a recency memory, in which one records the number of consecutive iterations that various solution components have been present in the current solution without interruption. For instance, in a CVRP application, one could record how long an arc has been used. A typical approach to intensification is to restart the search from the best currently known solution and to fix the components that seem more attractive. To continue the CVRP example, one could fix the arcs that have been used for the largest number of iterations and perform a restricted search on the remaining arcs. Another technique that is often used consists in changing the neighborhood structure to one allowing more powerful or more diverse moves. In the CVRP example, one could therefore allow more complex insertion moves or switch to an ejection chain neighborhood structure [32]. In probabilistic TS, one could increase the sample size or switch to searching without sampling.

Intensification is used in many TS implementations, but it is not always necessary. This is because there are many situations where the search performed by the normal process is thorough enough. There is thus no need to spend time exploring

more carefully the portions of the search space that have already been visited, and this time can be used more effectively as we shall see right now.

### ***2.4.2 Diversification***

One of the main problems of all methods based on local search approaches, and this includes TS in spite of the beneficial impact of tabus, is that they tend to be too local (as their name implies), i.e., they tend to spend most, if not all, of their time in a restricted portion of the search space. The negative consequence of this fact is that, although good solutions may be obtained, one may fail to explore the most interesting parts of the search space and thus end up with solutions that are still pretty far from the optimal ones. Diversification is an algorithmic mechanism that tries to alleviate this problem by forcing the search into previously unexplored areas of the search space. It is usually based on some form of long-term memory of the search, such as a frequency memory, in which one records the total number of iterations (since the beginning of the search) that various solution components have been present in the current solution or have been involved in the selected moves. For instance, in the CVRP application, one could note how many times each customer has been moved from its current route. In cases where it is possible to identify useful regions of the search space, the frequency memory can be refined to track the number of iterations spent in these different regions.

There are two major diversification techniques. The first, called restart diversification, involves forcing a few rarely used components in the current solution (or the best-known solution) and restarting the search from this point. In a CVRP heuristic, customers that have not yet been moved frequently could be forced into new routes. The second diversification method, continuous diversification, integrates diversification considerations directly into the regular searching process. This is achieved by biasing the evaluation of possible moves by adding to the objective a small term related to component frequencies (see [56] for an extensive discussion on these two techniques). A third way of achieving diversification is strategic oscillation as we will see in the next section.

Before closing this section, we would like to stress that ensuring proper search diversification is possibly the most critical issue in the design of TS heuristics. It should be addressed with extreme care fairly early in the design phase and revisited if the results obtained are not up to expectations.

### ***2.4.3 Allowing Infeasible Solutions***

Accounting for all problem constraints in the definition of the search space often restricts the searching process too much and can lead to mediocre solutions. This occurs, for example, in CVRP instances where the route capacity or

duration constraints are too tight to allow moving customers effectively between routes. In such cases, constraint relaxation is an attractive strategy, since it creates a larger search space that can be explored with simpler neighborhood structures. Constraint relaxation is easily implemented by dropping selected constraints from the search space definition and adding to the objective weighted penalties for constraint violations.

This, however, raises the issue of finding correct weights for constraint violations. An interesting way of circumventing this problem is to use self-adjusting penalties, i.e., weights are adjusted dynamically on the basis of the recent history of the search: weights are increased if only infeasible solutions were encountered in the last few iterations and decreased if all recent solutions were feasible (see, for instance, [24] for further details). Penalty weights can also be modified systematically to drive the search to cross the feasibility boundary of the search space and thus induce diversification. This technique, known as strategic oscillation, was introduced as early as 1977 in [28] and used since in several successful TS procedures (an important early variant oscillates among different types of moves, hence neighborhood structures, while another oscillates around a selected value for a critical function).

#### 2.4.4 Surrogate and Auxiliary Objectives

There are many problems for which the true objective function is quite costly to evaluate. When this occurs, the evaluation of moves may become prohibitive, even if sampling is used. An effective approach to handle this issue is to evaluate neighbors using a surrogate objective, i.e., a function that is correlated to the true objective, but is less computationally demanding, in order to identify a (small) set of promising candidates (potential solutions achieving the best values for the surrogate). The true objective is then computed for this small set of candidate moves and the best one selected to become the new current solution; an example of this approach is found in [15].

Another frequently encountered difficulty is that the objective function may not provide enough information to effectively drive the search to more interesting areas of the search space. A typical illustration of this situation is the variant of the CVRP in which the fleet size is not fixed, but is rather the primary objective (i.e., one is looking for the minimal fleet size allowing a feasible solution). In this problem, except for solutions where a route has only one or a few customers assigned to it, most neighborhood structures will lead to the situation where all elements in the neighborhood score equally with respect to the primary objective (i.e., all allowable moves produce solutions with the same number of vehicles). In such a case, it is absolutely necessary to define an auxiliary objective function to orient the search. Such a function must measure in some way the desirable attributes of solutions. In our example, one could, for instance, use a function that would favor solutions with routes having just a few customers, thus increasing the likelihood that a route can be totally emptied in a subsequent iteration. It should be noted that coming up with an

effective auxiliary objective is not always easy and may require a lengthy trial and error process. In some other cases, fortunately, the auxiliary objective is obvious for anyone familiar with the problem at hand (see [27], for an illustration).

## 2.5 Advanced Concepts and Recent Trends

The concepts and techniques described in the previous sections are sufficient to design effective TS heuristics for many combinatorial problems. Most early TS implementations, several of which were extremely successful, relied indeed almost exclusively on these algorithmic components. Nowadays, however, most leading edge research in TS makes use of more advanced concepts and techniques. While it is clearly beyond the scope of an introductory tutorial, such as this one, to review this type of advanced material, we would like to give readers some insight into it by briefly describing some current trends in TS research. Readers who wish to learn more about this topic should read our survey paper [21] and some of the references provided in the next section.

A large part of the recent research in TS deals with various techniques for making the search more effective. These include methods for exploiting better the information that becomes available during search and creating better starting points, as well as more powerful neighborhood operators and parallel search strategies (on this last topic, see the taxonomy in [16] and the survey in [17]). The numerous techniques for making better use of the information are of particular significance since they can lead to dramatic performance improvements. Many of these rely on elite solutions (the best solutions previously encountered) or on parts of these to create new solutions, the rationale being that fragments or elements of excellent solutions are often identified quite early in the searching process, but that the challenge is to complete these fragments or to recombine them [32–34, 53, 61]. Other methods, such as the reactive TS [4], attempt to find ways of making the search move away from local optima that have already been visited. An important issue is the general approach for exploiting the search framework provided by TS. Some favor simplicity, that is, a search strategy with only a few parameters and based on simple neighborhood operators, as illustrated by the unified TS [10, 11, 20]. Others propose complex neighborhood operators, thus leading to large or very large neighborhood searches [1].

Another important trend in TS (this is, in fact, a pervasive trend in the whole metaheuristics field) is hybridization, i.e., using TS in conjunction with other solution approaches such as genetic algorithms [12, 19], Lagrangean relaxation [37], constraint programming [3, 7, 49], column generation [13], and integer programming techniques (there is a whole chapter on this topic in [34]).

TS research has also started moving away from its traditional application areas (graph theory problems, scheduling, vehicle routing) to new ones: continuous optimization [5, 8, 9, 42, 54], multi-criteria optimization [38, 42], stochastic programming [2, 46], mixed integer programming [13, 47], real-time decision problems

[22, 24], etc. These new areas confront researchers with new challenges that, in turn, call for novel and original extensions of the method.

## 2.6 Key References

Readers who wish to read other introductory papers on TS can choose among several ones [26, 33, 36, 39, 57, 65]. The book by Glover and Laguna [34] is the ultimate reference on TS: apart from the fundamental concepts of the method, it presents a considerable amount of advanced material as well as a variety of applications. It is interesting to note that this book contains several ideas applicable to TS that yet remain to be fully exploited. The issues of *Annals of Operations Research*, respectively devoted to *Tabu Search* [35] and *Metaheuristics in Combinatorial Optimization* [44], are extremely valuable as well as the books made up from selected papers presented at the Metaheuristics International Conferences (MIC) in 1995 [48], 1997 [64], 1999 [52], 2001 [51], 2003 [41], and 2005 [18]. A book for the 2009 conference in Hamburg is also planned. Finally, a special issue of *Journal of Heuristics* was devoted to the MIC conference held in Montreal in 2007 [14].

## 2.7 Tricks of the Trade

Newcomers to TS trying to apply the method to a problem that they wish to solve are often confused about what they need to do to come up with a successful implementation. This section is aimed at providing some help in this regard.

### 2.7.1 Getting Started

The following step-by-step procedure should provide a useful framework for getting started.

A step-by-step procedure is given as follows:

1. Read one or two good introductory papers to gain some knowledge of the concepts and of the vocabulary.
2. Read several papers describing in detail applications in various areas to see how the concepts have been actually implemented by other researchers.
3. Think a lot about the problem at hand, focusing on the definition of the search space and the neighborhood structure.
4. Implement a simple version based on this search space definition and this neighborhood structure.

5. Collect statistics on the performance of this simple heuristic. It is usually useful at this point to introduce a variety of memories, such as frequency and recency memories, to really track down what the heuristic does.
6. Analyze results and adjust the procedure accordingly. It is at this point that one should eventually introduce mechanisms for search intensification and diversification or other intermediate features. Special attention should be paid to diversification, since this is often where simple TS procedures fail.

### **2.7.2 More Tips**

It is not unusual that, in spite of following carefully the preceding procedure, one ends up with a heuristic that nonetheless produces mediocre results. If this occurs, the following tips may prove useful:

1. If there are constraints, consider penalizing them. Letting the search move to infeasible solutions is often necessary in highly constrained problems to allow for a meaningful exploration of the search space (see Section 2.4).
2. Reconsider the neighborhood structure and change it if necessary. Many TS implementations fail because the neighborhood structure is too simple. In particular, one should make sure that the chosen neighborhood structure allows for a purposeful evaluation of possible moves (i.e., the moves that seem intuitively to move the search in the right direction should be the ones that are likely to be selected); it might also be a good idea to introduce a surrogate objective to achieve this (see Section 2.4).
3. Collect more statistics.
4. Follow the execution of the algorithm step by step on some reasonably sized instances.
5. Reconsider diversification. As mentioned earlier, this is a critical feature in most TS implementations.
6. Experiment with parameter settings. Many TS procedures are extremely sensitive to parameter settings; it is not unusual to see the performance of a procedure dramatically improve after changing the value of one or two key parameters (unfortunately, it is not always obvious to determine which parameters are the key ones in a given procedure).

### **2.7.3 Additional Tips for Probabilistic TS**

While it is an effective way of tackling many problems, probabilistic TS creates problems of its own that need to be carefully addressed. The most important of these is the fact that, more often than not, the best solutions returned by probabilistic TS will not be local optima with respect to the neighborhood structure being used. This

is particularly annoying since, in that case, better solutions can be easily obtained, sometimes even manually. An easy way to come around this is to simply perform a local improvement phase (using the same neighborhood operator) from the best found solution at the end of the TS itself. One could alternately switch to TS without sampling (again from the best found solution) for a short duration before completing the algorithm. A possibly more effective technique is to add throughout the search an intensification step without sampling; in this fashion, the best solutions available in the various regions of the search space explored by the method will be found and recorded (similar special aspiration criteria for allowing the search to reach local optima at useful junctures are proposed in [33]).

#### ***2.7.4 Parameter Calibration and Computational Testing***

Parameter calibration and computational experiments are key steps in the development of any algorithm. This is particularly true in the case of TS, since the number of parameters required by most implementations is fairly large and since the performance of a given procedure can vary quite significantly when parameter values are modified. The first step in any serious computational experimentation is to select a good set of benchmark instances (either by obtaining them from other researchers or by constructing them), preferably with some reasonable measure of their difficulty and with a wide range of size and difficulty. This set should be split into two subsets, the first one being used at the algorithmic design and parameter calibration steps, and the second reserved for performing the final computational tests that will be reported in the paper(s) describing the heuristic under development. The reason for doing so is quite simple: when calibrating parameters, one always run the risk of overfitting, i.e., finding parameter values that are excellent for the instances at hand, but poor in general, because these values provide too good a fit (from the algorithmic standpoint) to these instances. Methods with several parameters should thus be calibrated on much larger sets of instances than ones with few parameters to ensure a reasonable degree of robustness. The calibration process itself should proceed in several stages:

1. Perform exploratory testing to find good ranges of parameters. This can be done by running the heuristic with a variety of parameter settings.
2. Fix the value of parameters that appear to be robust, i.e., which do not seem to have a significant impact on the performance of the procedure.
3. Perform systematic testing for the other parameters. It is usually more efficient to test values for only a single parameter at a time, the others being fixed at what appear to be reasonable values. One must be careful, however, for cross effects between parameters. Where such effects exist, it can be important to jointly test pairs or triplets of parameters, which can be an extremely time-consuming task.

The work in [15] provides a detailed description of the calibration process for a fairly complex TS procedure and can be used as a guideline for this purpose.

## 2.8 Conclusion

Tabu search is a powerful algorithmic approach that has been applied with great success to many difficult combinatorial problems. A particularly nice feature of TS is that, like all approaches based on local search, it can quite easily handle complicating constraints that are typically found in real-life applications. It is thus a really practical approach. It is not, however, a panacea: every reviewer or editor of a scientific journal has seen more than his/her share of failed TS heuristics. These failures stem from two major causes: an insufficient understanding of fundamental concepts of the method (and we hope that this tutorial will help in alleviating this shortcoming), but also, more often than not, a crippling lack of understanding of the problem at hand. One cannot develop a good TS heuristic for a problem that he/she does not know well! This is because significant problem knowledge is absolutely required to perform the most basic steps of the development of any TS procedure, namely the choice of a search space and of an effective neighborhood structure. If the search space and/or the neighborhood structure are inadequate, no amount of TS expertise will be sufficient to save the day. A last word of caution: to be successful, all metaheuristics need to achieve both depth and breadth in their searching process; depth is usually not a problem for TS, which is quite aggressive in this respect (TS heuristics generally find pretty good solutions very early in the search), but breadth can be a critical issue. To handle this, it is extremely important to develop an effective diversification scheme.

**Acknowledgments** The authors are grateful to the Canadian Natural Sciences and Engineering Research Council for their financial support. The authors also wish to thank Fred Glover for his insightful comments on an earlier version of this chapter.

## References

1. Ahuja, R.K., Ergun, O., Orlin, J.B., Punnen A.P.: A survey of very large-scale neighborhood search techniques. *Discrete Appl. Math.* **123**, 75–102 (2002)
2. Aringhieri, R.: Solving chance-constrained programs combining tabu search and simulation. *Lect. Notes Comput. Sci.* **3059**, 30–41 (2004)
3. de Backer, B., Furnon, V., Shaw, P., Kilby P., Prosser P.: Solving vehicle routing problems using constraint programming and metaheuristics. *J. Heuristics* **6**, 501–523 (2000)
4. Battiti, R., Tecchiolli, G.: The reactive tabu search. *ORSA J. Comput.* **6**, 126–140 (1994)
5. Battiti, R., Tecchiolli, G.: The continuous reactive tabu search: blending combinatorial optimization and stochastic search for global optimization. *Ann. Oper. Res.* **63**, 151–188 (1996)
6. Bräysy, O., Gendreau, M.: Tabu search heuristics for the vehicle routing problem with time windows. *TOP* **10**, 211–237 (2002)
7. Caseau Y., Laburthe, F., Le Pape, C., Rottembourg, B.: Combining local and global search in a constraint programming environment. *Knowl. Eng. Rev.* **16**, 41–68 (2001)
8. Chelouah, R., Siarry, P.: Tabu Search applied to global optimization. *Eur. J. Oper. Res.* **123**, 256–270 (2000)
9. Chelouah, R., Siarry, P.: A hybrid method combining continuous tabu search and Nelder-Mead simplex algorithms for the global optimization of multimimima functions. *Eur. J. Oper. Res.* **161**, 636–654 (2005)

10. Cordeau, J.-F., Gendreau, M., Laporte, G.: A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks* **30**, 105–119 (1997)
11. Cordeau, J.-F., Laporte, G., Mercier, A.: A unified tabu search heuristic for vehicle routing problems with time windows. *J. Oper. Res. Soc.* **52**, 928–936 (2001)
12. Crainic, T.G., Gendreau, M.: Towards an evolutionary method—Cooperative multi-thread parallel tabu search heuristic hybrid. In: Voss, S., Martello, S., Osman, H.I., Roucairol, C. (eds.) *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pp. 331–344. Kluwer, Boston (1999)
13. Crainic, T.G., Gendreau, M., Farvolden, J.M.: Simplex-based tabu search for the multicommodity capacitated fixed charge network design problem. *INFORMS J. Comput.* **12**, 223–236 (2000)
14. Crainic, T.G., Gendreau, M., Rousseau, L.-M. (eds.): Special issue on Recent advances in metaheuristics. *J. Heuristics* **16**(3), 235–535 (2010)
15. Crainic, T.G., Gendreau, M., Soriano, P., Toulouse, M.: A tabu search procedure for multicommodity location/allocation with balancing requirements. *Ann. Oper. Res.* **41**, 359–383 (1993)
16. Crainic, T.G., Toulouse, M., Gendreau, M.: Toward a taxonomy of parallel tabu search heuristics. *INFORMS J. Comput.* **9**, 61–72 (1997)
17. Cung, V.-D., Martins, S.L., Ribeiro, C.C., Roucairol, C.: Strategies for the parallel implementation of metaheuristics. In: Ribeiro, C.C., Hansen, P. (eds.) *Essays and Surveys in Metaheuristics*, pp. 263–308. Kluwer, Boston (2002)
18. Doerner, K.F., Gendreau, M., Greistorfer, P., Gutjahr, W., Hartl, R.F., Reimann, M.: *Metaheuristics—Progress in Complex Systems Optimization*, Springer, New York, NY (2007)
19. Fleurant, C., Ferland, J.A.: Genetic and hybrid algorithms for graph colouring. *Ann. Oper. Res.* **63**, 437–461 (1996)
20. Fu, Z., Eglese, R., Li, L.Y.O.: A unified tabu search algorithm for vehicle routing problems with soft time windows. *J. Oper. Res. Soc.* **59**, 663–673 (2008)
21. Gendreau, M.: Recent advances in tabu search. In: Ribeiro, C.C., Hansen, P. (eds.) *Essays and Surveys in Metaheuristics*, pp. 369–377. Kluwer, Boston (2002)
22. Gendreau, M., Guertin, F., Potvin, J.-Y., Séguin, R.: Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries. *Transp. Res. Part C: Emerg. Technol.* **14**, 157–174 (2006)
23. Gendreau, M., Guertin, F., Potvin, J.-Y., Taillard, É.D.: Parallel tabu search for real-time vehicle routing and dispatching. *Transp. Sci.* **33**, 381–390 (1999)
24. Gendreau, M., Hertz, A., Laporte, G.: A tabu search heuristic for the vehicle routing problem. *Manage. Sci.* **40**, 1276–1290 (1994)
25. Gendreau, M., Laporte, G., Potvin, J.-Y.: Metaheuristics for the capacitated VRP. In: Toth, P., Vigo, D. (eds.) *The Vehicle Routing Problem*, SIAM Monographs on Discrete Mathematics and Applications, pp. 129–154. SIAM, Philadelphia (2002)
26. Gendreau, M., Potvin, J.-Y.: Tabu search. In: Burke, E.K., Kendall, G. (eds.) *Search Methodologies—Introductory Tutorials in Optimization and Decision Support Techniques*, pp. 165–186. Springer, New York, NY (2005)
27. Gendreau, M., Soriano, P., Salvail, L.: Solving the maximum clique problem using a tabu search approach. *Ann. Oper. Res.* **41**, 385–403 (1993)
28. Glover, F.: Heuristics for integer programming using surrogate constraints. *Decis. Sci.* **8**, 156–166 (1977)
29. Glover, F.: Future paths for integer programming and links to artificial intelligence. *Comput. Oper. Res.* **13**, 533–549 (1986)
30. Glover, F.: Tabu search—Part I. *ORSA J. Comput.* **1**, 190–206 (1989)
31. Glover, F.: Tabu search—Part II. *ORSA J. Comput.* **2**, 4–32 (1990)
32. Glover, F.: Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Appl. Math.* **65**, 223–253 (1996)

33. Glover, F., Laguna, M.: Tabu search. In: Reeves, C.R. (ed.) *Modern Heuristic Techniques for Combinatorial Problems*, pp. 70–150. Blackwell Scientific Publications, Oxford (1993)
34. Glover, F., Laguna, M.: *Tabu Search*. Kluwer, Boston (1997)
35. Glover, F., Laguna, M., Taillard, É.D., de Werra, D. (eds.): *Tabu search*. Ann. Oper. Research **41**, J.C. Baltzer AG Science Publishers, Basel (1993)
36. Glover, F., Taillard, É.D., de Werra, D.: A user's guide to tabu search. Ann. Oper. Research **41**, 3–28 (1993)
37. Grünert, T.: Lagrangean tabu search. In: Ribeiro, C.C., Hansen, P. (eds.) *Essays and Surveys in Metaheuristics*, pp. 379–397. Kluwer, Boston (2002)
38. Hansen, M.P.: Tabu search in multiobjective optimisation: MOTS. In: *Proceedings of the 13th International Conference on Multiple Criteria Decision Making*, pp. 574–586, Cape Town, South Africa (1997)
39. Hertz, A., de Werra, D.: The tabu search metaheuristic: how we used it. Ann. Math. Artif. Intell. **1**, 111–121 (1991)
40. Holland, J.H.: *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor (1975)
41. Ibaraki, T., Nonobe, K., Yagiura, M. (eds.): *Metaheuristics: Progress as Real Problem Solvers*, Springer, New York, NY (2005)
42. Jaeggi, D.M., Parks, G.T., Kipouros, T., Clarkson, P.J.: The development of a multi-objective tabu search algorithm for continuous optimisation problems. Eur. J. Oper. Res. **185**, 1192–1212 (2008)
43. Kirkpatrick, S., Gelatt Jr., C.D., Vecchi, M.P.: Optim. Simulated Annealing. Science **220**, 671–680 (1983)
44. Laporte, G., Osman, I.H. (eds.): *Metaheuristics in combinatorial optimization*. Ann. Oper. Res. **63**, J.C. Baltzer AG Science Publishers, Basel (1996)
45. Løkketangen, A., Glover, F.: Probabilistic move selection in tabu search for 0/1 mixed integer programming problems. In: Osman, I.H., Kelly, J.P. (eds.) *Meta-Heuristics: Theory and Applications*, pp. 467–488. Kluwer, Boston (1996)
46. Løkketangen, A., Woodruff, D.L.: Progressive hedging and tabu search applied to mixed integer (0,1) multistage stochastic programming. J. Heuristics **2**, 111–128 (1996)
47. Osman, I.H.: Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. Ann. Oper. Res. **41**, 421–451 (1993)
48. Osman, I.H., Kelly, J.P. (eds.): *Meta-heuristics: Theory and Applications*. Kluwer, Boston (1996)
49. Pesant, G., Gendreau, M.: A constraint programming framework for local search methods. J. Heuristics **5**, 255–280 (1999)
50. Rego, C., Roucairol, C.: A parallel tabu search algorithm using ejection chains for the vehicle routing problem. In: Osman, I.H., Kelly, J.P. (eds.) *Meta-heuristics: Theory and Applications*, pp. 661–675. Kluwer, Boston (1996)
51. Resende, M.G.C., de Sousa, J.P. (eds.): *Metaheuristics—Computer Decision Making*. Kluwer, Boston (2004)
52. Ribeiro, C.C., Hansen, P. (eds.): *Essays and Surveys in Metaheuristics*. Kluwer, Boston (2002)
53. Rochat, Y., Taillard, É.D.: Probabilistic diversification and intensification in local search for vehicle routing. J. Heuristics **1**, 147–167 (1995)
54. Rolland, E.: A tabu search method for constrained real-number search: Applications to portfolio selection. Technical Report, Department of Accounting and Management Information Systems, Ohio State University, Columbus (1997)
55. Skorin-Kapov, J.: Tabu search applied to the quadratic assignment problem. ORSA J. Comput. **2**, 33–45 (1990)
56. Soriano, P., Gendreau, M.: Diversification strategies in tabu search algorithms for the maximum clique problem. Ann. Oper. Res. **63**, 189–207 (1996)
57. Soriano, P., Gendreau, M.: Fondements et applications des méthodes de recherche avec tabous. RAIRO (Recherche opérationnelle) **31**, 133–159 (1997) (in French)

58. Taillard, É.D.: Some efficient heuristic methods for the flow shop sequencing problem. *Eur. J. Oper. Res.* **47**, 65–74 (1990)
59. Taillard, É.D.: Robust taboo search for the quadratic assignment problem. *Parallel Comput.* **17**, 443–455 (1991)
60. Taillard, É.D., Badeau, P., Gendreau, M., Guertin, F., Potvin, J.-Y.: A tabu search heuristic for the vehicle routing problem with soft time windows. *Transp. Sci.* **31**, 170–186 (1997)
61. Tarantilis, C.D., Kiranoudis, C.T.: BoneRoute—An adaptive memory-based method for effective fleet management. *Ann. Oper. Res.* **115**, 227–241 (2002)
62. Toth, P., Vigo, D. (eds.): The vehicle routing problem. SIAM Monographs on Discrete Mathematics and Applications. SIAM, Philadelphia (2002)
63. Toth, P., Vigo, D.: The granular tabu search and its application to the vehicle routing problem. *INFORMS J. Comput.* **15**, 333–346 (2003)
64. Voss, S., Martello, S., Osman, I.H., Roucairol, C. (eds.): Meta-heuristics: Advances and Trends in Local Search Paradigms for Optimization. Kluwer, Boston (1999)
65. de Werra, D., Hertz, A.: Tabu search techniques: a tutorial and an application to neural networks. *OR Spektrum* **11**, 131–141 (1989)



# Chapter 3

## Variable Neighborhood Search

Pierre Hansen, Nenad Mladenović, Jack Brimberg and José A. Moreno Pérez

**Abstract** Variable neighborhood search (VNS) is a metaheuristic for solving combinatorial and global optimization problems whose basic idea is a systematic change of neighborhood both within a descent phase to find a local optimum and in a perturbation phase to get out of the corresponding valley. In this chapter we present the basic schemes of VNS and some of its extensions. We then describe a recent development, i.e., formulation space search. We then present five families of applications in which VNS has proven to be very successful: (i) exact solution of large-scale location problems by primal–dual VNS; (ii) generation of feasible solutions to large mixed integer linear programs by hybridization of VNS and local branching; (iii) generation of good feasible solutions to continuous nonlinear programs; (iv) generation of feasible solutions and/or improved local optima for mixed integer nonlinear programs by hybridization of sequential quadratic programming and branch and bound within a VNS framework, and (v) exploration of graph theory to find conjectures, refutations, and proofs or ideas of proofs.

---

Pierre Hansen

GERAD and Ecole des Hautes Etudes Commerciales, 3000 ch. de la Côte-Sainte-Catherine,  
Montréal H3T 2A7, QC, Canada

e-mail: pierreh@crt.umontreal.ca

Nenad Mladenović

School of Mathematics, Brunel University-West London, Uxbridge, UB8 3PH, UK  
e-mail: nenad.mladenovic@brunel.ac.uk

Jack Brimberg

Department of Mathematics and Computer Science, Royal Military College of Canada, Kingston  
ON, Canada, K7K 7B4  
e-mail: jack.brimberg@rmc.ca

José A. Moreno Pérez

IUDR and DEIOC, Universidad de La Laguna, 38271 La Laguna, Santa Cruz de Tenerife, Spain  
e-mail: jamoreno@ull.es

### 3.1 Introduction

Optimization tools have greatly improved during the last two decades. This is due to several factors: (i) progress in mathematical programming theory and algorithmic design; (ii) rapid improvement in computer performances; (iii) better communication of new ideas and integration in widely used complex softwares. Consequently, many problems long viewed as out of reach are currently solved, sometimes in very moderate computing times. This success, however, has led researchers and practitioners to address much larger instances and more difficult classes of problems. Many of these may again only be solved heuristically. Therefore thousands of papers describing, evaluating, and comparing new heuristics appear each year. Keeping abreast of such a large literature is a challenge. Metaheuristics, or general frameworks for building heuristics, are therefore needed in order to organize the study of heuristics. As evidenced by this handbook, there are many of them. Some desirable properties of metaheuristics [64, 67, 68] are listed in the concluding section of this chapter.

Variable neighborhood search (VNS) is a metaheuristic proposed by some of the present authors a dozen years ago [85]. Earlier work that motivated this approach can be found in [28, 41, 44, 82]. It is based on the idea of a systematic change of neighborhood both in a descent phase to find a local optimum and in a perturbation phase to get out of the corresponding valley. Originally designed for approximate solution of combinatorial optimization problems, it was extended to address mixed integer programs, nonlinear programs, and recently mixed integer nonlinear programs. In addition VNS has been used as a tool for automated or computer-assisted graph theory. This led to the discovery of over 1500 conjectures in that field, the automated proof of more than half of them as well as the unassisted proof of about 400 of them by many mathematicians.

Applications are rapidly increasing in number and pertain to many fields: location theory, cluster analysis, scheduling, vehicle routing, network design, lot-sizing, artificial intelligence, engineering, pooling problems, biology, phylogeny, reliability, geometry, telecommunication design, etc. (see, e.g., [20, 21, 31, 38, 39, 66, 77, 99]). References are too numerous to be all listed here, but many others can be found in [69] and special issues of *IMA Journal of Management Mathematics* [81], *European Journal of Operational Research* [68], and *Journal of Heuristics* [89] are devoted to VNS.

This chapter is organized as follows. In the next section we present the basic schemes of VNS, i.e., variable neighborhood descent (VND), reduced VNS (RVNS), basic VNS (BVNS), and general VNS (GVNS). Two important extensions are presented in Section 3.3: skewed VNS and variable neighborhood decomposition search (VNDS). A further recent development called formulation space search (FSS) is discussed in Section 3.4. The remainder of this chapter describes applications of VNS to several classes of large scale and complex optimization problems for which it has proven to be particularly successful. Section 3.5 is devoted to primal-dual VNS (PD-VNS) and its application to location and clustering problems. Finding feasible solutions to large mixed integer linear programs with VNS

is discussed in Section 3.6. Section 3.7 addresses ways to apply VNS in continuous global optimization. The more difficult case of solving mixed integer nonlinear programming by VNS is considered in Section 3.8. Applying VNS to graph theory per se (and not just to particular optimization problems defined on graphs) is discussed in Section 3.9. Brief conclusions are drawn in Section 3.10.

## 3.2 Basic Schemes

A deterministic optimization problem may be formulated as

$$\min\{f(x)|x \in X, X \subseteq \mathcal{S}\}, \quad (3.1)$$

where  $\mathcal{S}$ ,  $X$ ,  $x$ , and  $f$  denote the *solution space*, the *feasible set*, a *feasible solution*, and a real-valued *objective function*, respectively. If  $\mathcal{S}$  is a finite but large set, a *combinatorial optimization* problem is defined. If  $\mathcal{S} = \mathbb{R}^n$ , we refer to *continuous optimization*. A solution  $x^* \in X$  is *optimal* if

$$f(x^*) \leq f(x), \forall x \in X.$$

An *exact algorithm* for problem (3.1), if one exists, finds an optimal solution  $x^*$ , together with the proof of its optimality, or shows that there is no feasible solution, i.e.,  $X = \emptyset$ , or the solution is unbounded. Moreover, in practice, the time needed to do so should be finite (and not too long). For continuous optimization, it is reasonable to allow for some degree of tolerance, i.e., to stop when sufficient convergence is detected.

Let us denote  $\mathcal{N}_k$  ( $k = 1, \dots, k_{\max}$ ), a finite set of pre-selected neighborhood structures, and with  $\mathcal{N}_k(x)$  the set of solutions in the  $k$ th neighborhood of  $x$ . Most local search heuristics use only one neighborhood structure, i.e.,  $k_{\max} = 1$ . Often successive neighborhoods  $\mathcal{N}_k$  are nested and may be induced from one or more metric (or quasi-metric) functions introduced into a solution space  $S$ . An *optimal solution*  $x_{\text{opt}}$  (or *global minimum*) is a feasible solution where a minimum is reached. We call  $x' \in X$  a *local minimum* of Equation (3.1) with respect to  $\mathcal{N}_k$  (w.r.t.  $\mathcal{N}_k$  for short), if there is no solution  $x \in \mathcal{N}_k(x') \subseteq X$  such that  $f(x) < f(x')$ . Metaheuristics (based on local search procedures) try to continue the search by other means after finding the first local minimum. VNS is based on three simple facts:

**Fact 1** A local minimum w.r.t. one neighborhood structure is not necessarily so for another;

**Fact 2** A global minimum is a local minimum w.r.t. all possible neighborhood structures;

**Fact 3** For many problems, local minima w.r.t. one or several  $\mathcal{N}_k$  are relatively close to each other.

This last observation, which is empirical, implies that a local optimum often provides some information about the global one. For instance, it might be several variables with the same value in both solutions. However, it is usually not known which variables are such. Since these variables usually cannot be identified in advance, one should conduct an organized study of the neighborhoods of the local optimum until a better solution is found.

In order to solve Equation (3.1) by using several neighborhoods, facts 1 to 3 can be used in three different ways: (i) deterministic, (ii) stochastic, (iii) both deterministic and stochastic.

We first give in Algorithm 1 the solution move and neighborhood change function that will be used later.

---

### Algorithm 1 Neighborhood change

---

```
Function NeighborhoodChange ( $x, x', k$ )
1 if  $f(x') < f(x)$  then
2    $x \leftarrow x'$  // Make a move
3    $k \leftarrow 1$  // Initial neighborhood
  else
4    $k \leftarrow k + 1$  // Next neighborhood
return  $x, k$ 
```

---

Function NeighborhoodChange ( ) compares the incumbent value  $f(x)$  with the new value  $f(x')$  obtained from the  $k$ th neighborhood (line 1). If an improvement is obtained, the new incumbent is updated (line 2) and  $k$  is returned to its initial value (line 3). Otherwise, the next neighborhood is considered (line 4).

(i) The **variable neighborhood descent** (VND) method is obtained if a change of neighborhoods is performed in a deterministic way. It is presented in Algorithm 2, where neighborhoods are denoted as  $N_k, k = 1, \dots, k_{\max}$ .

---

### Algorithm 2 Variable neighborhood descent

---

```
Function VND ( $x, k_{\max}$ )
1  $k \leftarrow 1$ 
2 repeat
3    $x' \leftarrow \arg \min_{y \in N_k(x)} f(y)$  // Find the best neighbor in  $N_k(x)$ 
4    $x, k \leftarrow \text{NeighborhoodChange} (x, x', k)$  // Change neighborhood
  until  $k = k_{\max}$ 
return  $x$ 
```

---

Most local search heuristics use a single or sometimes two neighborhoods for improving the current solution (i.e.,  $k_{\max} \leq 2$ ). Note that the final solution should be a local minimum w.r.t. all  $k_{\max}$  neighborhoods, and thus a global optimum is more likely to be reached than with a single structure. Beside this *sequential* order of neighborhood structures in VND, one can develop a *nested* strategy. Assume,

for example, that  $k_{\max} = 3$ ; then a possible nested strategy is to perform VND with Algorithm 2 for the first two neighborhoods from each point  $x'$  that belongs to the third one ( $x' \in N_3(x)$ ). Such an approach is successfully applied in [22, 27, 65].

(ii) The **reduced VNS** (RVNS) method is obtained if random points are selected from  $\mathcal{N}_k(x)$  and no descent is made. Rather, the values of these new points are compared with that of the incumbent and an update takes place in case of improvement. We also assume that a stopping condition has been chosen like the maximum CPU time allowed  $t_{\max}$  or the maximum number of iterations between two improvements. To simplify the description of the algorithms we always use  $t_{\max}$  below. Therefore, RVNS uses two parameters:  $t_{\max}$  and  $k_{\max}$ . It is presented in Algorithm 3.

---

**Algorithm 3** Reduced VNS

---

```
Function RVNS( $x, k_{\max}, t_{\max}$ )
  1 repeat
  2    $k \leftarrow 1$ 
  3   repeat
  4      $x' \leftarrow \text{Shake}(x, k)$ 
  5      $x, k \leftarrow \text{NeighborhoodChange}(x, x', k)$ 
  6   until  $k = k_{\max}$ 
  7    $t \leftarrow \text{CpuTime}()$ 
  8   until  $t > t_{\max}$ 
  9 return  $x$ 
```

---

The function `Shake` in line 4 generates a point  $x'$  at random from the  $k$ th neighborhood of  $x$ , i.e.,  $x' \in \mathcal{N}_k(x)$ . It is given in Algorithm 4, where it is assumed that points from  $\mathcal{N}_k(x)$  are  $\{x^1, \dots, x^{|\mathcal{N}_k(x)|}\}$ . RVNS is useful for very large instances

---

**Algorithm 4** Shaking function

---

```
Function Shake( $x, k$ )
  1  $w \leftarrow [1 + \text{Rand}(0, 1) \times |\mathcal{N}_k(x)|]$ 
  2  $x' \leftarrow x^w$ 
  3 return  $x'$ 
```

---

for which local search is costly. It can be used as well for finding initial solutions for large problems before decomposition. It has been observed that the best value for the parameter  $k_{\max}$  is often 2 or 3. In addition, a maximum number of iterations between two improvements is usually used as the stopping condition. RVNS is akin to a Monte Carlo method, but is more systematic (see, e.g., [86] where results obtained by RVNS were 30% better than those of the Monte Carlo method in solving a continuous min–max problem). When applied to the  $p$ -Median problem, RVNS gave equally good solutions as the *Fast Interchange* heuristic of [98] while being 20 to 40 times faster [70].

(iii) The **basic VNS** (BVNS) method [85] combines deterministic and stochastic changes of neighborhood. The deterministic part is represented by a local search heuristic. It consists in (i) choosing an initial solution  $x$ , (ii) finding a direction of descent from  $x$  (within a neighborhood  $N(x)$ ), and (iii) moving to the minimum of  $f(x)$  within  $N(x)$  along that direction. If there is no direction of descent, the heuristic stops, otherwise it is iterated. Usually the steepest descent direction, also referred to as *best improvement*, is used. This is summarized in Algorithm 5, where we assume that an initial solution  $x$  is given. The output consists of a local minimum, also denoted by  $x$ , and its value. As the Steepest descent heuristic may be time-

---

**Algorithm 5** Best improvement (steepest descent) heuristic

---

```
Function BestImprovement( $x$ )
1 repeat
2    $x' \leftarrow x$ 
3    $x \leftarrow \arg \min_{y \in N(x)} f(y)$ 
4   until ( $f(x) \geq f(x')$ )
return  $x$ 
```

---

consuming, an alternative is to use the *first descent* heuristic. Vectors  $x^i \in N(x)$  are then enumerated systematically and a move is made as soon as a direction for the descent is found. This is summarized in Algorithm 6.

---

**Algorithm 6** First improvement (first descent) heuristic

---

```
Function FirstImprovement( $x$ )
1 repeat
2    $x' \leftarrow x; i \leftarrow 0$ 
3   repeat
4      $i \leftarrow i + 1$ 
5      $x \leftarrow \arg \min\{f(x), f(x^i)\}, x^i \in N(x)$ 
6     until ( $f(x) < f(x^i)$  or  $i = |N(x)|$ )
7   until ( $f(x) \geq f(x')$ )
return  $x$ 
```

---

The stochastic phase is represented by the random selection of one point from the  $k$ th neighborhood. The BVNS is given in Algorithm 7.

Note that point  $x'$  is generated at random in step 5 in order to avoid cycling, which might occur with a deterministic rule.

**Example.** We illustrate the basic steps on a minimum  $k$ -cardinality tree instance taken from [76], see Figure 3.1. The minimum  $k$ -cardinality tree problem on graph  $G$  ( $k$ -card for short) consists in finding a subtree of  $G$  with exactly  $k$  edges whose sum of weights is minimum.

The steps of BVNS for solving the 4-card problem are illustrated in Figure 3.2. In step 0 the objective function value, i.e., the sum of edge weights, is equal to 40;

**Algorithm 7** Basic VNS

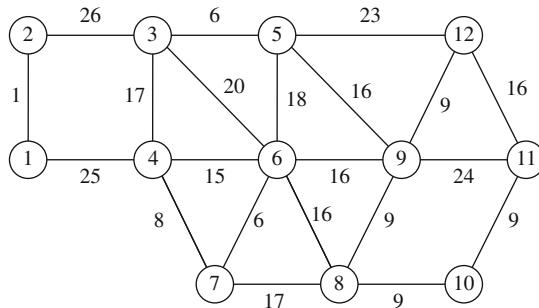
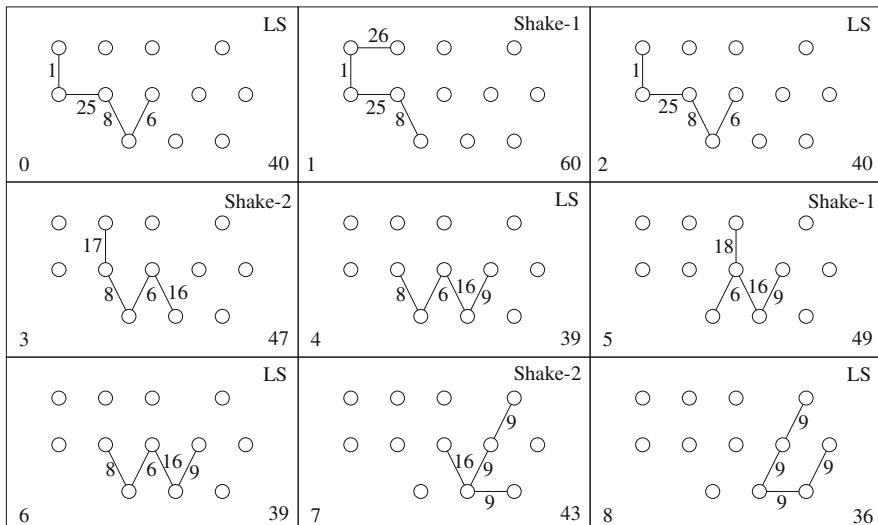
---

```

Function BVNS( $x, k_{max}, t_{max}$ )
  1  $t \leftarrow 0$ 
  2 while  $t < t_{max}$  do
  3    $k \leftarrow 1$ 
  4   repeat
  5      $x' \leftarrow \text{Shake}(x, k)$            // Shaking
  6      $x'' \leftarrow \text{BestImprovement}(x')$  // Local search
  7      $x, k \leftarrow \text{NeighborhoodChange}(x, x'', k)$  // Change neighborhood
  8   until  $k = k_{max}$ 
  9    $t \leftarrow \text{CpuTime}()$ 
return  $x$ 

```

---

**Fig. 3.1** 4-cardinality tree problem.**Fig. 3.2** Steps of the basic VNS for solving 4-card problem.

it is indicated in the right bottom corner of the figure. That first solution is a local minimum with respect to the edge-exchange neighborhood structure (one edge in, one out). After shaking, the objective function is 60, and after another local search, we are back to the same solution. Then, in step 3, we take out 2 edges and add another 2 at random, and after a local search, an improved solution is obtained with a value of 39. Continuing in that way, the optimal solution with an objective function value equal to 36 is obtained in step 8.

(iv) **General VNS.** Note that the local search step (line 6 in BVNS, Algorithm 7) may also be replaced by VND (Algorithm 2). This general VNS (VNS/VND) approach has led to some of the most successful applications reported in the literature (see, e.g., [2, 27, 30, 32, 34, 36, 37, 65, 71, 92, 93]). The general VNS (GVNS) is given in Algorithm 8 below.

---

**Algorithm 8** General VNS

---

```

Function GVNS ( $x, \ell_{max}, k_{max}, t_{max}$ )
  1 repeat
  2    $k \leftarrow 1$ 
  3   repeat
  4      $x' \leftarrow \text{Shake}(x, k)$ 
  5      $x'' \leftarrow \text{VND}(x', \ell_{max})$ 
  6      $x, k \leftarrow \text{NeighborhoodChange}(x, x'', k)$ 
  7   until  $k = k_{max}$ 
     $t \leftarrow \text{CpuTime}()$ 
  until  $t > t_{max}$ 
return  $x$ 

```

---

### 3.3 Some Extensions

(i) The **skewed VNS** (SVNS) method [59] addresses the problem of exploring valleys far from the incumbent solution. Indeed, once the best solution in a large region has been found, it is necessary to go quite far to obtain an improved one. Solutions drawn at random in far-away neighborhoods may differ substantially from the incumbent, and VNS may then degenerate, to some extent, into the multistart heuristic (in which descents are made iteratively from solutions generated at random, which is known not to be very efficient). So some compensation for distance from the incumbent must be made and a scheme called skewed VNS is proposed for that purpose. Its steps are presented in Algorithms 9, 10, and 11. The  $\text{KeepBest}(x, x')$  function (Algorithm 9) in SVNS simply keeps the best of solutions  $x$  and  $x'$ . The  $\text{NeighborhoodChangeS}$  function (Algorithm 10) performs the move and neighborhood change for the SVNS.

SVNS makes use of a function  $\rho(x, x'')$  to measure the distance between the incumbent solution  $x$  and the local optimum  $x''$ . The distance function used to define  $\mathcal{N}_k$ , as in the above examples, could be used also for this purpose. The parameter

---

**Algorithm 9** Keep best solution

---

```
Function KeepBest( $x, x'$ )
  1 if  $f(x') < f(x)$  then
  2    $x \leftarrow x'$ 
return  $x$ 
```

---



---

**Algorithm 10** Neighborhood change for the skewed VNS

---

```
Function NeighborhoodChangeS( $x, x', k, \alpha$ )
  1 if  $f(x') - \alpha\rho(x, x') < f(x)$  then
  2    $x \leftarrow x'$ 
  3    $k \leftarrow 1$ 
    else
  4    $k \leftarrow k + 1$ 
return  $x, k$ 
```

---



---

**Algorithm 11** Skewed VNS

---

```
Function SVNS ( $x, k_{max}, t_{max}, \alpha$ )
  1  $x_{best} \leftarrow x$ 
  2 repeat
  3    $k \leftarrow 1$ 
  4   repeat
  5      $x' \leftarrow \text{Shake}(x, k)$ 
  6      $x'' \leftarrow \text{FirstImprovement}(x')$ 
  7      $x, k \leftarrow \text{NeighborhoodChangeS}(x, x'', k, \alpha)$ 
    until  $k = k_{max}$ 
  8    $x_{best} \leftarrow \text{KeepBest}(x_{best}, x)$ 
  9    $x \leftarrow x_{best}$ 
 10    $t \leftarrow \text{CpuTime}()$ 
    until  $t > t_{max}$ 
return  $x$ 
```

---

$\alpha$  must be chosen to allow movement to valleys far away from  $x$  when  $f(x'')$  is larger than  $f(x)$  but not too much larger (otherwise one will always leave  $x$ ). A good value for  $\alpha$  is to be found experimentally in each case. Moreover, in order to avoid frequent moves from  $x$  to a close solution one may take a smaller value for  $\alpha$  when  $\rho(x, x'')$  is small. More sophisticated choices of a function of  $\alpha\rho(x, x'')$  could be made through some learning process.

(ii) The **variable neighborhood decomposition search** (VNDS) method [70] extends the basic VNS into a two-level VNS scheme based on decomposition of the problem. It is presented in Algorithm 12, where  $t_d$  is an additional parameter that represents the running time allowed for solving decomposed (smaller sized) problems by basic VNS (line 5).

For ease of presentation, but without loss of generality, we assume that the solution  $x$  represents a set of some attributes. In step 4 we denote by  $y$  a set of  $k$  solution attributes present in  $x'$  but not in  $x$  ( $y = x' \setminus x$ ). In step 5 we find the local optimum

---

**Algorithm 12** Variable neighborhood decomposition search

---

```

Function VNDS ( $x, k_{max}, t_{max}, t_d$ )
  1 repeat
  2    $k \leftarrow 1$ 
  3   repeat
  4      $x' \leftarrow \text{Shake}(x, k); y \leftarrow x' \setminus x$ 
  5      $y' \leftarrow \text{BVNS}(y, k, t_d); x'' = (x' \setminus y) \cup y'$ 
  6      $x''' \leftarrow \text{FirstImprovement}(x'')$ 
  7      $x, k \leftarrow \text{NeighborhoodChange}(x, x''', k)$ 
    until  $k = k_{max}$ 
  until  $t > t_{max}$ 
return  $x$ 

```

---

$y'$  in the space of  $y$ ; then we denote with  $x''$  the corresponding solution in the whole space  $S$  ( $x'' = (x' \setminus y) \cup y'$ ). We notice that exploiting some *boundary effects* in a new solution can significantly improve solution quality. That is why, in step 6, the local optimum  $x'''$  is found in the whole space  $S$  using  $x''$  as an initial solution. If this is time consuming, then at least a few local search iterations should be performed.

VNDS can be viewed as embedding the classical successive approximation scheme (which has been used in combinatorial optimization at least since the 1960s, see, e.g., [52]) in the VNS framework.

### 3.4 Variable Neighborhood Formulation Space Search

Traditional ways to tackle an optimization problem consider a given formulation and search in some way through its feasible set  $X$ . Given that the same problem can often be formulated in different ways, it is possible to extend search paradigms to include jumps from one formulation to another. Each formulation should lend itself to some traditional search method, its “local search” that works totally within this formulation, and yields a final solution when started from some initial solution. Any solution found in one formulation should easily be translatable to its equivalent solution in any other formulation. We may then move from one formulation to another by using the solution resulting from the local search of the former as an initial solution for the local search of the latter. Such a strategy will of course only be useful when local searches in different formulations behave differently.

This idea was recently investigated in [87] using an approach that systematically alternates between different formulations for solving the circle packing problem (CPP). It is shown there that a stationary point for a nonlinear programming formulation of CPP in Cartesian coordinates is not necessarily a stationary point in polar coordinates. A method called *Reformulation Descent* (RD) that alternates between these two formulations until the final solution is stationary with respect to both formulations is suggested. Results obtained were comparable with the best known values, but were achieved some 150 times faster than with an alternative single

formulation approach. In the same paper the idea suggested above of *Formulation Space Search* (FSS) is also introduced, using more than two formulations. Some research in that direction has also been reported in [74, 83, 88, 90]. One methodology that uses the variable neighborhood idea when searching through the formulation space is given in Algorithms 13 and 14. Here  $\phi$  ( $\phi'$ ) denotes a formulation from a given space  $\mathcal{F}$ ,  $x$  ( $x'$ ) denotes a solution in the feasible set defined with that formulation, and  $\ell \leq \ell_{\max}$  is the formulation neighborhood index. Note that Algorithm 14 uses a reduced VNS strategy in the formulation space  $\mathcal{F}$ . Note also that the `ShakeFormulation()` function must provide a search through the solution space  $\mathcal{S}'$  in order to get a new solution  $x'$ . Any appropriate method can be used for this purpose.

---

**Algorithm 13** Formulation change
 

---

**Function** FormulationChange( $x, x', \phi, \phi', \ell$ )

1 Set  $\ell_{\min}$  and  $\ell_{step}$

2 **if**  $f(\phi', x') < f(\phi, x)$  **then**

3   |    $\phi \leftarrow \phi'$

4   |    $x \leftarrow x'$

5   |    $\ell \leftarrow \ell_{\min}$

**else**

6   |    $\ell \leftarrow \ell + \ell_{step}$

7 **return**  $x, \phi, \ell$

---



---

**Algorithm 14** Reduced variable neighborhood FSS
 

---

**Function** VNFSS( $x, \phi, \ell_{\max}$ )

1 **repeat**

2   |    $\ell \leftarrow 1$  // Initialize formulation in  $\mathcal{F}$

3   |   **while**  $\ell \leq \ell_{\max}$  **do**

4   |   |    $x', \phi', \ell \leftarrow \text{ShakeFormulation}(x, x', \phi, \phi', \ell)$  //  $(\phi', x') \in (N_\ell(\phi), \mathcal{N}(x))$  at random

5   |   |    $x, \phi, \ell \leftarrow \text{FormulationChange}(x, x', \phi, \phi', \ell)$  // Change formulation

  |   **until** some stopping condition is met

6 **return**  $x$

---

### 3.5 Primal–Dual VNS

For most modern heuristics the difference in value between the optimal solution and the one obtained is completely unknown. Guaranteed performance of the primal heuristic may be determined if a lower bound on the objective function value is known. To this end, the standard approach is to relax the integrality condition on the primal variables, based on a mathematical programming formulation of the problem. However, when the dimension of the problem is large, even the relaxed problem may be impossible to solve exactly by standard commercial solvers. Therefore, it seems

to be a good idea to solve dual relaxed problems heuristically as well. In this way we get guaranteed bounds on the primal heuristic's performance. The next problem arises if we want to get an exact solution within a branch and bound framework since having the approximate value of the relaxed dual does not allow us to branch in an easy way, for example, by exploiting complementary slackness conditions. Thus, the exact value of the dual is necessary.

In primal-dual VNS (PD-VNS) [58] one possible general way to get both the guaranteed bounds and the exact solution is proposed. It is given in Algorithm 15.

---

**Algorithm 15** Basic PD-VNS

---

```

Function PD-VNS ( $x, k_{max}, t_{max}$ )
  1 BVNS ( $x, k_{max}, t_{max}$ )           // Solve primal by VNS
  2 DualFeasible( $x, y$ )             // Find (infeasible) dual such that  $f_P = f_D$ 
  3 DualVNS( $y$ )                  // Use VNS do decrease infeasibility
  4 DualExact( $y$ )                 // Find exact (relaxed) dual
  5 BandB( $x, y$ )                // Apply branch-and-bound method

```

---

In the first stage a heuristic procedure based on VNS is used to obtain a near-optimal solution. In [58] it is shown that VNS with decomposition is a very powerful technique for large-scale simple plant location problems (SPLP) with up to 15,000 facilities and 15,000 users. In the second phase the objective is to find an exact solution of the relaxed dual problem. Solving the relaxed dual is accomplished in three stages: (i) find an initial dual solution (generally infeasible) using the primal heuristic solution and complementary slackness conditions; (ii) find a feasible solution by applying VNS to the unconstrained nonlinear form of the dual; and (iii) solve the dual exactly starting with the found initial feasible solution using a customized “sliding simplex” algorithm that applies “windows” on the dual variables, thus substantially reducing the problem size. On all problems tested, including instances much larger than those previously reported in the literature, the procedure was able to find the exact dual solution in reasonable computing time. In the third and final phase, armed with tight upper and lower bounds obtained from the heuristic primal solution in phase 1 and the exact dual solution in phase 2, respectively, a standard branch-and-bound algorithm is applied to find an optimal solution of the original problem. The lower bounds are updated with the dual sliding simplex method and the upper bounds whenever new integer solutions are obtained at the nodes of the branching tree. In this way it was possible to solve exactly problem instances of sizes up to  $7000 \times 7000$  for uniform fixed costs and  $15,000 \times 15,000$  otherwise.

### 3.6 Variable Neighborhood Branching—VNS for Mixed Integer Linear Programming

The mixed integer linear programming (MILP) problem consists of maximizing or minimizing a linear function, subject to equality or inequality constraints, and

integrality restrictions on some of the variables. The mixed integer programming problem (*MILP*) can be expressed as follows:

$$(MILP) \quad \begin{cases} \min & \sum_{j=1}^n c_j x_j \\ \text{s.t.} & \sum_{j=1}^n a_{ij} x_j \geq b_i \quad \forall i \in M = \{1, 2, \dots, m\} \\ & x_j \in \{0, 1\} \quad \forall j \in \mathcal{B} \neq \emptyset \\ & x_j \geq 0, \text{integer} \quad \forall j \in \mathcal{G} \\ & x_j \geq 0 \quad \forall j \in \mathcal{C} \end{cases},$$

where the set of indices  $N = \{1, 2, \dots, n\}$  is partitioned into three subsets  $\mathcal{B}$ ,  $\mathcal{G}$ , and  $\mathcal{C}$ , corresponding to binary, general integer, and continuous variables, respectively.

Numerous combinatorial optimization problems, including a wide range of practical problems in business, engineering and science, can be modeled as MILP problems. Several special cases, such as knapsack, set packing, cutting and packing, network design, protein alignment, traveling salesman, and other routing problems, are known to be NP-hard [48].

There are several commercial solvers such as CPLEX [75] for solving MILPs. Methods included in such software packages are usually of branch-and-bound (B&B) or of branch-and-cut (B&C) types. Basically, those methods enumerate all possible integer values in some order and perform some restrictions for the cases where such enumeration cannot improve the current best solution.

The connection between local search-based heuristics and exact solvers may be established by introducing the so-called *local branching constraint* [43]. By adding just one constraint into (MILP), the  $k$ th neighborhood of (MILP) is defined. This allows the use of all local search-based metaheuristics, such as tabu search, simulating annealing, VNS. More precisely, given two solutions  $x$  and  $y$  of the (MILP), the distance between  $x$  and  $y$  is defined as follows:

$$\delta(x, y) = \sum_{j \in \mathcal{B}} |x_j - y_j|.$$

Let  $X$  be the solution space of the (MILP) considered. The neighborhood structures  $\{\mathcal{N}_k \mid k = 1, \dots, k_{\max}\}$  can be defined, knowing the distance  $\delta(x, y)$  between any two solutions  $x, y \in X$ . The set of all solutions in the  $k$ th neighborhood of  $y \in X$  is denoted as  $\mathcal{N}_k(y)$  where

$$\mathcal{N}_k(y) = \{x \in X \mid \delta(x, y) \leq k\}.$$

For the pure 0-1 MILP given above ( $\mathcal{G} = \emptyset$ ),  $\delta(\cdot, \cdot)$  represents the Hamming distance and  $\mathcal{N}_k(y)$  may be expressed by the following *local branching constraint*

$$\delta(x, y) = \sum_{j \in S} (1 - x_j) + \sum_{j \in \mathcal{B} \setminus S} x_j \leq k, \quad (3.2)$$

where  $S = \{j \in \mathcal{B} \mid y_j = 1\}$ .

In [71] we developed a general VNS procedure for solving 0-1 MILPs. An exact MILP solver (CPLEX) is used as a black box for finding the best solution in the neighborhood, based on the given formulation (MILP) plus the added local

---

**Algorithm 16** VNS branching

---

```

Function VnsBra(total_time_limit, node_time_limit, k_step, x_opt)
  1 TL := total_time_limit; UB :=  $\infty$ ; first := true
  2 stat := MIPSOLVE(TL, UB, first, x_opt, f_opt)
  3 x.cur := x_opt; f.cur := f_opt
  4 while (elapsedtime < total_time_limit) do
  5   cont := true; rhs := 1; first := false
  6   while (cont or elapsedtime < total_time_limit) do
  7     TL = min(node_time_limit, total_time_limit - elapsedtime)
  8     add local br. constr.  $\delta(x, x.cur) \leq rhs$ ; UB := f.cur
  9     stat := MIPSOLVE(TL, UB, first, x.next, f.next)
 10    switch stat do
 11      case "opt_sol_found":
 12        | reverse last local br. const. into  $\delta(x, x.cur) \geq rhs + 1$ 
 13        | x.cur := x.next; f.cur := f.next; rhs := 1;
 14      case "feasible_sol_found":
 15        | reverse last local br. constr. into  $\delta(x, x.cur) \geq 1$ 
 16        | x.cur := x.next; f.cur := f.next; rhs := 1;
 17      case "proven_infeasible":
 18        | remove last local br. constr.; rhs := rhs+1;
 19      case "no_feasible_sol_found":
 20        | cont := false
 21    if f.cur < f.opt then
 22      | x_opt := x.cur; f.opt := f.cur; k.cur := k.step;
 23    else
 24      | k.cur := k.cur+k.step;
 25    remove all added constraints; cont := true
 26    while cont and (elapsedtime < total_time_limit) do
 27      | add constraints  $k.cur \leq \delta(x, x.opt) < k.cur + k.step$ 
 28      | TL := total_time_limit - elapsedtime; UB :=  $\infty$ ; first := true
 29      | stat := MIPSOLVE(TL, UB, first, x.cur, f.cur)
 30      | remove last two added constraints; cont = false
 31      | if stat = "proven_infeasible" or "no_feasible" then
        |   | cont := true; k.cur := k.cur+k.step

```

---

branching constraints. Shaking is performed using the Hamming distance defined above. A detailed description of this VNS branching method is provided in Algorithm 16. The variables and constants used in the algorithm are defined as follows [71]:

- . UB—input variable for CPLEX solver which represents the current upper bound.
- . *first*—logical input variable for CPLEX solver which is *true* if the first solution lower than UB is asked for in the output; if *first* = *false*, CPLEX returns the best solution found so far.
- . TL—maximum time allowed for running CPLEX.
- . *rhs*—right-hand side of the local branching constraint; it defines the size of the neighborhood within the inner or VND loop.

- . *cont*—logical variable which indicates if the inner loop continues (`true`) or not (`false`).
- . *x\_opt* and *f\_opt*—incumbent solution and corresponding objective function value.
- x\_cur*, *f\_cur*, *k\_cur*—current solution, objective function value and neighborhood from where VND local search starts.
- . *x\_next* and *f\_next*—solution and corresponding objective function value obtained by CPLEX in inner loop.

### 3.7 Variable Neighborhood Search for Continuous Global Optimization

The general form of the continuous constrained nonlinear global optimization problem (GOP) is given as follows:

$$(GOP) \quad \begin{cases} \min f(x) \\ \text{s.t. } g_i(x) \leq 0 \quad \forall i \in \{1, 2, \dots, m\} \\ h_i(x) = 0 \quad \forall i \in \{1, 2, \dots, r\} \\ a_j \leq x_j \leq b_j \quad \forall j \in \{1, 2, \dots, n\} \end{cases}$$

where  $x \in R^n$ ,  $f : R^n \rightarrow R$ ,  $g_i : R^n \rightarrow R$ ,  $i = 1, 2, \dots, m$ , and  $h_i : R^n \rightarrow R$ ,  $i = 1, 2, \dots, r$ , are possibly nonlinear continuous functions, and  $a, b \in R^n$  are the variable bounds. A box constraint GOP is defined when only the variable bound constraints are present in the model.

The GOP naturally arises in many applications, e.g., in advanced engineering design, data analysis, financial planning, risk management, scientific modeling. Most cases of practical interest are characterized by multiple local optima and, therefore, a search effort of global scope is needed to find the globally optimal solution.

If the feasible set  $X$  is convex and objective function  $f$  is convex, then (GOP) is relatively easy to solve, i.e., the Karush–Kuhn–Tucker conditions can be applied. However, if  $X$  is not a convex set or  $f$  is not a convex function, we can have many local optima and the problem may not be solved with classical techniques.

For solving (GOP), VNS has been used in two different ways: (i) with neighborhoods induced by using an  $\ell_p$  norm and (ii) without using an  $\ell_p$  norm.

(i) **VNS with  $\ell_p$  norm neighborhoods** [42, 79, 84, 86]. A natural approach in applying VNS for solving GOPs is to induce neighborhood structures  $\mathcal{N}_k(x)$  from an  $\ell_p$  metric such as

$$\rho(x, y) = \left( \sum_{i=1}^n |x_i - y_i|^p \right)^{1/p} \quad (1 \leq p < \infty) \quad (3.3)$$

or

$$\rho(x, y) = \max_{1 \leq i \leq n} |x_i - y_i| \quad (p \rightarrow \infty). \quad (3.4)$$

The neighborhood  $\mathcal{N}_k(x)$  denotes the set of solutions in the  $k$ th neighborhood of  $x$ , and using the metric  $\rho$ , it is defined as

$$\mathcal{N}_k(x) = \{y \in X \mid \rho(x, y) \leq \rho_k\} \quad (3.5)$$

or

$$\mathcal{N}_k(x) = \{y \in X \mid \rho_{k-1} < \rho(x, y) \leq \rho_k\}, \quad (3.6)$$

where  $\rho_k$ , known as the radius of  $\mathcal{N}_k(x)$ , is monotonically increasing with  $k$ .

For solving box constraint GOPs, both [42] and [79] use neighborhoods as defined in Equation (3.6). The basic differences between the two are as follows: (1) in the procedure suggested in [79] the  $\ell_\infty$  norm is used, while in [42] the choice of metric is either left to the analyst or changed automatically in some predefined order; (2) the commercial solver SNOPT [49] is used as a local search procedure within VNS in [79], while in [42], the analyst may choose one out of six different convex minimizers. A VNS-based heuristic for solving the generally constrained GOP is suggested in [84]. There, the problem is first transformed into a sequence of box constrained problems within the well-known exterior point method:

$$\min_{a \leq x \leq b} F_{\mu, q}(x) = f(x) + \frac{1}{\mu} \sum_{i=1}^m (\max\{0, g_i(x)\})^q + \sum_{i=1}^r |h_i(x)|^q, \quad (3.7)$$

where  $\mu$  and  $q \geq 1$  are a positive penalty parameter and penalty exponent, respectively. Algorithm 17 outlines the steps for solving the box constraint subproblem as proposed in [84]:

---

**Algorithm 17** VNS using a  $\ell_p$  norm

---

**Function** Glob-VNS ( $x^*, k_{\max}, t_{\max}$ )

```

1 Select the set of neighborhood structures  $\mathcal{N}_k$ ,  $k = 1, \dots, k_{\max}$ 
2 Select the array of random distributions types and an initial point  $x^* \in X$ 
3  $x \leftarrow x^*$ ,  $f^* \leftarrow f(x)$ ,  $t \leftarrow 0$ 
4 while  $t < t_{\max}$  do
5    $k \leftarrow 1$ 
6   repeat
7     for all distribution types do
8        $y \leftarrow \text{Shake}(x^*, k)$  // Get  $y \in \mathcal{N}_k(x^*)$  at random
9        $y' \leftarrow \text{BestImprovement}(y)$  // Apply LS to obtain a local minimum  $y'$ 
10      if  $f(y') < f^*$  then
11         $x^* \leftarrow y'$ ,  $f^* \leftarrow f(y')$ , go to line 5
12       $k \leftarrow k + 1$ 
13    until  $k = k_{\max}$ 
14     $t \leftarrow \text{CpuTime}()$ 

```

---

The Glob-VNS procedure from Algorithm 17 contains the following parameters in addition to  $k_{\max}$  and  $t_{\max}$ : (1) *Values of radii*  $\rho_k$ ,  $k = 1, \dots, k_{\max}$ . Those

values may be defined by the user or calculated automatically in the minimizing process; (2) *Geometry* of neighborhood structures  $\mathcal{N}_k$ , defined by the choice of metric. Usual choices are the  $\ell_1$ ,  $\ell_2$ , and  $\ell_\infty$  norms; (3) *Distribution* used for obtaining the random point  $y$  from  $\mathcal{N}_k$  in the *Shaking* step. Uniform distribution in  $\mathcal{N}_k$  is the obvious choice, but other distributions may lead to much better performance on some problems. Different choices of geometric neighborhood shapes and random point distributions lead to different VNS-based heuristics.

(ii) **VNS without using  $\ell_p$  norm.** Two different neighborhoods,  $N_1(x)$  and  $N_2(x)$ , are used in the VNS-based heuristic suggested in [97]. In  $N_1(x)$ ,  $r$  (a parameter) random directions from the current point  $x$  are generated and a one-dimensional search along each direction is performed. The best point (out of  $r$ ) is selected as a new starting solution for the next iteration, if it is better than the current one. If not, as in VND, the search is continued within the next neighborhood  $N_2(x)$ . The new point in  $N_2(x)$  is obtained as follows. The current solution is moved for each  $x_j$  ( $j = 1, \dots, n$ ) by a value  $\Delta_j$ , taken at random from interval  $(-\alpha, \alpha)$ , i.e.,  $x_j^{(\text{new})} = x_j + \Delta_j$  or  $x_j^{(\text{new})} = x_j - \Delta_j$ . Points obtained by the plus or minus sign for each variable define the neighborhood  $N_2(x)$ . If a relative increase of 1% in the value of  $x_j^{(\text{new})}$  produces a better solution than  $x^{(\text{new})}$ , the + sign is chosen; otherwise the – sign is chosen.

Neighborhoods  $N_1$  and  $N_2$  are used for designing two algorithms. The first, called VND, iterates over these neighborhoods until there is no improvement in the solution value. In the second variant, a local search is performed with  $N_2$  and  $k_{\max}$  is set to 2 for the shaking step. In other words, a point from the neighborhood  $N_2$  is obtained by generating a random direction followed by a line search along it (as prescribed for  $N_1$ ) and then by changing each of the variables (as prescribed for  $N_2$ ).

It is interesting to note that computational results reported by all VNS-based heuristics were very promising. They usually outperformed other recent approaches from the literature.

### 3.8 Mixed Integer Nonlinear Programming (MINLP) Problem

The problems we address here are cast in the following general form:

$$(MINLP) \quad \begin{cases} \min & f(x) \\ \text{s.t.} & \ell_i \leq g_i(x) \leq u_i \quad \forall i \in \{1, \dots, m\} \\ & a_j \leq x_j \leq b_j \quad \forall j \in N \\ & x_j \in \{0, 1\} \quad \forall j \in \mathcal{B} \neq \emptyset \\ & x_j \geq 0, \text{integer} \quad \forall j \in \mathcal{G} \\ & x_j \geq 0 \quad \forall j \in \mathcal{C} \end{cases},$$

where the set of indices  $N = \{1, 2, \dots, n\}$ , as in the formulation of MILP, is partitioned into three subsets  $\mathcal{B}$ ,  $\mathcal{G}$  and  $\mathcal{C}$ , corresponding to binary, general integer and

continuous variables, respectively. Therefore, in the above formulation, the  $x_j$  are the decision variables,  $f : R^n \rightarrow R$  is a possibly nonlinear function,  $g : R^n \rightarrow R^m$  is a vector of  $m$  possibly nonlinear functions (assumed to be differentiable),  $\ell, u \in R^n$  are the constraint bounds (which may be set to  $\pm\infty$ ), and  $a, b \in R^n$  are the variable bounds.

In order to apply VNS for solving (MINLP), one needs to answer three questions: (i) how to define the set of neighborhoods around any solution  $x$ ; (ii) how to perform (local) search starting from any point and finishing with a feasible solution; (iii) how to get a feasible solution starting from an infeasible point.

(i) **Neighborhoods.** Naturally, for the set of binary variables  $x_j, j \in \mathcal{B}$ , the Hamming distance, expressed by local branching constraints (3.2), can be used. For the set of continuous variables  $x_j, j \in \mathcal{G}$  one can use the  $\ell_p$  norm (3.3) or (3.4); for the set of integer variables, either an extension of formula (3.2) given in [43] or (3.6) can be used. The point  $x' \in \mathcal{N}_k(x)$  denotes a  $k$ th neighborhood solution of combined binary, continuous, and integer parts.

(ii) **Local search.** The local search phase mainly depends on available software. The simplest way is just to use an existing commercial solver for MINLP by adding constraints that define neighborhood  $N_k$ . Such an approach for solving (MILP) is applied in the local branching [43] and VNS branching [71] methods explained earlier. Since such a solver for MINLP does not exist on the market, it becomes necessary to split the problem at any branching node into easier subproblems and alternately solve these subproblems until an improved feasible solution is hopefully found. For example, integrality conditions may be relaxed in one subproblem, and continuous variables fixed in the next. The partition into subproblems depends mostly on existing solvers and their qualities. By relaxing all binary and integer variables, a NLP problem is obtained, whose complexity depends on the properties of  $f(x)$  and  $g_i(x), i \in \{1, \dots, m\}$ . If all functions are convex, the problem may be much easier to solve. The relaxed solution is then used to get a lower bound within a branch and bound (BB) enumerative procedure. Thus, the quality of the solution obtained in this local search phase mostly depends on the way different solvers are combined and on the quality of such solvers.

(iii) **Feasible solution.** Realistically sized MINLPs can often have thousands (or tens of thousands) of variables (continuous and integer) and nonconvex constraints. With such sizes, it becomes a difficult challenge to even find a feasible solution, and BB algorithms become almost useless. Some good solvers targeting convex MINLPs exist in the literature [1, 24, 26, 45, 46, 78]; and although they can all be used on nonconvex MINLPs as well (forsaking the optimality guarantee), their quality varies wildly in practice with the instance of the problem being solved, resulting in a high fraction of “false negatives” (i.e., feasible problems for which no feasible solution was found). The feasibility pump (FP) idea was recently extended to convex MINLPs [25], but again this does not work so well when applied to unmodified nonconvex MINLPs.

In a recent paper [80] an effective and reliable MINLP heuristic based on VNS is suggested, called Relaxed-Exact Continuous-Integer Problem Exploration

(RECIPE for short). RECIPE puts together a global search phase based on VNS and a local search phase based on a BB-type heuristic. The VNS global phase relies on neighborhoods defined as hyperrectangles for the continuous and general integer variables and by local branching constraints for the binary variables. The local phase employs a BB solver for convex MINLPs [46], which is applied to non-convex MINLPs heuristically. A local NLP solution using a sequential quadratic programming (SQP) algorithm [49] supplies an initial constraint-feasible solution to be employed by the BB as initial upper bound. RECIPE (see Algorithm 18) is an efficient, effective, and reliable general-purpose algorithm for solving complex MINLPs of small and medium scale. The original contribution of RECIPE is the particular combination of some well-known and well-tested tools to produce a very powerful global optimization method. It turns out that RECIPE, acting on the whole MINLPLib library [29], is able to find optima equal to or better than those reported in the literature for 55% of the instances. The closest competitor is SBB+CONOPT with 37%. The known optima are improved in 7% of the cases.

---

**Algorithm 18** The RECIPE heuristic for solving MINLP

---

```

Function RECIPE ( $a, b, k_{max}, t_{max}, x^*$ )
  1  $x^* = (a + b)/2; t \leftarrow 0$ 
  2 while  $t < t_{max}$  do
  3    $k \leftarrow 1$ 
  4   while  $k \leq k_{max}$  do
  5      $i \leftarrow 1$ 
  6     while  $i \leq b$  do
  7       Sample  $\bar{x} \in \mathcal{N}_k(x^*)$  at random
  8        $x \leftarrow \text{SQP}(\bar{x})$ 
  9       if  $x$  not feasible then
 10          $x \leftarrow \bar{x}$ 
 11        $x' \leftarrow \text{BB}(x, k, k_{max})$ 
 12       if  $x'$  is better than  $x^*$  then
 13          $x^* \leftarrow x'; k \leftarrow 0$ ; Exit loop  $i$ 
 14        $i \leftarrow i + 1$ 
 15    $k \leftarrow k + 1$ 
 16  $t \leftarrow \text{CpuTime}()$ 

```

---

### 3.9 Discovery Science

In all the above applications, VNS is used as an optimization tool. It can also lead to results in “discovery science,” i.e., help in the development of theories. This has been done for graph theory in a long series of papers with the common title “Variable neighborhood search for extremal graphs” that report on the development and applications of the AutoGraphiX (AGX) system [4, 36, 37]. This system addresses the following problems:

- Find a graph satisfying given constraints.
- Find optimal or near optimal graphs for an invariant subject to constraints.
- Refute a conjecture.
- Suggest a conjecture (or repair or sharpen one).
- Provide a proof (in simple cases) or suggest an idea of proof.

A basic idea is then to address all of these problems as parametric combinatorial optimization problems on the infinite set of all graphs (or in practice some smaller subset) using a generic heuristic to explore the solution space. This is done by applying VNS to find extremal graphs with a given number  $n$  of vertices (and possibly also a given number of edges). Extremal graphs may be viewed as a family of graphs that maximize some invariant such as the independence number or chromatic number, possibly subject to constraints. We may also be interested in finding lower and upper bounds on some invariant for a given graph  $G$ . Once an extremal graph is obtained, VND with many neighborhoods may be used to build other such graphs. Those neighborhoods are defined by modifications of the graphs such as the removal or addition of an edge, rotation of an edge. Once a set of extremal graphs, parameterized by their order, is found, their properties are explored with various data mining techniques, leading to conjectures, refutations, and simple proofs or ideas of proof.

The current list of references in the series “VNS for extremal graphs” is given by [3–8, 10–12, 17–19, 23, 32, 34, 36, 37, 40, 47, 53, 61–63, 72, 73, 94, 95]. Another list of papers, not included in this series is given in [9, 13–16, 33, 35, 54–57, 60, 96]. Papers in these two lists cover a variety of topics:

- (i) **Principles of the approach** [36, 37] and its implementation [4];
- (ii) **Applications to spectral graph theory**, e.g., bounds on the index for various families of graphs, graphs maximizing the index subject to some conditions [3, 17, 23, 40, 57];
- (iii) **Studies of classical graph parameters**, e.g., independence, chromatic number, clique number, average distance [5, 9–12, 94, 95];
- (iv) **Studies of little known or new parameters of graphs**, e.g., irregularity, proximity, and remoteness [13, 62];
- (v) **New families of graphs discovered by AGX**, e.g., bags, which are obtained from complete graphs by replacing an edge by a path, and bugs, which are obtained by cutting the paths of a bag [8, 72];
- (vi) **Applications to mathematical chemistry**, e.g., study of chemical graph energy, and of the Randić index [18, 19, 34, 47, 53–56, 61];
- (vii) **Results of a systematic study of 20 graph invariants**, which led to almost 1500 new conjectures, more than half of which were proved by AGX and over 300 by various mathematicians [7];
- (viii) **Refutation or strengthening of conjectures from the literature** [6, 33, 56];
- (ix) **Surveys and discussions about various discovery systems in graph theory**, assessment of the state of the art and the forms of interesting conjectures together with proposals for the design of more powerful systems [35, 60].

## 3.10 Conclusions

The general schemes of variable neighborhood search have been presented and discussed. In order to evaluate research development related to VNS, one needs a list of the desirable properties of metaheuristics. Eight of these are presented in Hansen and Mladenović (2003):

- (i) *Simplicity*: the metaheuristic should be based on a simple and clear principle, which should be widely applicable;
- (ii) *Precision*: the steps of the metaheuristic should be formulated in precise mathematical terms, independent of possible physical or biological analogies which may have been the initial source of inspiration;
- (iii) *Coherence*: all steps of the heuristics for solving a particular problem should follow naturally from the metaheuristic principles;
- (iv) *Effectiveness*: heuristics for particular problems should provide optimal or near-optimal solutions for all or at least most realistic instances. Preferably, they should find optimal solutions for most benchmark problems for which such solutions are known;
- (v) *Efficiency*: heuristics for particular problems should take a moderate computing time to provide optimal or near-optimal solutions, or comparable or better solutions than the state of the art;
- (vi) *Robustness*: the performance of the metaheuristics should be consistent over a variety of instances, i.e., not merely fine tuned to some training set and not so good elsewhere;
- (vii) *User friendliness*: the metaheuristics should be clearly expressed, easy to understand and, most importantly, easy to use. This implies they should have as few parameters as possible, ideally none;
- (viii) *Innovation*: the principle of the metaheuristic and/or the efficiency and effectiveness of the heuristics derived from it should lead to new types of application.

This list has been completed with three more items added by one member of the present team and his collaborators:

- (ix) *Generality*: the metaheuristic should lead to good results for a wide variety of problems;
- (x) *Interactivity*: the metaheuristic should allow the user to incorporate his knowledge to improve the resolution process;
- (xi) *Multiplicity*: the metaheuristic should be able to produce several near-optimal solutions from which the user can choose.

As shown above, VNS possesses, to a great extent, all of the above properties. This has led to heuristics which are among the very best ones for many problems. Interest in VNS is growing quickly. This is evidenced by the increasing number of papers published each year on this topic (10 years ago, only a few; 5 years ago, about a dozen; and about 50 in 2007). Moreover, the 18th EURO mini-conference held in Tenerife in November 2005 was entirely devoted to VNS. It led to special

issues of *IMA Journal of Management Mathematics* in 2007 [81], *European Journal of Operational Research* [68], and *Journal of Heuristics* [89] in 2008. In retrospect, it appears that the good shape of VNS research is due to the following perspectives, strongly influenced by Karl Popper's philosophy of science [91]: (i) in devising heuristics favor insight over efficiency (which comes later) and (ii) learn from heuristic failures.

## References

1. Abhishek, K., Leyffer, S., Linderoth, J.: FilMINT: An outer-approximation based solver for nonlinear mixed-integer programs. Technical Report ANL/MCSP1374- 0906, Argonne National Laboratory, 2007
2. Aloise, D.J., Aloise, D., Rocha, C.T.M., Ribeiro, C.C., Ribeiro, J.C., Moura, L.S.S.: Scheduling workover rigs for onshore oil production. *Discrete Appl. Math.* **154**, 695–702 (2006)
3. Aouchiche, M., Bell, F.K., Cvetković, D., Hansen, P., Rowlinson, P., Simić, S.K., Stevanović, D.: Variable neighborhood search for extremal graphs 16. Some conjectures related to the largest eigenvalue of a graph. *Eur. J. Oper. Res.* **191**, 661–676 (2008)
4. Aouchiche, M., Bonnafoy, J.M., Fidahoussen, A., Caporossi, G., Hansen, P., Hiesse, L., Lacheré, J., Monhait, A.: Variable neighborhood search for extremal graphs 14. The AutoGraphiX 2 system. In: Liberti, L., Maculan, N. (eds.) *Global Optimization: From Theory to Implementation*, pp. 281–309. Springer, Berlin (2006)
5. Aouchiche, M., Brinkmann, G., Hansen, P.: Variable neighborhood search for extremal graphs 21. Conjectures and results about the independence number. *Discrete Appl. Math.* **156**, 2530–2542 (2009)
6. Aouchiche, M., Caporossi, G., Cvetković, D.: Variable neighborhood search for extremal graphs 8. Variations on Graffiti 105. *Congressus Numerantium* **148**, 129–144 (2001)
7. Aouchiche, M., Caporossi, G., Hansen, P.: Variable Neighborhood search for extremal graphs 20. Automated comparison of graph invariants. *MATCH. Commun. Math. Comput. Chem.* **58**, 365–384 (2007)
8. Aouchiche, M., Caporossi, G., Hansen, P.: Variable neighborhood search for extremal graphs 27. Families of extremal graphs. *Les Cahiers du GERAD G-2007-87* (2007)
9. Aouchiche, M., Caporossi, G., Hansen, P., Laffay, M.: AutoGraphiX: a survey. *Electron. Notes Discrete Math.* **22**, 515–520 (2005)
10. Aouchiche, M., Favaron, O., Hansen, P.: Variable neighborhood search for extremal graphs 22. Extending bounds for independence to upper irredundance. *Discrete Appl. Math.* **157**, 3497–3510 (2009)
11. Aouchiche, M., Favaron, O., Hansen, P.: Recherche à voisinage variable de graphes extrêmes 26. Nouveaux résultats sur la maille (French). *Les Cahiers du GERAD G-2007-55* (2007)
12. Aouchiche, M., Hansen, P.: Recherche à voisinage variable de graphes extrêmes 13. À propos de la maille (French). *RAIRO Oper. Res.* **39**, 275–293 (2005)
13. Aouchiche, M., Hansen, P.: Automated results and conjectures on average distance in graphs. *Graph Theory in Paris, Trends Math.* **6**, 21–36 (2007)
14. Aouchiche, M., Hansen, P.: On a conjecture about the Randic index. *Discrete Math.* **307**, 262–265 (2007)
15. Aouchiche, M., Hansen, P.: Bounding average distance using minimum degree. *Graph Theory Notes of New York* (to appear) (2009)
16. Aouchiche, M., Hansen, P.: Nordhaus-Gaddum relations for proximity and remoteness in graphs. *Les Cahiers du GERAD G-2008-36* (2008)
17. Aouchiche, M., Hansen, P., Stevanović, D.: Variable neighborhood search for extremal graphs 17. Further conjectures and results about the index. *Discussiones Mathematicae: Graph Theory* (to appear) (2009)

18. Aouchiche, M., Hansen, P., Zheng, M.: Variable neighborhood search for extremal graphs 18. Conjectures and results about the Randic index. *MATCH. Commun. Math. Comput. Chem.* **56**, 541–550 (2006)
19. Aouchiche, M., Hansen, P., Zheng, M.: Variable Neighborhood Search for Extremal Graphs 19. Further Conjectures and Results about the Randic Index. *MATCH. Commun. Math. Comput. Chem.* **58**, 83–102 (2007)
20. Audet, C., Bâcher, V., Le, Digabel, S.: Nonsmooth optimization through mesh adaptive direct search and variable neighborhood search. *J. Global Optim.* **41**, 299–318 (2008)
21. Audet, C., Brimberg, J., Hansen, P., Mladenović, N.: Pooling problem: alternate formulation and solution methods. *Manage. Sci.* **50**, 761–776 (2004)
22. Belacel, N., Hansen, P., Mladenović, N.: Fuzzy J-means: a new heuristic for fuzzy clustering. *Pattern Recognit.* **35**, 2193–2200 (2002)
23. Belhaiza, S., de, Abreu, N., Hansen, P., Oliveira, C.: Variable neighborhood search for extremal graphs 11. Bounds on algebraic connectivity. In: Avis, D., Hertz, A., Marcotte, O. (eds.) *Graph Theory and Combinatorial Optimization*, pp. 1–16. (2007)
24. Bonami, P., Biegler, L.T., Conn, A.R., Cornuejols, G., Grossmann, I.E., Laird, C.D., Lee, J., Lodi, A., Margot, F., Sawaya, N., Wachter, A.: An algorithmic framework for convex mixed integer nonlinear programs. Technical Report RC23771, IBM Corporation (2005)
25. Bonami, P., Cornuejols, G., Lodi, A., Margot, F.A.: Feasibility pump for mixed integer nonlinear programs. Technical Report RC23862 (W0602-029), IBM Corporation (2006)
26. Bonami, P., Lee, J.: BONMIN User's manual. Technical Report, IBM Corporation (2007)
27. Brimberg, J., Hansen, P., Mladenović, N., Taillard, É.: Improvements and comparison of heuristics for solving the multisource Weber problem. *Oper. Res.* **48**, 444–460 (2000)
28. Brimberg, J., Mladenović, N.: A variable neighborhood algorithm for solving the continuous location-allocation problem. *Stud. Locat. Anal.* **10**, 1–12 (1996)
29. Bussieck, M.R., Drud, A.S., Meeraus, A.: MINLPLib - A collection of test models for mixed-integer nonlinear programming. *INFORMS J. Comput.* **15**, 114–119 (2003)
30. Canuto, S., Resende, M., Ribeiro, C.: Local search with perturbations for the prize-collecting Steiner tree problem in graphs. *Networks* **31**, 201–206 (2001)
31. Caporossi, G., Alamargot, D., Chesnet, D.: Using the computer to study the dynamics of the handwriting processes. *Lect. Notes Comput. Sci.* **3245**, 242–254 (2004)
32. Caporossi, G., Cvetković, D., Gutman, I., Hansen, P.: Variable neighborhood search for extremal graphs 2. Finding graphs with extremal energy. *J. Chem. Inf. Comput. Sci.* **39**, 984–996 (1999)
33. Caporossi, G., Dobrynin, A.A., Gutman, I., Hansen, P.: Trees with palindromic Hosoya polynomials. *Graph Theory Notes New York* **37**, 10–16 (1999)
34. Caporossi, G., Gutman, I., Hansen, P.: Variable neighborhood search for extremal graphs 4. Chemical trees with extremal connectivity index. *Comput. Chem.* **23**, 469–477 (1999)
35. Caporossi, G., Gutman, I., Hansen, P., Pavlović, L.: Graphs with maximum connectivity index. *Comput. Biol. Chem.* **27**, 85–90 (2003)
36. Caporossi, G., Hansen, P.: Variable neighborhood search for extremal graphs 1. The Auto-GraphiX system. *Discrete Math.* **212**, 29–44 (2000)
37. Caporossi, G., Hansen, P.: Variable neighborhood search for extremal graphs 5. Three ways to automate finding conjectures. *Discrete Math.* **276**, 81–94 (2004)
38. Carrabs, F., Cordeau, J.-F., Laporte, G.: Variable neighbourhood search for the pickup and delivery traveling salesman problem with LIFO loading. *INFORMS J. Comput.* **19**, 618–632 (2007)
39. Carrizosa, E., Martín-Barragán, B., Plastria, F., Romero, Morales, D.: On the selection of the globally optimal prototype subset for nearest-neighbor classification. *INFORMS J. Comput.* **19**, 470–479 (2007)
40. Cvetkovic, D., Simic, S., Caporossi, G., Hansen, P.: Variable neighborhood search for extremal graphs 3. On the largest eigenvalue of color-constrained trees. *Linear Multilinear Algebra* **49**, 143–160 (2001)

41. Davidon, W.C.: Variable metric algorithm for minimization. Argonne National Laboratory Report ANL-5990 (1959)
42. Dražić, M., Kovacevic-Vujcić, V., Cangalović, M., Mladenović, N.: GLOB - A new VNS-based software for global optimization In: Liberti L., Maculan N. (eds.) Global Optimization: From Theory to Implementation, pp. 135–144, Springer, Berlin (2006)
43. Fischetti, M., Lodi, A.: Local branching. *Math. Programming* **98**, 23–47 (2003)
44. Fletcher, R., Powell, M.J.D.: Rapidly convergent descent method for minimization. *Comput. J.* **6**, 163–168 (1963)
45. Fletcher, R., Leyffer, S.: Solving mixed integer nonlinear programs by outer approximation. *Math. Program.* **66**, 327–349 (1994)
46. Fletcher, R., Leyffer, S.: Numerical experience with lower bounds for MIQP branch-and-bound. *SIAM J. Optim.* **8**, 604–616 (1998)
47. Fowler, P.W., Hansen, P., Caporossi, G., Soncini, A.: Variable neighborhood search for extremal graphs 7. Polyenes with maximum HOMO-LUMO gap. *Chem. Phys. Lett.* **49**, 143–146 (2001)
48. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. Freeman, New York (1978)
49. Gill, P., Murray, W., Saunders, M.A.: SNOPT: An SQP algorithms for largescale constrained optimization. *SIAM J. Optim.* **12**, 979–1006 (2002)
50. Gill, P., Murray, W., Wright, M.: Practical Optimization. Academic Press, London (1981)
51. Glover, F., Kochenberger, G., (eds.) Handbook of Metaheuristics. Kluwer, Dorchester, London, New York (2003)
52. Griffith, R.E., Stewart, R.A.: A nonlinear programming technique for the optimization of continuous processing systems. *Manage. Sci.* **7**, 379–392 (1961)
53. Gutman, I., Hansen, P., Mélot, H., Variable neighborhood search for extremal graphs 10. Comparison of irregularity indices for chemical trees. *J. Chem. Inf. Model.* **45**, 222–230 (2005)
54. Gutman, I., Miljković, O., Caporossi, G., Hansen, P.: Alkanes with small and large Randić connectivity indices. *Chem. Phys. Lett.* **306**, 366–372 (1999)
55. Hansen, P.: Computers in Graph Theory. *Graph Theory Notes New York* **43**, 20–39 (2002)
56. Hansen, P.: How far is, should and could be conjecture-making in graph theory an automated process? *Graph Discov., Dimacs Series Discrete Math. Theor. Comput. Sci.* **69**, 189–229 (2005)
57. Hansen, P., Aouchiche, M., Caporossi, G., Mélot, H., Stevanović, D.: What forms do interesting conjectures have in graph theory? *Graph Discov., Dimacs Ser. Discrete Math. Theor. Comput. Sci.* **69**, 231–251 (2005)
58. Hansen, P., Brimberg, J., Urošević, D., Mladenović, N.: Primal-dual variable neighborhood search for the simple plant location problem. *INFORMS J. Comput.* **19**, 552–564 (2007)
59. Hansen, P., Jaumard, B., Mladenović, N., Parreira, A.: Variable neighborhood search for weighted maximum satisfiability problem. *Les Cahiers du GERAD G-2000-62* (2000)
60. Hansen, P., Mélot, H.: Computers and discovery in algebraic graph theory. *Linear Algebra Appl.* **356**, 211–230 (2002)
61. Hansen, P., Mélot, H.: Variable neighborhood search for extremal graphs 6. Analysing bounds for the connectivity index. *J. Chem. Inf. Comput. Sci.* **43**, 1–14 (2003)
62. Hansen, P., Mélot, H.: Variable neighborhood search for extremal graphs 9. Bounding the irregularity of a graph. *Graph Discov., Dimacs Series Discrete Math. Theor. Comput. Sci.* **69**, 253–264 (2005)
63. Hansen, P., Mélot, H., Gutman, I.: Variable neighborhood search for extremal graphs 12. A note on the variance of bounded degrees in graphs. *MATCH Commun. Math. Comput. Chem.* **54**, 221–232 (2005)
64. Hansen, P., Mladenović, N.: Variable neighborhood search: Principles and applications. *Eur. J. Oper. Research* **130**, 449–467 (2001)
65. Hansen, P., Mladenović, N.: J-Means: A new local search heuristic for minimum sum-of-squares clustering. *Pattern Recognit.* **34**, 405–413 (2001)

66. Hansen, P., Mladenović, N.: Developments of variable neighborhood search. In: Ribeiro, C., Hansen, P. (eds.) *Essays and Surveys in Metaheuristics*, pp. 415–440, Kluwer, Dorchester, London, New York (2001)
67. Hansen, P., Mladenović, N.: Variable neighborhood search. In: Glover F., Kochenberger G. (eds.) *Handbook of Metaheuristics*, pp. 145–184, Kluwer, Dorchester, London, New York (2003)
68. Hansen, P., Mladenović, N., Moreno Pérez, J.A.: Variable neighborhood search. *Eur. J. Oper. Res.* **191**, 593–595 (2008)
69. Hansen, P., Mladenović, N., Moreno Pérez, J.A.: Variable neighborhood search: methods and applications. *4OR. Q. J. Oper. Res.* **6**, 319–360 (2008)
70. Hansen, P., Mladenović, N., Pérez-Brito, D.: Variable neighborhood decomposition search. *J. Heuristics* **7**, 335–350 (2001)
71. Hansen, P., Mladenović, N., Urošević, D.: Variable neighborhood search and local branching. *Comput. Oper. Res.* **33**, 3034–3045 (2006)
72. Hansen, P., Stevanović, D.: Variable neighborhood search for extremal graphs 15. On bags and bugs, *Discrete Appl. Math.* **156**, 986–997 (2005)
73. Hansen, P., Vukičević, D.: Variable neighborhood search for extremal graphs 23. On the Randic index and the chromatic number. *Discrete Math.* **309**, 4228–4234 (2009)
74. Hertz, A., Plumettaz, M., Zufferey, N.: Variable space search for graph coloring. *Discrete Appl. Math.* **156**, 2551–2560 (2008)
75. ILOG CPLEX 10.1. User's Manual (2006)
76. Jornsten, K., Lokketangen, A.: Tabu search for weighted k-cardinality trees. *Asia-Pacific J. Oper. Res.* **14**, 9–26 (1997)
77. Lejeune, M.A.: A variable neighborhood decomposition search method for supply chain management planning problems. *Eur. J. Oper. Res.* **175**, 959–976 (2006)
78. Leyffer, S., User, manual, for, , *minlp.bb* Technical Report, University of Dundee, UK (1999)
79. Liberti, L., Dražić, M.: Variable neighbourhood search for the global optimization of constrained NLPs. In: Proceedings of GO Workshop, Almeria, Spain (2005)
80. Liberti, L., Nannicini, G., Mladenović, N.: A good recipe for solving MINLPs. In: Maniezzo, V., Stuetze, T., Voss, S. (eds.) *MATHEURISTICS: Hybridizing metaheuristics and mathematical programming*, Operations Research/Computer Science Interface Series. Springer, Berlin (2008)
81. Melián, B., Mladenović, N. (eds.) *IMA J. Manage. Math.* **18**, 99–100 (2007)
82. Mladenović, N.: A variable neighborhood algorithm—a new metaheuristic for combinatorial optimization. Abstracts of papers presented at Optimization Days, Montréal, p. 112 (1995)
83. Mladenović, N.: Formulation space search—a new approach to optimization (plenary talk). In: Vučeta, J. (ed.) *Proceedings of XXXII SYMOPIS'05*, pp. 3–5 Vrnjacka Banja, Serbia (2005)
84. Mladenović, N., Dražić, M., Kovačević-Vujčić, V., Čangalović, M.: General variable neighborhood search for the continuous optimization. *Eur. J. Oper. Res.* **191**, 753–770 (2008)
85. Mladenović, N., Hansen, P.: Variable neighborhood search. *Comput. Oper. Res.* **24**, 1097–1100 (1997)
86. Mladenović, N., Petrović, J., Kovačević-Vujčić, V., Čangalović, M.: Solving spread spectrum radar polyphase code design problem by tabu search and variable neighborhood search. *Eur. J. Oper. Res.* **151**, 389–399 (2003)
87. Mladenović, N., Plastria, F., Urošević, D.: Reformulation descent applied to circle packing problems. *Comput. Oper. Res.* **32**, 2419–2434 (2005)
88. Mladenović, N., Plastria, F., Urošević, D.: Formulation space search for circle packing problems. *Lect. Notes Comput. Sci.* **4638**, 212–216 (2007)
89. Moreno-Vega, J.M., Melián, B.: Introduction to the special issue on variable neighborhood search. *J. Heuristics* **14**, 403–404 (2008)
90. Plastria, F., Mladenović, N., Urošević, D.: Variable neighborhood formulation space search for circle packing. 18th Mini Euro Conference VNS., Tenerife, Spain (2005)

91. Popper K.: The Logic of Scientific Discovery. London, Hutchinson (1959)
92. Ribeiro, C.C., de, Souza, M.C.: Variable neighborhood search for the degree-constrained minimum spanning tree problem. *Discrete Appl. Math.* **118**, 43–54 (2002)
93. Ribeiro, C.C., Uchoa, E., Werneck, R.: A hybrid GRASP with perturbations for the Steiner problem in graphs. *INFORMS J. Comput.* **14**, 228–246 (2002)
94. Sedlar, J., Vukicevic, D., Aouchiche, M., Hansen, P.: Variable neighborhood search for extremal graphs 24. Conjectures and results about the clique number. *Les Cahiers du GERAD G-2007-33* (2007)
95. Sedlar, J., Vukicevic, D., Aouchiche, M., Hansen, P.: Variable neighborhood search for extremal graphs 25. Products of connectivity and distance measures. *Les Cahiers du GERAD G-2007-47* (2007)
96. Stevanovic, D., Aouchiche, M., Hansen, P.: On the spectral radius of graphs with a given domination number. *Linear Algebra Appl.* **428**, 1854–1864 (2008)
97. Toksari, A.D., Güner, E.: Solving the unconstrained optimization problem by a variable neighborhood search. *J. Math. Anal. Appl.* **328**, 1178–1187 (2007)
98. Whitaker, R.: A fast algorithm for the greedy interchange of large-scale clustering and median location problems. *INFOR* **21**, 95–108 (1983)
99. Zhang, C., Lin, Z., Lin, Z.: Variable neighborhood search with permutation distance for QAP. *Lect. Notes Comput. Sci.* **3684**, 81–88 (2005)

# Chapter 4

## Scatter Search and Path-Re-linking: Fundamentals, Advances, and Applications

Mauricio G.C. Resende, Celso C. Ribeiro, Fred Glover and Rafael Martí

**Abstract** Scatter search is an evolutionary metaheuristic that explores solution spaces by evolving a set of reference points, operating on a small set of solutions while making only limited use of randomization. We give a comprehensive description of the elements and methods that make up its template, including the most recent elements incorporated in successful applications in both global and combinatorial optimization. Path-relinking is an intensification strategy to explore trajectories connecting elite solutions obtained by heuristic methods such as scatter search, tabu search, and GRASP. We describe its mechanics, implementation issues, randomization, the use of pools of high-quality solutions to hybridize path-relinking with other heuristic methods, and evolutionary path-relinking. We also describe the hybridization of path-relinking with genetic algorithms to implement a progressive crossover operator. Some successful applications of scatter search and of path-relinking are also reported.

---

Mauricio G.C. Resende  
Algorithms and Optimization Research Department, AT&T Labs Research, Florham Park, NJ 07932 USA  
e-mail: mgcr@research.att.com

Celso C. Ribeiro  
Computer Science Department, Universidade Federal Fluminense, Niterói, RJ 22410-240 Brazil  
e-mail: celso@ic.uff.br

Fred Glover  
University of Colorado and OptTek Systems, Inc., Boulder, CO 80302 USA  
e-mail: glover@colorado.edu

Rafael Martí  
Departamento de Estadística e Investigación Operativa, Universidad de Valencia, Spain  
e-mail: rafael.marti@uv.es

## 4.1 Introduction

Scatter search (SS) is a metaheuristic that explores solution spaces by evolving a set of reference points. It can be viewed as an evolutionary method that operates on a small set of solutions and makes only limited use of randomization as a proxy for diversification when searching for a globally optimal solution. The scatter search framework is flexible, allowing the development of alternative implementations with varying degrees of sophistication.

The fundamental concepts and principles of the method were first proposed in the 1970s [5], based on formulations dating back to the 1960s for combining decision rules and problem constraints. In contrast to other evolutionary methods like genetic algorithms, scatter search is founded on the premise that systematic designs and methods for creating new solutions afford significant benefits beyond those derived from recourse to randomization. It uses strategies for search diversification and intensification that have proved effective in a variety of settings.

Scatter search orients its explorations systematically relative to a set of reference points that typically consist of good solutions obtained by prior problem-solving efforts. The criteria for “good” are not restricted to objective function values and may apply to sub-collections of solutions rather than to a single solution, as in the case of solutions that differ from each other according to certain specifications.

The scatter search template [7] has served as the main reference for most of the scatter search implementations to date. The dispersion patterns created by these designs have been found useful in several application areas. Section 4.2 gives a comprehensive description of the elements and methods of this template based on the formulation given in Laguna and Martí [13]. It includes the most recent elements incorporated in successful applications in both global and combinatorial optimization.

Path-relinking is an intensification strategy to explore trajectories connecting elite solutions obtained by heuristic methods [6]. Path-relinking can be considered an extension of the *combination method* of scatter search. Instead of directly producing a new solution when combining two or more original solutions, path-relinking generates paths between and beyond the selected solutions in the neighborhood space. It should be noted that the combination method in scatter search is a problem-dependent element, which is customized depending on the problem and the solution representation. In particular, in global optimization, where solutions are represented as real vectors, most scatter search applications perform linear combinations between pairs of solutions. Alternatively, in problems where solutions are represented as permutations, such as ordering problems, voting methods have been widely applied. In problems where solutions are represented as binary vectors, such as knapsack problems, probabilistic scores have provided very good results [13]. This way, one can also view path-relinking as a unified combination method for all types of problems and in this way it also generalizes the combination methods. In Section 4.3, we focus on path-relinking, including its mechanics, implementation issues, randomization, the use of pools of high-quality solutions to hybridize path-relinking with other heuristic methods, and evolutionary path-relinking.

Concluding remarks are made in Section 4.4, where some successful applications of scatter search and of path-relinking are listed.

## 4.2 Scatter Search

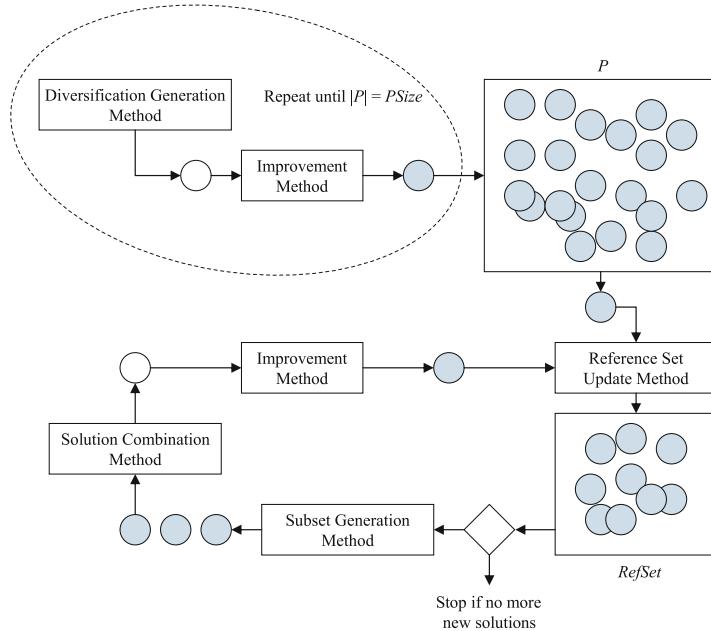
From an algorithmic point of view we can consider that scatter search basically performs iterations over a set of good solutions called the Reference Set (*RefSet*). It must be noted that the meaning of good is not restricted here to the quality of the solutions, but also considers the diversity that they add to this set of solutions.

Once the initial *RefSet* is created, a global iteration of the method consists of three steps: combine, improve, and update the solutions in the *RefSet*. We first describe the five elements in the template. Next, we explain how they interact.

1. A *Diversification Generation Method* to generate a collection of diverse trial solutions, using one or more arbitrary trial solutions (or seed solutions) as an input.
2. An *Improvement Method* to transform a trial solution into one or more enhanced trial solutions: neither the input nor the output solutions are required to be feasible, though the output solutions are typically feasible. If the input trial solution is not improved as a result of the application of this method, the “enhanced” solution is considered to be the same as the input solution.
3. A *Reference Set Update Method* to build and maintain a reference set consisting of the  $b$  “best” solutions found (where the value of  $b$  is typically small, e.g., no more than 20), organized to provide efficient access by other parts of the solution procedure. Several alternative criteria may be used to add solutions to the reference set and delete solutions from the reference set.
4. A *Subset Generation Method* to operate on the reference set, to produce a subset of its solutions as a basis for creating combined solutions. The most common subset generation method is to generate all pairs of reference solutions (i.e., all subsets of size 2).
5. A *Solution Combination Method* to transform a given subset of solutions produced by the Subset Generation Method into one or more combined solutions.

Figure 4.1 shows the interaction among these five methods and highlights the central role of the reference set. This basic design starts with the creation of an initial set of solutions  $P$  and then extracts from it the reference set (*RefSet*) of solutions. The darker circles represent improved solutions resulting from the application of the Improvement Method.

The Diversification Generation Method is used to build a large set  $P$  of diverse solutions. The size of  $P$  (*PSize*) is typically at least ten times the size of *RefSet*. The initial reference set is built according to the Reference Set Update Method. For example, the Reference Set Update Method could consist of selecting  $b$  distinct and maximally diverse solutions from  $P$ .



**Fig. 4.1** Scatter search diagram.

A typical construction of the initial reference set starts with the selection of the best  $b/2$  solutions from  $P$ . These solutions are added to  $RefSet$  and deleted from  $P$ . For each solution in  $P$  but not in  $RefSet$ , the minimum of the distances to the solutions in  $RefSet$  is computed. Then, the solution with the maximum of these minimum distances is selected. This solution is added to  $RefSet$  and deleted from  $P$  and the minimum distances are updated. (In applying this max–min criterion, or any criterion based on distances, it can be important to scale the problem variables to avoid a situation where a particular variable or subset of variables dominate the distance measure and distort the appropriate contribution of the vector components.) The process is repeated  $b/2$  times. The resulting reference set has  $b/2$  high-quality solutions and  $b/2$  highly diverse solutions. Note that with this criterion we are considering as equally important quality and diversity in the original  $RefSet$ . Alternative designs may include a different composition of the  $b$  solutions in this set. For example, we could consider just a single solution selected because of its quality (say the best one in  $P$ ) and the remaining  $b - 1$  solutions in the  $RefSet$  could be selected from  $P$  because of their diversity. Since the reference set is the heart of a scatter search procedure, its initial composition may result in significant changes during the search process.

The solutions in  $RefSet$  are ordered according to quality, where the best solution is the first one in the list. The search is then initiated applying the Subset Generation Method. In its simplest (and typical) form it consists of generating all pairs of reference solutions. That is, the method would focus on subsets of size 2 resulting

in  $(b^2 - b)/2$  new subsets. The pairs are selected one at a time in lexicographical order and the Solution Combination Method is applied to generate one or more trial solutions. These trial solutions are subjected to the Improvement Method, if one is available. The Reference Set Update Method is applied once again to build the new *RefSet* with the best solutions, according to the objective function value, from the current *RefSet* and the set of trial solutions. A global iteration finishes with the update of the *RefSet*. Note that in subsequent iterations we only combine the pairs of solutions not combined in previous iterations. The basic procedure terminates after all the generated subsets are subjected to the Combination Method and none of the improved trial solutions are admitted to *RefSet* under the rules of the Reference Set Update Method. However, in advanced scatter search designs, the *RefSet* rebuilding is applied at this point and the best  $b/2$  solutions are kept in the *RefSet* and the other  $b/2$  are selected from  $P$ , replacing the worst  $b/2$  solutions.

It is interesting to observe similarities and contrasts between scatter search and the original Genetic Algorithm (GA) proposals. Both are instances of what are sometimes called population-based or evolutionary approaches. Both incorporate the idea that a key aspect of producing new elements is to generate some form of combination of existing elements. However, original GA approaches were predicated on the idea of choosing parents randomly to produce offspring and further on introducing randomization to determine which components of the parents should be combined. By contrast, scatter search is based on deterministic designs in which we implement strategic rules to generate new solutions. These rules do not resort to randomization, as usually happens in GAs. They are based on the structure and properties of the problem being solved, as well as on the search history. Moreover, GAs usually apply general-purpose combination methods, such as the well-known crossover operator, while scatter search customizes the combination method for each particular problem. It should be noted, however, that GAs have been progressively incorporating more advanced design elements from more powerful meta-heuristics and solution strategies.

#### 4.2.1 New Strategies in Global Optimization

Egea et al. [2] proposed an evolutionary method for global optimization of complex-process models, which employs some elements of scatter search and path-relinking. Regarding scatter search, the method uses a relatively small population size, partially chosen by a quality criterion from an initial set of diverse solutions. It also performs systematic combinations among the population members. Regarding path-relinking, the new solutions are generated within the areas defined by every pair of solutions in the population, introducing a bias to generate new solutions which share more properties with the best population members than with the rest. We mentioned this method here because it introduces new strategies and modifies some standard scatter search designs. Specifically, it employs the following:

- a small population without memory structures, in which repeated sampling is allowed;
- a new combination method based on wide hyper-rectangles;
- an aggressive population update for a quick convergence;
- a new search intensification strategy called *the go-beyond*.

Considering its potential applicability to other domains, we describe the *go-beyond* strategy, which consists in exploiting promising directions, extending the combination method.

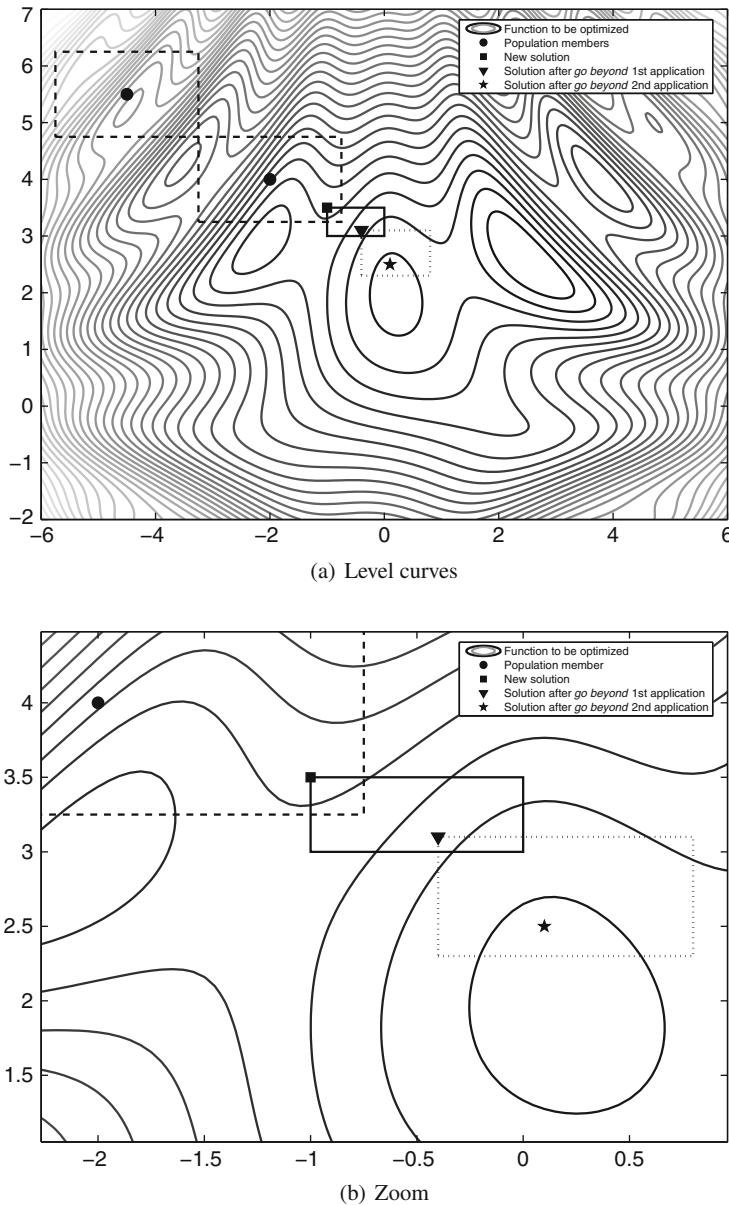
Figure 4.2 depicts the level curves (contour plots) of the 2D dimensional unconstrained function  $f(x_1, x_2)$  in the range  $x_1 \in [-6, 6]$ ,  $x_2 \in [-2, 7]$ , which presents several minima:

$$f(x_1, x_2) = 2 + 0.01 (x_2 - x_1^2)^2 + (1 - x_1)^2 + 2(2 - x_2)^2 + 7 \sin(0.5x_1) \sin(0.7x_1 x_2).$$

We illustrate in this diagram how the *go-beyond* strategy works. From a pair of *RefSet* solutions  $x$  and  $y$  (labeled as *population members* in the figure and depicted with black points) a new solution is generated in the corresponding hyper-rectangle,  $z$ , and depicted in the figure (labeled as *new solution* and represented with a black square). If  $z$  is better than  $x$  and  $y$  ( $f(z) < f(x)$  and  $f(z) < f(y)$ ), then we consider that this is a promising direction and apply the *go-beyond* strategy, *extending* the combination method. In the present problem, this means that we consider a new hyper-rectangle (solid line) defined by the distance between  $z$  and  $y$  (its closest reference set solution). A new solution (depicted with a triangle) is created in this hyper-rectangle and the process is repeated as long as good solutions are obtained. Figure 4.2 shows a new solution (starred) created in an area very close to the global minimum.

#### 4.2.2 New Strategies in Combinatorial Optimization

Martí et al. [15] proposed a scatter search algorithm for the well-known max-cut problem based on the standard design described in this section. Their method extends the basic scatter search implementation in three different ways. First, it uses a new selection procedure for constructing a reference set from a population of solutions. Traditionally, scatter search implementations have used the criterion of maximizing the minimum distance between the solution under consideration and the solutions already in the reference set. In such a process, diverse solutions are selected one by one from the population  $P$  and the distances are updated after each selection. In contrast, Martí et al. [15] propose a method that selects all the diverse solutions at once by solving the *maximum diversity problem* (MDP). Given a set of elements  $\mathcal{S}$  and the corresponding distances between the elements of the set, the MDP consists in finding the most diverse subset of  $\mathcal{S}$  of a specified size. The diversity of the chosen subset is given by the sum of the distances between each pair of its elements. The distance between two max-cut solutions is defined to be the number of different edges in the cut.



**Fig. 4.2** The *go-beyond* strategy.

The use of the MDP within scatter search is based on recognizing that the original set of elements is given by  $P \setminus \{\text{the } b/2 \text{ best solutions}\}$ . The MDP scheme is also used to complete the current *RefSet*, which is already partially populated with the  $b/2$  best solutions from  $P$ .

The second extension consists of a dynamic adjustment of the depth parameter  $k$  associated with the ejection chain mechanism, which is at the core of the search-based improvement method. This local search has an associated parameter that measures the depth of the search in the ejection chain process. The solution representation incorporates the information related to the particular  $k$  value used to generate it. In this way, the depth of the ejection chain produced depends on the parameter values associated with the solutions being combined.

The third extension implements a probabilistic selection of the combination methods. The probability of selecting one of three methods proposed in [18] for the max-cut problem is proportional to the number of high-quality solutions generated by the method in previous iterations. A probability-based mechanism is introduced to select a combination method each time the solutions are combined. The probability of selecting one of the three methods is set to 1/3 at the beginning of the search. The probability values are then updated at the end of each SS iteration in order to favor the combination methods that produce solutions of sufficiently high quality to be included in the reference set.

## 4.3 Path-Relinking

Path-relinking was originally proposed by Glover [6] as an intensification strategy to explore trajectories connecting elite solutions obtained by tabu search or scatter search [8–10]. In the remainder of this chapter, we focus on path-relinking, including its mechanics, implementation issues, randomization, the use of pools of high-quality solutions to hybridize path-relinking with other heuristic methods, and evolutionary path-relinking. We conclude the chapter with some computational results illustrating the effect of using path-relinking with other heuristics. For completeness, we have included in this section some material that also appears in the chapter of the handbook on GRASP.

### 4.3.1 Mechanics of Path-Relinking

We consider an undirected graph  $G = (S, M)$  associated with the solution space, where the nodes in  $S$  correspond to feasible solutions and the edges in  $M$  correspond to moves in the neighborhood structure, i.e.,  $(i, j) \in M$  if and only if  $i \in S$ ,  $j \in S$ ,  $j \in N(i)$ , and  $i \in N(j)$ , where  $N(s)$  denotes the neighborhood of a solution  $s \in S$ . Path-relinking is usually carried out between two solutions: one is called the *initial solution*, while the other is the *guiding solution*. One or more paths in the

solution space graph connecting these solutions are explored in the search for better solutions. Local search is applied to the best solution in each of these paths, since there is no guarantee that this solution is locally optimal.

Let  $s \in S$  be a node on the path between an initial solution and a guiding solution  $g \in S$ . Not all solutions in the neighborhood  $N(s)$  are allowed to follow  $s$  on the path from  $s$  to  $g$ . We restrict the choice to those solutions in  $N(s)$  that are more similar to  $g$  than  $s$  is. This is accomplished by selecting moves from  $s$  that introduce attributes contained in the guiding solution  $g$ . Therefore, path-relinking may be viewed as a strategy that seeks to incorporate attributes of high-quality solutions (i.e., the guiding solutions), by favoring these attributes in the selected moves. After an analysis of each potential move, the most common strategy is to select a move that results in the best quality restricted neighbor of  $s$ . The restricted neighbors of  $s$  are all solutions in the neighborhood of  $s$  that incorporate an attribute of the guiding solution not present in  $s$ .

Several alternatives for path-relinking have been considered and combined in recent implementations. These include forward, backward, back-and-forward, mixed, truncated, greedy randomized adaptive, and evolutionary path-relinking. All these alternatives involve trade-offs between computation time and solution quality.

Suppose that path-relinking is applied to a minimization problem between solutions  $x_1$  and  $x_2$  such that  $z(x_1) \leq z(x_2)$ , where  $z(\cdot)$  denotes the objective function. In *forward* path-relinking, the initial and guiding solutions are set to  $g = x_1$  and  $s = x_2$ . Conversely, in *backward* path-relinking, we set  $g = x_2$  and  $s = x_1$ . In *back-and-forward* path-relinking, backward path-relinking is applied first followed by forward path-relinking. Path-relinking explores the neighborhood of the initial solution more thoroughly than the neighborhood of the guiding solution because, as it moves along the path, the size of the restricted neighborhood decreases. Consequently, backward path-relinking tends to do better than forward path-relinking. Back-and-forward path-relinking does at least as well as either backward or forward path-relinking but takes about twice as long to compute.

In applying *mixed path-relinking* [11, 21] between feasible solutions  $s$  and  $t$  in  $S$ , two paths are started simultaneously, one at  $s$  and the other at  $t$ . These two paths meet at some solution  $r \in S$ , thus connecting  $s$  and  $t$  with a single path. Algorithm 1 describes a mixed path-relinking procedure for a 0-1 minimization problem, such as the set covering problem, where  $x^s$  and  $x^t$  are binary vectors representing the solutions to be linked.

The set  $\Delta = \left\{ j = 1, \dots, n : x_j^s \neq x_j^t \right\}$  of positions in which  $x^s$  and  $x^t$  differ is computed in line 2. The cardinality of this set is called the *Hamming distance* between  $x^s$  and  $x^t$ . The best solution,  $x^*$ , among  $x^t$  and  $x^s$  and its cost,  $z^* = z(x^*)$ , are determined in lines 3 and 4, respectively. The current path-relinking solution,  $x$ , is initialized to  $x^s$  in line 5. The loop in lines 6 to 16 progressively determines the next solution in the path connecting  $x^s$  and  $x^t$ , until the entire path is traversed. For every position  $\ell \in \Delta$ , we define  $x \oplus \ell$  to be the solution obtained from  $x$  by complementing the current value of  $x_\ell$ . Line 7 determines the component  $\ell^*$  of  $\Delta$  for which  $x \oplus \ell$  results in the least cost solution. This component is removed from  $\Delta$  in line 8 and the current solution is updated in line 9 by complementing the value of its  $\ell^*$ th position.

---

**Algorithm 1** Mixed path-relinking procedure for problems where solutions are represented by binary vectors

---

```

1 MixedPathRelinking
2  $\Delta \leftarrow \{j = 1, \dots, n : x_j^s \neq x_j^t\}$ ;
3  $x^* \leftarrow \operatorname{argmin}\{z(x^s), z(x^t)\}$ ;
4  $z^* \leftarrow \min\{z(x^s), z(x^t)\}$ ;
5  $x \leftarrow x^*$ ;
6 while  $|\Delta| > 1$  do
7    $\ell^* \leftarrow \operatorname{argmin}\{z(x \oplus \ell) : \ell \in \Delta\}$ ;
8    $\Delta \leftarrow \Delta \setminus \{\ell^*\}$ ;
9    $x_\ell \leftarrow 1 - x_\ell$ ;
10  if  $z(x) < z^*$  then
11     $x^* \leftarrow x$ ;
12     $z^* \leftarrow z(x)$ ;
13  end
14   $x^s \leftarrow x^t$ ;
15   $x^t \leftarrow x$ ;
16 end
17  $x \leftarrow \operatorname{LocalSearch}(x)$ ;
18 return  $x$  ;

```

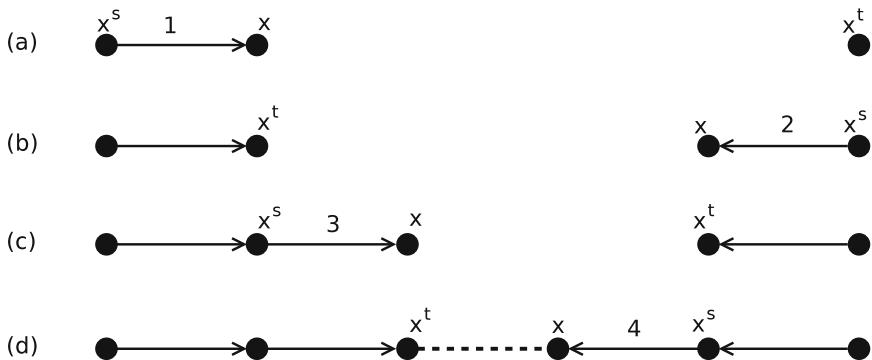
---

If the test in line 10 detects that the new current solution  $x$  improves the best solution  $x^*$  in the path, then  $x^*$  and its cost are updated in lines 11 and 12, respectively. The roles of the starting and target solutions are swapped in lines 14 and 15 to implement the mixed path-relinking strategy. If  $|\Delta| = 0$ , then the local search is applied to the best solution in the path in line 16 and the locally optimal solution is returned by the procedure.

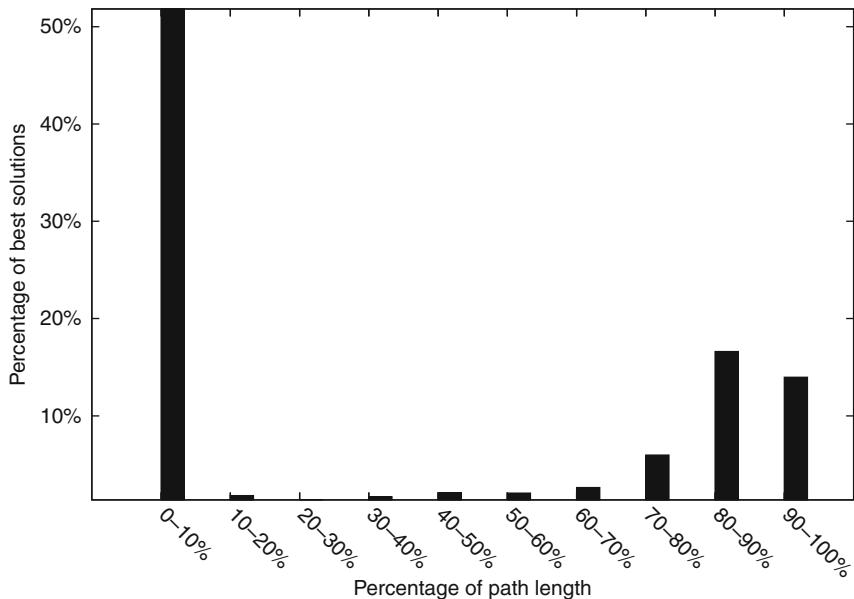
Like back-and-forward path-relinking, the mixed variant explores both neighborhoods  $N(x^s)$  and  $N(x^t)$ . Unlike back-and-forward path-relinking, it is usually less than twice as long as the backward or forward variants.

In the case of the set covering problem, there always exists a path connecting  $x^s$  and  $x^t$ . We just need to observe that setting to one all components with value 0 in  $x^s$  and value 1 in  $x^t$  results in a series of feasible covers leading from  $x^s$  to some feasible solution  $x$ . Next, by setting to 0 those components with value 1 in  $x$  and value 0 in  $x^t$  result again in a series of feasible covers leading from  $x$  to  $x^t$ . Figure 4.3 illustrates the application of mixed path-relinking to solutions  $x^s$  and  $x^t$  for which the Hamming distance is equal to 5.

One can expect to see most solutions produced by path-relinking to come from subpaths close to either the initiating or guiding solutions. Resende et al. [18] showed that this occurs in instances of the max–min diversity problem. In that experiment, a back and forward path-relinking scheme was tested. Figure 4.4 shows the percentage of best solutions found by path-relinking taken over several instances and several applications of path-relinking. The 0–10% range in the figure corresponds to subpaths near the initial solutions for the forward path-relinking phase as well as the backward phase, while the 90–100% range are subpaths near the guiding solutions. As the figure indicates, exploring the subpaths near the extremities may



**Fig. 4.3** Mixed path-relinking between two solutions with Hamming distance of 5: numbers above the arrows represent the order in which the moves are performed.



**Fig. 4.4** Percentage of best solutions found at different depths of the path from the initial solution to the guiding solution on instances of the max-min diversity problem.

produce solutions about as good as those found by exploring the entire path. There is a higher concentration of better solutions close to the initial solutions explored by path-relinking.

As shown in Algorithm 2, it is simple to adapt path-relinking to explore only the neighborhoods close to the extremes. Let  $\rho$ ,  $0 < \rho \leq 1$ , be a real parameter that defines the portion of the path to be explored. Instead of carrying out the main loop while  $|\Delta| > 1$  as in the mixed path-relinking of Algorithm 1, the main loop is applied while  $|\Delta| > \rho \cdot \delta_t$ , where  $\delta_t$  is the cardinality of the initial set  $\Delta$ .

---

**Algorithm 2** Truncated mixed path-relinking procedure for problems where solutions are represented by binary vectors

---

```

1 TruncatedMixedPathRelinking
2  $\Delta \leftarrow \{j = 1, \dots, n : x_j^s \neq x_j^t\};$ 
3  $\delta_t \leftarrow |\Delta|;$ 
4  $x^* \leftarrow \operatorname{argmin}\{z(x^s), z(x^t)\};$ 
5  $z^* \leftarrow \min\{z(x^s), z(x^t)\};$ 
6  $x \leftarrow x^s;$ 
7 while  $|\Delta| > \rho \cdot \delta_t$  do
8    $\ell^* \leftarrow \operatorname{argmin}\{z(x \oplus \ell) : \ell \in \Delta\};$ 
9    $\Delta \leftarrow \Delta \setminus \{\ell^*\};$ 
10    $x_\ell \leftarrow 1 - x_\ell;$ 
11   if  $z(x) < z^*$  then
12      $x^* \leftarrow x;$ 
13      $z^* \leftarrow z(x);$ 
14   end
15    $x^s \leftarrow x^t;$ 
16    $x^t \leftarrow x;$ 
17 end
18  $x \leftarrow \text{LocalSearch}(x);$ 
19 return  $x;$ 

```

---

### 4.3.2 Minimum Distance Required for Path-Relinking

We assume that we want to connect solutions  $s$  and  $t$  with path-relinking. If the distance  $|\Delta(s, t)|$  between  $s$  and  $t$ , i.e., the number of components in which  $s$  and  $t$  differs, is equal to 1, then the path directly connects the two solutions and no solution, other than  $s$  and  $t$ , is visited.

If we assume that  $s$  and  $t$  are both locally optimal, we know that  $z(s) \leq z(r)$  for all  $r \in N(s)$  and  $z(t) \leq z(r)$  for all  $r \in N(t)$ . If  $|\Delta(s, t)| = 2$ , then any path is of the type  $s \rightarrow r \rightarrow t$ , where  $r \in N(s) \cap N(t)$ , and consequently  $r$  cannot be better than either  $s$  or  $t$ . Likewise, if  $|\Delta(s, t)| = 3$ , then any path is of the type  $s \rightarrow r_s \rightarrow r_t \rightarrow t$ , where  $r_s \in N(s)$  and  $r_t \in N(t)$ , and consequently neither  $r_s$  nor  $r_t$  can be better than both  $s$  and  $t$ .

Therefore, things only get interesting for  $|\Delta(s, t)| > 3$ . For those cases, any path is of the type  $s \rightarrow r_s \rightarrow w_1 \rightarrow \dots \rightarrow w_p \rightarrow r_t \rightarrow t$ , where  $w_1, \dots, w_p$  are candidates to be better than both  $s$  and  $t$ . Therefore, we do not consider relinking a pair of solutions  $s, t$  unless  $|\Delta(s, t)| \geq 4$ .

### 4.3.3 Randomization in Path-Relinking

Consider again a problem whose solution can be represented as a binary vector of size  $n$ , such as the set covering problem, the satisfiability problem, or the max-cut

problem. Let us denote the set of solutions spanned by the common elements of solutions  $s$  and  $t$  as

$$\mathcal{S}(s, t) := \{w \in \{0, 1\}^n : w_i = s_i = t_i, i \notin \Delta(s, t)\} \setminus \{s, t\}, \quad (4.1)$$

with  $|\mathcal{S}(s, t)| = 2^{|\Delta(s, t)|} - 2$ . The underlying assumption of path-relinking is that there exist good quality solutions in  $\mathcal{S}(s, t)$ , since this space consists of all solutions which contain the common elements of two good solutions  $s$  and  $t$ . Taking into consideration that the size of this space is exponentially large, we normally adopt a greedy search where a path of solutions

$$s = w_0, w_1, \dots, w_{|\Delta(s, t)|} = t,$$

is constructed, such that  $|\Delta(w_i, w_{i+1})| = 1$ ,  $i = 0, \dots, |\Delta(s, t)| - 1$ , and the best solution from this path is chosen. However, by adopting the greedy strategy, we limit ourselves to exploring a single path from a set of exponentially many paths. By adding randomization to path-relinking, *greedy randomized adaptive path-relinking* (GRAPR) [3] is not constrained to explore a single path.

The pseudo-code for GRAPR for a minimization problem is shown in Algorithm 3. The main difference with respect to Algorithm 1 are lines 6 and 8–11. Instead of selecting the move that results in the best solution as is the case in standard path-relinking, a restricted candidate list (RCL) is constructed with the moves

---

**Algorithm 3** Greedy randomized adaptive path-relinking with a mixed variant of path-relinking

---

```

1 GreedyRandomizedAdaptivePathRelinking
2  $\Delta \leftarrow \{j = 1, \dots, n : x_j^s \neq x_j^t\}$ ;
3  $x^* \leftarrow \operatorname{argmin}\{z(x^s), z(x^t)\}$ ;
4  $z^* \leftarrow \min\{z(x^s), z(x^t)\}$ ;
5  $x \leftarrow x^s$ ;
6 Select  $\alpha \in [0, 1] \subset \mathbb{R}$  at random;
7 while  $|\Delta| > 1$  do
8    $z^- \leftarrow \min\{z(x \oplus \ell) : \ell \in \Delta\}$ ;
9    $z^+ \leftarrow \max\{z(x \oplus \ell) : \ell \in \Delta\}$ ;
10   $RCL \leftarrow \{\ell \in \Delta : z(x \oplus \ell) \leq z^- + \alpha(z^+ - z^-)\}$ ;
11  Select  $\ell^* \in RCL$  at random;
12   $\Delta \leftarrow \Delta \setminus \{\ell^*\}$ ;
13   $x_\ell \leftarrow 1 - x_\ell$ ;
14  if  $z(x) < z^*$  then
15     $x^* \leftarrow x$ ;
16     $z^* \leftarrow z(x)$ ;
17  end
18   $x^s \leftarrow x^t$ ;
19   $x^t \leftarrow x$ ;
20 end
21  $x \leftarrow \text{LocalSearch}(x)$ ;
22 return  $x$ ;

```

---

that result in solutions with costs in an interval that depends on the value of the best move, the value of the worst move, and a random parameter  $\alpha$ . From this set, one move is selected at random to produce the next step in the path.

GRAPR is useful when path-relinking is applied more than once between the same pair of solutions as it can occur in evolutionary path-relinking (discussed in Section 4.3.5).

#### 4.3.4 Hybridization with a Pool of Elite Solutions

Path-relinking is a major enhancement to metaheuristics that generate a sequence of locally optimal feasible solutions. These metaheuristics include, but are not limited to, GRASP, variable neighborhood search, tabu search, scatter search, and simulated annealing. To hybridize path-relinking with these metaheuristics, one usually makes use of an *elite set*, i.e., a diverse pool of high-quality solutions found during the search. The elite set starts empty and is limited in size. Each locally optimal solution produced by the metaheuristic is relinked with one or more solutions from the elite set. Each solution produced by path-relinking is a candidate for inclusion in the elite set where it can replace an elite solution of worse value.

The pool of elite solutions is initially empty. Each locally optimal solution produced by the metaheuristic and each solution resulting from path-relinking is considered as a candidate to be inserted into the pool. If the pool is not yet full, the candidate is simply added to the pool if it differs from all pool members. If the pool is full and the candidate is better than the incumbent, then it replaces an element of the pool. In case the candidate is better than the worst element of the pool but not better than the best element, then it replaces some element of the pool if it is sufficiently different from every other solution currently in the pool. To balance the impact on pool quality and diversity, the element selected to be replaced is the one that is most similar to the entering solution among those elite solutions of quality no better than the entering solution [20].

Given a local optimum  $s_1$  produced by the metaheuristic, we need to select at random from the pool a solution  $s_2$  to be connected with  $s_1$  via path-relinking. In principle, any solution in the pool could be selected. However, one should avoid solutions that are too similar to  $s_1$ , because relinking two solutions that are similar limits the scope of the path-relinking search. If the solutions are represented by binary vectors, one should favor pairs of solutions for which the Hamming distance between them is high. A strategy introduced in [20] is to select a pool element at random with probability proportional to the Hamming distance between the pool element and the local optimum  $s_1$ . Since the number of paths between two solutions grows exponentially with their Hamming distance, this strategy favors pool elements that have a large number of paths connecting them to and from  $s_1$ .

Algorithm 4 illustrates the pseudo-code of a hybrid heuristic that uses path-relinking for minimization. In line 2, the pool of elite solutions  $P$  is initially empty. The loop in lines 3 to 12 makes up an iteration of the hybrid algorithm. In line 4,  $x$  is

---

**Algorithm 4** Hybridization of path-relinking with a heuristic that generates local optima

---

```

1 HEUR+PR
2 Initialize elite set  $P \leftarrow \emptyset$ ;
3 while stopping criterion not satisfied do
4    $x \leftarrow \text{HeuristicLocalOptimal}()$ ;
5   if  $P = \emptyset$  then insert  $x$  into  $P$ ;
6   else
7      $x^s \leftarrow x$ ;
8     Choose, at random, a pool solution  $x^t \in P$ ;
9      $x \leftarrow \text{PathRelinking}(x^s, x^t)$ ;
10    Update the elite set  $P$  with  $x$  ;
11  end
12 end
13 return  $P$  ;

```

---

a locally optimal solution generated by procedure `HeuristicLocalOptimal()`. If the elite set is empty, then  $x$  is inserted into the pool in line 5. Otherwise,  $x$  becomes the initiating solution in lines 7 and a guiding solution is selected at random from the pool in line 8. The initiating and guiding solutions are relinked in line 9 and the resulting solution is tested for inclusion into the elite set in line 10. The hybrid procedure returns the set of elite solutions which includes the best solution found during the search.

#### 4.3.5 Evolutionary Path-Relinking

Path-relinking can also be applied between elite set solutions to search for new high-quality solutions and to improve the average quality of the elite set. This can be done in a post-optimization phase, after the main heuristic stops, or periodically, when the main heuristic is still being applied [1, 18, 20].

We describe two schemes called *evolutionary path-relinking* for this purpose. Both schemes take as input the elite set and return either the same elite set or one with an improved average cost.

The first scheme, described by Resende and Werneck [20], works with a population that evolves over a number of generations. The initial population is the input elite set. In the  $k$ th generation the procedure builds the  $k$ th population, which is initially empty. Path-relinking is applied between all pairs of solutions in population  $k - 1$ . Each solution output from the path-relinking operation is a candidate for inclusion in population  $k$ . The usual rules for inclusion into an elite set are adopted in evolutionary path-relinking. If population  $k$  is not yet full, the solution is accepted if it differs from all solutions in the population. After population  $k$  is full, the solution is accepted if either it is better than the best solution in the population or it is better than the worst and is sufficiently different from all solutions in the population.

Once a solution is accepted for inclusion into population  $k$ , it replaces the solution in population  $k$  that does not have a better cost and that is most similar to it. The procedure halts when the best solution in population  $k$  does not have better cost than the best solution in population  $k - 1$ .

A variation of the above scheme is described by Resende et al. [18]. In that scheme, while there exists a pair of solutions in the elite set for which path-relinking has not yet been applied, the two solutions are combined with path-relinking and the resulting solution is tested for membership in the elite set. If it is accepted, it then replaces the elite solution most similar to it among all solutions having worse cost.

Since some elite solutions may remain in the elite set over several applications of evolutionary path-relinking, greedy randomized adaptive path-relinking [3] can be used in evolutionary path-relinking to avoid repeated explorations of the same paths in the solution space in different applications of the procedure.

GRASP with evolutionary path-relinking and scatter search are evolutionary methods based on evolving a small set of selected solutions (elite set in the former and reference set in the latter). We can, therefore, observe similarities between them. In some implementations of scatter search, GRASP is used to populate the reference set. Note, however, that other constructive methods can be used as well. Similarly, path-relinking can be used to combine solutions in scatter search, but we can use any other combination method. From an algorithmic point of view, we may find two main differences between these methods. The first one is that in scatter search we do not apply path-relinking to the solutions obtained with GRASP, but rather, we only apply path-relinking as a combination method between solutions already in the reference set. The second difference is that in scatter search when none of the new solutions obtained with combinations are admitted to the reference set (elite set), it is rebuilt, removing some of its solutions, as specified in the reference set update method. In GRASP with evolutionary path-relinking we do not remove solutions from the elite set, but rather, we reapply GRASP and use the same rules for inclusion in the elite set.

#### ***4.3.6 Progressive Crossover: Hybridization with Genetic Algorithms***

Path-relinking was first applied in the context of a genetic algorithm by Ribeiro and Vianna [22] in order to implement a progressive crossover operator. In this innovative application, the hybridization strategy was applied to a phylogeny problem.

The original proposal was extended and improved in [23]. In this case, a bidirectional (or back and forward) path-relinking strategy is used: given two parent solutions  $s_1$  and  $s_2$ , one path is computed leading from  $s_1$  to  $s_2$  and another leading from  $s_2$  to  $s_1$ . The best solution along them is returned as the offspring resulting from crossover. This mechanism is an extension of the traditional crossover operator: instead of producing only one offspring, defined by one single combination of two parents, it investigates many solutions that share characteristics of the selected

parents. The solution found by path-relinking corresponds to the best offspring that could be obtained by applying the standard crossover to the parents.

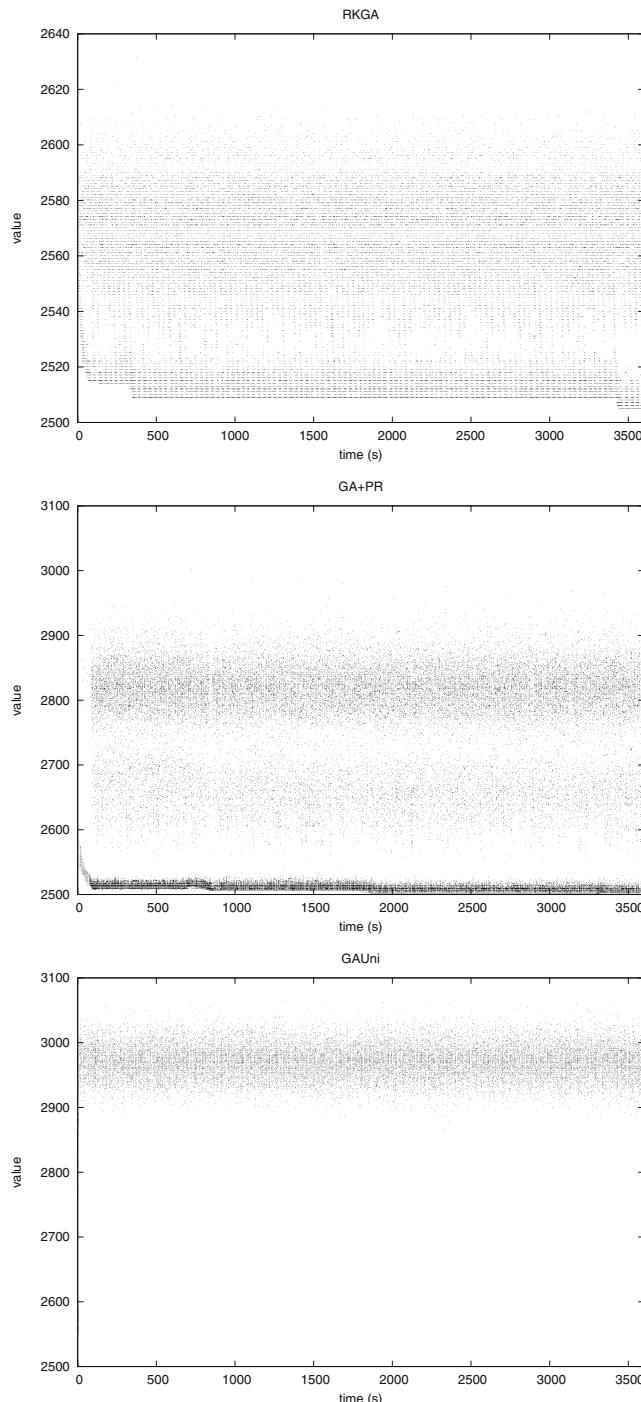
The experiments reported in [23] make use of the results obtained on one randomly generated instance (TST17) of the phylogeny problem to assess the evolution of the solutions found by three different genetic algorithm in 1 h (3600 s) of computations: the random-keys genetic algorithm RKGA [22], the proposed genetic algorithm GA+PR using path-relinking to implement the progressive crossover operator, and the simpler genetic algorithm GAUni using uniform crossover. Figure 4.5 presents the solution value at the end of each generation for each of the 100 individuals in the population. Since the original random-keys genetic algorithm RKGA made use of elitism, the solution values are restricted to a smaller interval ranging between 2500 and 2620. The solution values obtained by the two other algorithms show more variability. The solutions found by algorithm GA+PR are better than those obtained by RKGA and GAUni, illustrating the contribution of the strategy based on path-relinking to implement the crossover operator.

Path-relinking was also applied by Zhang and Lai [25] following the strategy proposed in [22] in the implementation of a genetic algorithm for the multiple-level warehouse layout problem. Their approach also makes use of path-relinking when the genetic algorithm seems to be trapped in a locally optimal solution. Once again, path-relinking was used by Vallada and Ruiz [24] as a progressive crossover operator within a genetic algorithm for the minimum tardiness permutation flow-shop problem. It was also applied as an intensification strategy after a number of generations without improvement to the best solution. The selected individuals are marked in order not to be selected again during the application of path-relinking. Path-relinking was also hybridized with a genetic algorithm as a post-optimization procedure [17]. In this work, the solutions in the final population produced by the genetic algorithm are progressively combined and refined.

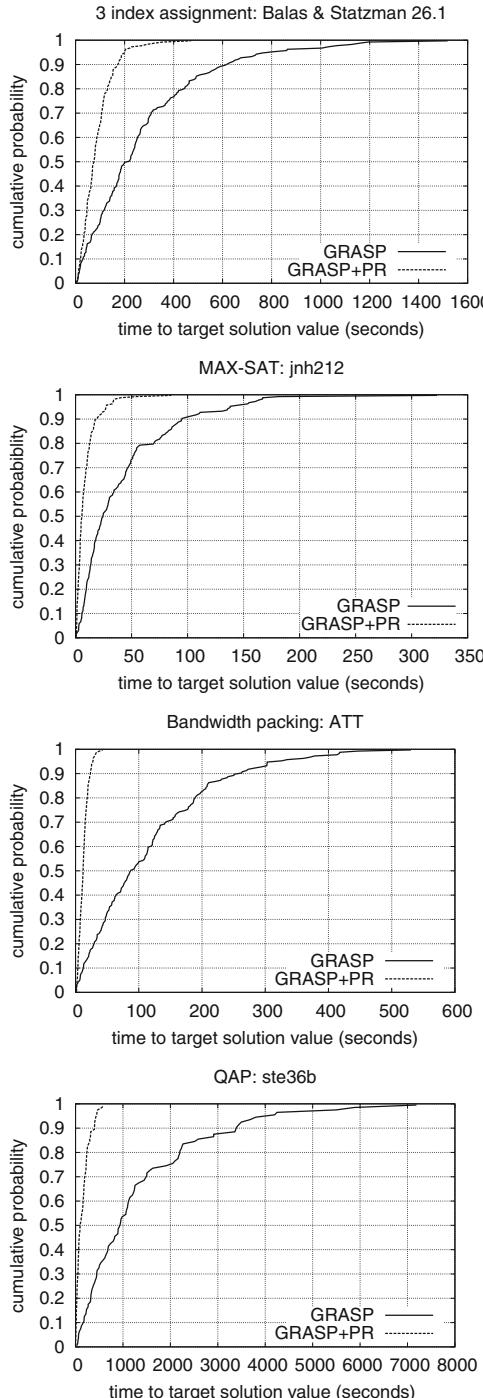
#### 4.3.7 Hybridization of Path-Relinking with Other Heuristics

The basic implementation of GRASP is memoryless because it does not make use of information collected in previous iterations. The use of path-relinking within a GRASP procedure, as an intensification strategy applied to each locally optimal solution, was first proposed by Laguna and Martí [12]. It was followed by several extensions, improvements, and successful applications [19]. Each local minimum produced by the GRASP is combined with a randomly selected elite solution. The resulting solution is a candidate for inclusion into the elite set. Evolutionary path-relinking can be applied periodically to improve the quality of the elite set.

Enhancing GRASP with path-relinking almost always improves the performance of the heuristic. As an illustration, Figure 4.6 shows time-to-target plots for GRASP and GRASP with path-relinking implementations for four different applications. These time-to-target plots show the empirical cumulative probability distributions of the *time-to-target* random variable when using pure GRASP and GRASP



**Fig. 4.5** Solutions obtained by genetic algorithms for random instance TST17 for 3600 s of computations.



**Fig. 4.6** Time-to-target plots comparing running times of pure GRASP and GRASP with path-relinking on four instances of distinct problem types: three index assignment [1], maximum satisfiability [4], bandwidth packing [4], and quadratic assignment [6].

with path-relinking, i.e., the time needed to find a solution at least as good as a prespecified target value. For all problems, the plots show that GRASP with path-relinking is able to find target solutions faster than GRASP.

## 4.4 Applications and Concluding Remarks

There are three main sources where successful applications of scatter search and path-relinking can be found. First, Chapter 8 of the monograph on scatter search by Laguna and Martí [13] identifies 14 applications, including neural networks, multi- and mono-objective routing problems, graph drawing, scheduling, and coloring problems. A second source of successful implementations of both methodologies is a special issue of EJOR [14] in which they are classified into the following seven categories: foundations, nonlinear optimization, optimization in graphs, parallel optimization, prediction and clustering, routing, and scheduling. There is also a third source, which is frequently updated with current applications: the web site <http://www.uv.es/rmarti/scattersearch> on scatter search and path-relinking publications, in which more than 100 implementations are collected.

## References

1. Aiex, R.M., Pardalos, P.M., Resende, M.G.C., Toraldo., G.: GRASP with path-relinking for three-index assignment. *INFORMS J. Comput.*, **17**:224–247 (2005)
2. Egea, J., Martí, R., Banga, J.: An evolutionary method for complex-process optimization. *Comput. Oper. Res.* (2009). doi:10.1016/j.cor.2009.05.003
3. Faria, H., Jr., Binato, S., Resende, M.G.C., Falcão., D.J.: Transmission network design by a greedy randomized adaptive path relinking approach. *IEEE Trans. Power Syst.*, **20**:43–49 (2005)
4. Festa, P., Pardalos, P.M., Pitsoulis, L.S., Resende., M.G.C.: GRASP with path-relinking for the weighted MAXSAT problem. *ACM J. Exp. Algorithms*, **11**:1–16 (2006)
5. Glover., F.: Heuristics for integer programming using surrogate constraints. *Decis. Sci.*, **8**:156–166 (1977)
6. Glover., F.: Tabu search and adaptive memory programing – Advances, applications and challenges. In: Barr, R.S., Helgason, R.V., Kennington, J.L. (eds.) *Interfaces in Computer Science and Operations Research*, 1–75. Kluwer, Boston, MA, USA (1996)
7. Glover, F.: A template for scatter search and path relinking. In Hao, J.-K., Lutton, E., Ronald, E., Schoenauer, M., Snyers, D. (eds.) *Artificial Evolution, Third European Conference, AE97*, Lecture Notes in Computer Science, vol. 1363 pp. 13–54. Springer, Nimes, France (1998)
8. Glover, F.: Multi-start and strategic oscillation methods – principles to exploit adaptive memory. In: Laguna, M., González-Velarde, J.L. (eds.), Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research, pp. 1–24. Kluwer, Boston, MA, USA (2000)
9. Glover, F., Laguna., M.: Tabu Search. Kluwer, Boston, MA, USA (1997)
10. Glover, F., Laguna, M., Martí, R.: Fundamentals of scatter search and path relinking. *Cont. Cybern.*, **39**:653–684 (2000)

11. Glover, F., Laguna, M., Martí., R.: Scatter search and path relinking: foundations and advanced designs. In: Godfrey, C. O., Babu, B.V. (eds.), New Optimization Techniques in Engineering, Studies in Fuzziness and Soft Computing vol. 141, pp. 87–100. Springer, Berlin, Germany (2004)
12. Laguna M., Martí., R.: GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS J. Comput.*, **11**:44–52 (1999)
13. Laguna, M., Martí., R.: Scatter search: methodology and implementations in C. *Operations Research/Computer Science Interfaces Series*. Kluwer, Boston (2003)
14. Martí, R. (ed.): Feature cluster on scatter search methods for optimization. *European J. Oper. Res.*, **169**(2):351–698 (2006)
15. Oliveira, C.A., Pardalos, P.M., Resende., M.G.C.: GRASP with path-relinking for the quadratic assignment problem. In: Ribeiro, C.C., Martins, S.L. (eds.), Proceedings of III Workshop on Efficient and Experimental Algorithms, vol. 3059, pp. 356–368. Springer (2004)
16. Martí, R., Duarte, A., Laguna., M.: Advanced scatter search for the max-cut problem. *INFORMS J. Comput.*, **21**:26–38 (2009)
17. Ranjbar, M., Kianfar, F., Shadrokh., S.: Solving the resource availability cost problem in project scheduling by path relinking and genetic algorithm. *Appl. Math. Comput.*, **196**:879–888 (2008)
18. Resende, M.G.C., Martí, R., Gallego, M., Duarte., A.: GRASP and path relinking for the max-min diversity problem. *Comput. Oper. Res.* (2008) Published online 28 May 2008, doi: 10.1016/j.cor.2008.05.011.
19. Resende, M.G.C., Ribeiro., C.C.: GRASP with path-relinking: Recent advances and applications. In: Ibaraki, T., Nonobe, K., Yagiura, M. (eds.) *Metaheuristics: Progress as Real Problem Solvers*, pp. 29–63. Springer (2005)
20. Resende, M.G.C., Werneck., R.F.: A hybrid heuristic for the  $p$ -median problem. *J. Heuristics*, **10**:59–88 (2004)
21. Ribeiro, C.C., Rossetti., I.: A parallel GRASP heuristic for the 2-path network design problem. In: *Proceedings of Euro-Par 2002, Lecture Notes in Computer Science* vol. 2400, pp. 922–926. Springer, Paderborn (2002)
22. Ribeiro, C.C., Vianna., D.S.: A genetic algorithm for the phylogeny problem using an optimized crossover strategy based on path-relinking. In: *Anais do II Workshop Brasileiro de Bioinformática*, pp. 97–102, Macaé (2003)
23. Ribeiro, C.C., Vianna., D.S.: A hybrid genetic algorithm for the phylogeny problem using path-relinking as a progressive crossover strategy. *Int. Trans. Oper. Res.*, **16**(5):641–657 (2009)
24. Vallada, E., Ruiz., R.: New genetic algorithms with path relinking for the minimum tardiness permutation flowshop problem. *Omega*, **38**:57–67 (2010) doi:10.1016/j.omega.2009.04.002.
25. Zhang, G.Q., Lai., K.K.: Combining path relinking and genetic algorithms for the multiple-level warehouse layout problem. *Eur. J. Oper. Res.*, **169**:413–425 (2006)



# Chapter 5

## Genetic Algorithms

Colin R. Reeves

**Abstract** Genetic algorithms (GAs) have become popular as a means of solving hard combinatorial optimization problems. The first part of this chapter briefly traces their history, explains the basic concepts and discusses some of their theoretical aspects. It also references a number of sources for further research into their applications. The second part concentrates on the detailed implementation of a GA. It discusses the fundamentals of encoding a ‘genotype’ in different circumstances and describes the mechanics of population selection and management and the choice of genetic ‘operators’ for generating new populations. In closing, some specific guidelines for using GAs in practice are provided.

### 5.1 Introduction

The term *genetic algorithm*, almost universally abbreviated nowadays to GA, was first used by John Holland [1], whose book *Adaptation in Natural and Artificial Systems* of 1975 was instrumental in creating what is now a flourishing field of research and application that goes much wider than the original GA. Many people now use the term *evolutionary computing* or *evolutionary algorithms* (EAs), in order to cover the developments of the last 15 years. However, in the context of metaheuristics, it is probably fair to say that GAs in their original form encapsulate most of what one needs to know.

Holland’s influence in the development of the topic has been very important, but several other scientists with different backgrounds were also involved in developing similar ideas. In 1960s Germany, Ingo Rechenberg [2] and Hans-Paul Schwefel [3] developed the idea of the *Evolutionsstrategie* (in English, *evolution strategy*), while—also in the 1960s—Bremermann, Fogel and others in the USA implemented

---

Colin R. Reeves  
Department of Mathematics, Statistics and Engineering Science, Coventry University, Priory St,  
Coventry, UK  
e-mail: c.reeves@coventry.ac.uk

their idea for what they called *evolutionary programming*. The common thread in these ideas was the use of mutation and selection—the concepts at the core of the neo-Darwinian theory of evolution.<sup>1</sup> Although some promising results were obtained, evolutionary computing did not really take off until the 1980s. Note the least important reason for this was that the techniques needed a great deal of computational power. Nevertheless, the work of these early pioneers is fascinating to read in the light of our current knowledge; David Fogel (son of one of the early pioneers) has documented some of this work in [4].

1975 was a pivotal year in the development of genetic algorithms. It was in that year that Holland's book was published, but perhaps more relevantly for those interested in metaheuristics, that year also saw the completion of a PhD thesis by one of Holland's graduate students, Ken De Jong [5]. Other students of Holland's had completed theses in this area before, but this was the first to provide a thorough treatment of the GA's capabilities in optimization.

A series of further studies followed, the first conference on the nascent subject was convened in 1985, and another graduate student of Holland's, David Goldberg, produced first an award-winning PhD thesis on his application to gas pipeline optimization, and then, in 1989, an influential book [6]—*Genetic Algorithms in Search, Optimization, and Machine Learning*. This was the final catalyst in setting off a sustained development of GA theory and applications that is still growing rapidly.

Optimization has a fairly small place in Holland's work on adaptive systems, yet the majority of research on GAs tends to assume this is their purpose. De Jong, who initiated this interest in optimization, has cautioned that this emphasis may be misplaced in a paper [7] in which he contends that GAs are not really function optimizers, and that this is in some ways incidental to the main theme of adaptation. Nevertheless, using GAs for optimization is very popular, and frequently successful in real applications, and to those interested in metaheuristics, it will undoubtedly be the viewpoint that is most useful.

Unlike the earlier evolutionary algorithms, which focused on mutation and could be considered as straightforward developments of hill-climbing methods, Holland's GA had an extra ingredient—the idea of recombination. It is interesting in this regard to compare some of the ideas being put forward in the 1960s in the field of *operational research* (OR).

OR workers had by that time begun to develop techniques that seemed able to provide ‘good’ solutions, even if the quality was not provably optimal (or even near-optimal). Such methods became known as *heuristics*. A popular technique, which remains at the heart of many of the metaheuristics described in this handbook, was that of *neighbourhood search*, which has been used to attack a vast range of combinatorial optimization problems. The basic idea is to explore ‘neighbours’ of an existing solution—these being defined as solutions obtainable by a specified operation on the base solution.

One of the most influential papers in this context was that published by Lin [8], who found excellent solutions to the travelling salesman problem by investigating

---

<sup>1</sup> Well-meaning attempts to read off the validity or otherwise of Darwinism from the performance of GAs are illegitimate. GAs are clear examples of ‘intelligent design’.

neighbourhoods formed by breaking any three links of a tour and re-connecting them. Empirically, Lin found that these ‘3-optimal’ solutions were of excellent quality—in the case of the (rather small) problems he investigated, often close to the global optimum. However, he also made another interesting observation and suggested a way of exploiting it. While starting with different initial permutations gave different 3-optimal solutions, these 3-optimal solutions were observed to have a lot of features (links) in common. Lin therefore suggested that search should be concentrated on those links about which there was not a consensus, leaving the common characteristics of the solutions alone. This was not a GA as Holland was developing it, but there are clear resonances. Much later, after GAs had become more widely known, Lin’s ideas were re-discovered as ‘multi-parent recombination’ and ‘consensus operators’.

Other OR research of the same era took up these ideas. Roberts and Flores [9] (apparently independently) used a similar approach to Lin’s for the TSP, while Nugent et al. [10] applied this basic idea for the quadratic assignment problem. However, the general principle was not adopted into OR methodology, and relatively little was done to exploit the idea until GAs came on the OR scene in the 1990s.

In what follows, Section 5.2 provides an overview of the basic GA concepts, Section 5.3 gives a sketch of the theoretical background, while Section 5.4 lists some important sources for further exploration. The remaining sections focus on the various stages required for the implementation of a GA.

## 5.2 Basic Concepts

Assume we have a discrete search space  $\mathcal{X}$  and a function

$$f : \mathcal{X} \mapsto \mathbb{R}.$$

The general problem is to find

$$\arg \min_{x \in \mathcal{X}} f.$$

Here,  $x$  is a vector of *decision variables*, and  $f$  is the *objective function*. We assume here that the problem is one of minimization, but the modifications necessary for a maximization problem are nearly always obvious. Such a problem is commonly called a discrete or combinatorial optimization problem (COP).

One of the distinctive features of the GA approach is to allow the separation of the *representation* of the problem from the actual variables in which it was originally formulated. In line with biological usage of the terms, it has become customary to distinguish the ‘genotype’—the encoded representation of the variables, from the ‘phenotype’—the set of variables themselves. That is, the vector  $x$  is represented by a string  $s$ , of length  $l$ , made up of symbols drawn from an alphabet  $\mathcal{A}$ , using a mapping

$$c : \mathcal{A}^l \mapsto \mathcal{X}.$$

In practice, we may need to use a search space

$$\mathcal{S} \subseteq \mathcal{A}^l,$$

to reflect the fact that some strings in the image of  $\mathcal{A}^l$  under  $c$  may represent invalid solutions to the original problem. (This is a potential source of difficulty for GAs in combinatorial optimization—a topic that is covered in [11].) The string length  $l$  depends on the dimensions of both  $\mathcal{X}$  and  $\mathcal{A}$ , and the elements of the string correspond to ‘genes’, and the values those genes can take to ‘alleles’. This is often designated as the *genotype–phenotype mapping*. Thus the optimization problem becomes one of finding

$$\arg \min_{s \in \mathcal{S}} g,$$

where the function

$$g(s) = f(c(s)).$$

It is usually desirable that  $c$  should be a bijection. (The important property of a bijection is that it has an inverse, i.e., there is a unique vector  $x$  for every string  $s$ , and a unique string  $s$  for every vector  $x$ .) In some cases the nature of this mapping itself creates difficulties for a GA in solving optimization problems.

In using this device, Holland’s ideas are clearly distinct from the similar methodology developed by Rechenberg [2] and Schwefel [3], who preferred to work with the original decision variables directly. Both Holland’s and Goldberg’s books claim that representing the variables by binary strings (i.e.,  $\mathcal{A} = \{0, 1\}$ ) is in some sense ‘optimal’, and although this idea has been challenged, it is still often convenient from a mathematical standpoint to consider the binary case. Certainly, much of the theoretical work in GAs tends to make this assumption. In applications, many representations are possible—some of the alternatives that can be used in particular COPs are discussed in [11].

The original motivation for the GA approach was a biological analogy. In the selective breeding of plants or animals, for example, offspring are sought that have certain desirable characteristics—characteristics that are determined at the genetic level by the way the parents’ chromosomes combine. In the case of GAs, a *population* of strings is used, and these strings are often referred to in the GA literature as *chromosomes*. The recombination of strings is carried out using simple analogies of genetic *crossover* and *mutation*, and the search is guided by the results of evaluating the objective function  $f$  for each string in the population. Based on this evaluation, strings that have higher *fitness* (i.e., represent better solutions) can be identified, and these are given more opportunity to breed. It is also relevant to point out here that fitness is not necessarily to be identified simply with the composition  $f(c(s))$ ; more generally, fitness is  $h(f(c(s)))$  where  $h : \mathbb{R} \mapsto \mathbb{R}^+$  is a suitable monotonic function used to eliminate the problem of ‘negative’ fitness.

Perhaps the most fundamental characteristic of genetic algorithms is their use of populations of many strings. Here again, the German ‘evolution strategy’ (ES) school initially did not use populations and focused almost exclusively on ‘mutation’ operators which are generally closer in concept to the types of operator used

in neighbourhood search and its extensions. Holland did use mutation, but in his scheme it is generally treated as subordinate to crossover. Thus, in Holland's GA, instead of the search moving from point to point as in methods based on local search, the whole set of strings undergoes 'reproduction' in order to generate a new population.

De Jong's work established that population-based GAs using crossover and mutation operators could successfully deal with optimization problems of several different types, and in the years since his work was published, the application of GAs to COPs has grown almost exponentially.

These operators and some developments of them are described more fully in Sections 5.9 and 5.10. At this point, however, it might be helpful to provide a very basic introduction. Crossover is a matter of replacing some of the genes in one parent by corresponding genes of the other. An example of one-point crossover would be the following. Given the parents P1 and P2, with crossover point 3 (indicated by X), the offspring will be the pair O1 and O2:

P1	1	0	1	0	0	1	0		O1	1	0	1	1	0	0	1
			X													
P2	0	1	1	1	0	0	1		O2	0	1	1	0	0	1	0

The other common operator is mutation in which a gene (or subset of genes) is chosen randomly and the allele value of the chosen genes is changed. In the case of binary strings, this simply means complementing the chosen bit(s). For example, the string O1 above, with genes 3 and 5 mutated, would become 1 0 0 1 1 0 1. A simple template for the operation of a genetic algorithm is shown in Figure 5.1. The individual parts of this very general formulation will be discussed in detail later.

```

Choose an initial population of chromosomes;
while termination condition not satisfied do
    repeat
        if crossover condition satisfied then
            {select parent chromosomes;
            choose crossover parameters;
            perform crossover};
            if mutation condition satisfied then
                {choose mutation points;
                perform mutation};
                evaluate fitness of offspring
        until sufficient offspring created;
        select new population;
    endwhile

```

**Fig. 5.1** A genetic algorithm template. This is a fairly general formulation, accommodating many different forms of selection, crossover and mutation. It assumes user-specified conditions under which crossover and mutation are performed, a new population is created, and whereby the whole process is terminated.

## 5.3 Why Does It Work?

Exactly how and why GAs work is still hotly debated. There are various schools of thought, and none can be said to provide a definitive answer. A comprehensive survey is available in [12]. Meanwhile, the following is a brief guide to the main concepts that have been used.

### 5.3.1 The ‘Traditional’ View

Holland’s explanation of why it is advantageous to search the space  $\mathcal{A}^l$  rather than  $\mathcal{X}$  hinges on three main ideas. Central to this understanding is the concept of a *schema* (plural *schemata*). A schema is a subset of the space  $\mathcal{A}^l$  in which all the strings share a particular set of defined values. This can be represented by using the alphabet  $\mathcal{A} \cup *$ ; in the binary case,  $1 * * 1$ , for example, represents the subset of the 4-dimensional hypercube  $\{0, 1\}^4$  in which both the first and the last genes take the value 1, i.e., the strings  $\{1 0 0 1, 1 0 1 1, 1 1 0 1, 1 1 1 1\}$ .

The first of Holland’s ideas is that of *intrinsic* (also known as *implicit*) parallelism—the notion that information on many schemata can be processed in parallel. Under certain conditions that depend on population size and schema characteristics, Holland estimated that a population of size  $M$  contains information on  $\mathcal{O}(M^3)$  schemata. However, these schemata cannot *actually* be processed in parallel, because independent estimates of their fitness cannot be obtained in general [14].

The second concept is expressed by the so-called Schema Theorem, in which Holland showed that if there are  $N(S, t)$  instances of schema  $S$  in the population at time  $t$ , then at the next time step (following reproduction), the expected number of instances in the new population can be bounded by

$$E[N(S, t + 1)] \geq \frac{F(S, t)}{\overline{F(t)}} N(S, t) \{1 - \varepsilon(S, t)\},$$

where  $F(S, t)$  is the fitness of schema  $S$ ,  $\overline{F(t)}$  is the average fitness of the population, and  $\varepsilon(S, t)$  is a term that reflects the potential for genetic operators to destroy instances of schema  $S$ .

By failing to appreciate the stochastic and dynamic nature of this relationship, somewhat extravagant conclusions have been drawn from this theorem, expressed in the frequently made statement that good schemata will receive exponentially increasing numbers of trials in subsequent generations. However, it is clear that the Schema Theorem is a result in expectation only, and even then for just one generation. Any attempt to extrapolate this result for more than one generation is doomed to failure because the terms are then no longer independent of what is happening in the rest of the population. Moreover, given the finite population size, it is clear that any exponential increase cannot last very long.

Holland also attempted to model schema processing (or hyperplane competitions) by means of an analogy to stochastic two-armed bandit problems. This is

a well-known statistical problem: we are given two ‘levers’ which if pulled give ‘payoff’ values according to different probability distributions. The problem is to use the results of previous pulls in order to maximize the overall future expected payoff. In [1] it is argued that a GA approximates an ‘optimal’ strategy which allocates an (exponentially) increasing number of trials to the observed better lever; this is then used to contend for the supposed efficiency of a GA in distinguishing between competing schemata and hyperplanes.

Early accounts of GAs suggested quite strongly that in a GA we had thus discovered an algorithm that used the best available search strategy to solve not merely one, but many hyperplane competitions at once: the ‘only case where combinatorial explosion works in our favour’. Unfortunately, Wolpert and Macready’s ‘No-Free-Lunch’ Theorem (NFLT) [13] has rather destroyed such dreams.<sup>2</sup>

In fact, intrinsic parallelism turns out to be of strictly limited application; it merely describes the number of schemata that are likely to be present in some numbers given certain assumptions about string length, population size and (most importantly) the way in which the population has been generated—and the last assumption is unlikely to be true except at a very early stage of the search. Even then, only in very unusual circumstances—that of orthogonal populations [14]—could the hyperplane competitions actually be processed in parallel; normally, the competitions are not independent. The two-armed bandit analogy also fails in at least two ways: Macready and Wolpert [15] have firstly argued that there is no reason to believe that the strategy described by Holland as approximated by a GA is an optimal one, while they also believe there is also a flaw in Holland’s mathematics.

This is not to say that the Schema Theorem in particular, or the idea of a schema in general, is useless, but that what it says is of limited and mainly short-term value—principally, that certain schemata are likely to increase their presence in the next population, and that those schemata will be on the average fitter, and less resistant to destruction by crossover and mutation, than those that do not. Nevertheless, several researchers are working on new ways of formulating and understanding schema theory, while connecting it to other approaches; a recent summary can be found in [16].

This brings us to the third assumption implicit in the implementation of a GA—that the recombination of small pieces of the genotype (good schemata) into bigger pieces is indeed a sensible method of finding optimal solutions. Goldberg [6] calls this the building-block hypothesis (BBH). There is certainly some negative evidence, in that problems constructed to contain misleading building blocks may indeed be hard for a GA to solve. The failure of the BBH is often invoked as an explanation when a GA fails to solve particular COPs.

However, the properties of these problems are not usually such that they are uniquely difficult for GAs. Holland himself, with two other co-workers, looked for positive evidence in favour of the building-block hypothesis [17] and found

---

<sup>2</sup> The NFLT, put simply, says that on the average, nothing—ant colonies, GAs, simulated annealing, tabu search, etc.—is better than random search. Success comes from adapting the technique to the problem at hand, which of course implies some input of information from the researcher.

the results rather problematical: functions constructed precisely to provide a ‘royal road’ made up of building blocks of increasing size and fitness turned out to be much more efficiently solved by ‘non-genetic’ methods.

### 5.3.2 Other Approaches

By writing his theorem in the form of a lower bound, Holland was able to make a statement about schema  $S$  that is independent of what happens to other schemata. However, in practice what happens to schema  $S$  will influence the survival (or otherwise) of other schemata, and what happens to other schemata will affect what happens to  $S$ , as is made plain by the exact models of Vose [18] and Whitley [19].

Markov chain theory [18, 19] has been applied to GAs [20–22] to gain a better understanding of the GA as a whole. However, while the results are fascinating in illuminating some nuances of GA behaviour, the computational requirements are formidable for all but the smallest of problems, as shown by De Jong et al. [22], for example.

Shapiro et al. [23] first examined GAs from a statistical mechanics perspective, and there is a growing literature on this topic. Peck and Dhawan [24] have linked GAs to global randomized search methods. But one of the difficulties in analyzing GAs is that there is not a single generic GA, the behaviour of which will characterize the class of algorithms that it represents. In practice, there is a vast number of ways of implementing a GA, as will be seen in the discussion later, and what works in one case may not work in another. Some workers have therefore tried to look for ways of predicting algorithm performance for particular problem classes.

Reeves and Wright [14] summarize a perspective based on relating GAs to statistical methods of experimental design, which draws upon the biological concept of *epistasis*. This expresses the idea that the expression of a chromosome is not merely a sum of the effects of its individual alleles, but that the alleles located in some genes influence the expression of the alleles in others. From a mathematical viewpoint, epistasis is equivalent to the existence of interactions in the fitness function. If we knew the extent of these non-linearities, we might be able to choose an appropriate algorithm. Unfortunately, as is explained in [25], it is unlikely that this approach will be successful, although the literature surrounding the question of epistasis has produced some useful insights into GAs.

Several authors [26–28] have pointed out connections between GAs and neighbourhood search methods, and this has led to a considerable literature on the analysis of problem landscapes. The concept of a landscape has been used informally for many years, but recent work [29, 30] has put the idea on a rigorous mathematical foundation which is still being explored. Some of its uses in the context of GAs is described in [31]. It appears that this way of thinking about algorithms has great potential for unifying different metaheuristics and increasing our understanding of them.

## 5.4 Applications and Sources

There are numerous examples of the successful application of GAs to combinatorial optimization problems. Books such as those by Davis [32] and Chambers [33, 34] are useful in displaying the range of problems to which GAs have been applied. In a chapter such as this, it is impossible to give an exhaustive survey of relevant applications of GAs, but [11] lists some of the more useful and accessible references that should be of interest to people who are experimenting with metaheuristics. However, because of the enormous growth in reported applications of GAs, this list is inevitably incomplete, as well as somewhat dated already. For a time, Alander attempted to maintain a comprehensive bibliography: an early version of this is included in [34]. However, this is one area where the phenomenon of exponential growth is indubitable, and the sheer number of papers published in the last 15 years have rather overwhelmed this enterprise. Nonetheless, updates are made available periodically of selected papers in specific areas—the one of most interest to readers of this book being the OR bibliography [35], which is claimed to be comprehensive up to 1998, although it also includes some papers published later.

For more information on applications, and on GAs in general, the reader has several useful books to choose from: the early ones by Holland, Goldberg and Michalewicz [1, 6, 36] tend to be over-committed to the schema-processing point of view, but they are all still useful sources of information. Reeves [37] also reflects the state of the theory at the time the book was written, although it covers other heuristic methods too. More recently, Mitchell [38] and Falkenauer [39] demonstrate a more careful approach to schemata, and Bäck [40] covers the wider field of evolutionary algorithms. Eiben and Smith [41] also provide an elementary overview of the whole field, while—in contrast—Spears [42] offers an in-depth study on the trade-off between mutation and crossover.

All are worth consulting, but the best book now available is the recent work by De Jong [43]. For a very rigorous theoretical study, there is the book by Vose [44], which deals mainly with the Markov chain and dynamical systems approach, while Reeves and Rowe [12] have surveyed in some detail several other theoretical perspectives on GAs. Another rigorous theoretical study is that by Schmitt [45, 46].

There are now also many conferences on GAs and related topics—too many to list in detail. The original biennial *International Conference on Genetic Algorithms* series [47–53] is still of considerable historical interest<sup>3</sup>, while the IEEE established an alternative series under the title of the *Congress on Evolutionary Computation* [54–56]. These have now merged, under the auspices of the ACM since 2005, to create the annual GECCO series of conferences [57–59]. In Europe, there are two biennial series of somewhat wider scope: the *Parallel Problem Solving from Nature* series [67–72] and the *International Conference on Artificial Neural Networks and Genetic Algorithms* [77–82], recently renamed the *International Conference on Adaptive and Natural Computing Algorithms* [83, 84]. For the theoretically minded,

---

<sup>3</sup> Apart from the intrinsic interest of these papers, it is well worth checking to see if someone has tried your bright new idea already!

there is a biennial workshop to consider—the *Foundations of Genetic Algorithms* series [85–93].

There are also many journals now publishing GA-related research. The major GA journals are *Evolutionary Computation* (MIT Press) and *IEEE Transactions on Evolutionary Computation* (IEEE); other theoretical articles appear in journals related to AI or to complex systems. Most OR journals—*INFORMS Journal on Computing*, *Computers and OR*, *Journal of the OR Society*, *European Journal of OR*, etc.—have frequent papers on GAs, mainly applications. There are discussion groups on the Internet (`comp.ai.genetic`), and the moderated news digest at `GA-List-Request@aic.nrl.navy.mil`.

## 5.5 Initial Population

The previous sections have provided an overview of the underlying concepts, but it should be clear already that implementation of a GA requires many practical decisions. The major initial questions to consider relate to the population: first its size and second the method by which its individuals are chosen. The size of the population has been approached from several theoretical points of view, although the underlying idea is always of a trade-off between efficiency and effectiveness. Intuitively, it would seem that there should be some ‘optimal’ value for a given string length, on the grounds that too small a population would not allow sufficient room for exploring the search space effectively, while too large a population would so impair the efficiency of the method that no solution could be expected in a reasonable amount of time. Goldberg [94, 95] was probably the first to attempt to answer this question, using the idea of schemata. Unfortunately, from this viewpoint, it appeared that the population size  $M$  should increase as an exponential function of the string length. Experimental evidence [96, 97] suggests that populations of the size proposed by Goldberg’s theory are not necessary.

A slightly different question that we could ask is regarding a *minimum* population size for a meaningful search to take place. In Reeves [98], the initial principle was adopted that, at the very least, every point in the search space should be reachable from the initial population by crossover only. This requirement can only be satisfied if there is at least one instance of every allele at each locus in the whole population of strings. On the assumption that the initial population is generated by a random sample with replacement (which is a conservative assumption in this context), the probability that at least one allele is present at each locus can be found. For binary strings this is easily seen to be

$$P_2^* = (1 - (1/2)^{M-1})^l,$$

from which we can calculate that, for example, a population of size 17 is enough to ensure that the required probability exceeds 99.9% for strings of length 50. For  $q$ -ary alphabets, the calculation is somewhat less straightforward, but expressions are

given in [98] that can be converted numerically into graphs for specified confidence levels. The results of this work suggested that a population growth of  $\mathcal{O}(\log l)$  would be sufficient to cover the search space.

Finally, as to how the population is chosen, it is nearly always assumed that initialization should be random. Rees and Koehler [99], using a model-based approach that draws on the theoretical work of Vose [18], have demonstrated that sampling without replacement is preferable in the context of very small populations. More generally, it is obvious that randomly chosen points do not necessarily cover the search space uniformly, and there may be advantages in terms of coverage if we use more sophisticated statistical methods, especially for non-binary alphabets. One such simple idea is a generalization of the Latin hypercube which can be illustrated as follows:

Suppose each gene has 5 alleles, labelled 0, …, 4. We choose the population size  $M$  to be a multiple of 5, and the alleles in each ‘column’ are generated as an independent random permutation of 0, …,  $(M - 1)$ , which is then taken modulo 5. Figure 5.2 shows an example for a population of size 10. To obtain search space coverage at this level with simple random initialization would need a much larger population.

Individual	Gene
1	0 1 3 0 2 4
2	1 4 4 2 3 0
3	0 0 1 2 4 3
4	2 4 0 3 1 4
5	3 3 0 4 4 2
6	4 1 2 4 3 0
7	2 0 1 3 0 1
8	1 3 3 1 2 2
9	4 2 2 1 1 3
10	3 2 4 0 0 1

**Fig. 5.2** An example of Latin hypercube sampling for  $l = 6$  and  $|\mathcal{A}| = 5$ . Notice that each allele occurs exactly twice for each gene.

Another point to mention here is the possibility of ‘seeding’ the initial population with known good solutions. Some reports (e.g., in [100, 101]) have found that including a high-quality solution, obtained from another heuristic technique, can help a GA find better solutions rather more quickly than it can from a random start. However, there is also the possibility of inducing premature convergence [102, 103].

## 5.6 Termination

Unlike simple neighbourhood search methods that terminate when a local optimum is reached, GAs are stochastic search methods that could in principle run for ever. In practice, a termination criterion is needed; common approaches are to set a limit

on the number of fitness evaluations or the computer clock time or to track the population's diversity and stop when this falls below a preset threshold. The meaning of diversity in the latter case is not always obvious, and it could relate either to the genotype or to the phenotype, or even, conceivably, to the fitnesses, but the most common way to measure it is by genotype statistics. For example, we could decide to terminate a run if at every locus the proportion of one particular allele rose above 90%. Some attempts have been made to attack this problem from a theoretical point of view [104, 105], but as they are based on the idea of finding a probabilistic guarantee that all possible strings have been seen, their practical application is limited.

## 5.7 Crossover Condition

Given the stress on recombination in Holland's original work, it might be thought that crossover should always be used, but in fact there is no reason to suppose that it has to be so. Thus, while we could follow a strategy of crossover-AND-mutation to generate new offspring, it is also possible to use crossover-OR-mutation. There are many examples of both in the literature. The first strategy initially tries to carry out crossover, then attempts mutation on the offspring (either or both). It is conceivable that in some cases nothing actually happens at all with this strategy—the offspring are simply clones of the parents. Others always do something, either crossover or mutation, but not both. (Even then, cloning is still possible with crossover if the parents are too alike.)

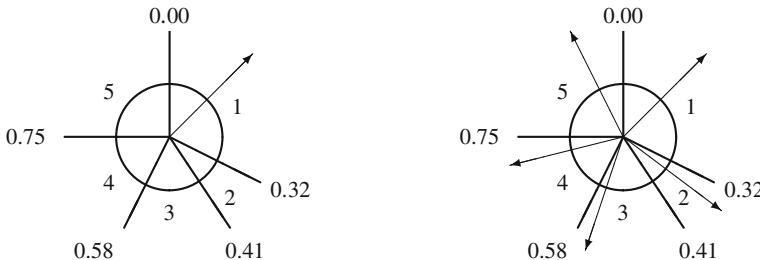
The mechanism for implementing such choices is customarily a randomized rule, whereby the operation is carried out if a pseudo-random uniform deviate exceeds a threshold value. In the case of crossover, this is often called the crossover rate, often denoted by the symbol  $\chi$ . For mutation, we have a choice between describing the number of mutations per string or per bit; bit-wise mutation, at a rate denoted by  $\mu$ , is more common.

In the -OR- case, there is a further possibility of modifying the relative proportions of crossover and mutation as the search progresses. Davis [32] has argued that different rates are appropriate at different times: high crossover at the start, high mutation as the population converges. In fact, he has suggested that the operator proportions could be adapted online, in accordance with their track record in finding new high-quality chromosomes.

## 5.8 Selection

The basic idea of selection is that it should be related to fitness, and the original scheme for its implementation is commonly known as the *roulette-wheel* method. It uses a probability distribution for selection in which the selection probability of

a given string is proportional to its fitness. Figure 5.3 provides a simple example of roulette-wheel selection (RWS). Pseudo-random numbers are used one at a time to choose strings for parenthood. For example, in Figure 5.3, the number 0.13 would select string 1, the number 0.68 would select string 4.



**Fig. 5.3** Suppose there are five strings in a population with fitnesses  $\{32, 9, 17, 17, 25\}$ , respectively. The probability of selection of each individual is proportional to the area of a sector of a roulette-wheel (or equivalently, to the angle subtended at the centre). The numbers on the spokes of the wheel are the cumulative probabilities for use by a pseudo-random number generator. On the left we have standard roulette-wheel selection, with a single pointer that has to be spun five times. On the right we have SUS, using five connected equally spaced pointers; one spin provides five selections.

Finding the appropriate number for a given pseudo-random number  $r$  requires searching an array for values that bracket  $r$ —this can be done in  $\mathcal{O}(\log M)$  time for a population of size  $M$ . However, this approach has a high stochastic variability, and the actual number of times  $N_C$  that chromosome  $C$  is selected in any generation may be very different from its expected value  $E[N_C]$ . For this reason, sampling *without replacement* may be used to ensure that at least the integral part of  $E[N_C]$  is achieved, with fractions being allocated using random sampling.

In practice, Baker's *stochastic universal selection* (SUS) [106] is a particularly effective way of realizing this outcome. Instead of a single choice at each stage, we imagine that the roulette wheel has an equally spaced multi-armed spinner. Spinning the wheel produces simultaneously the values  $N_C$  for all the chromosomes in the population. From the viewpoint of statistical sampling theory, this corresponds to systematic sampling [107]. Experimental work by Hancock [108] clearly demonstrates the superiority of this approach, although much published work on applications of GAs still appears to rely on the basic roulette-wheel method<sup>4</sup>.

An associated problem is that of finding a suitable measure of fitness for the members of the population. Simply using the objective function values  $f(x)$  is rarely sufficient, because the scale on which  $f(x)$  is measured is important. (For example, values of 10 and 20 are much more clearly distinguished than 1010 and 1020.) Also, in some cases, observed values of  $f$  may be negative, which complicates

<sup>4</sup> Note that the purpose of SUS is not to reduce the total of random numbers needed. Having generated a multiset of size  $M$  as our ‘mating pool’, we still have to decide which pairs mate together, whereas in RWS we can simply pair them in the order generated.

fitness-proportional schemes. Further, if the objective is minimization rather than maximization, a transformation is clearly required.

Some sort of scaling is thus usually applied, and Goldberg [6] gives a simple algorithm to deal with both minimization and maximization. The method is cumbersome, however, and it needs continual re-scaling as the search progresses. Two alternatives provide more elegant solutions.

### 5.8.1 Ranking

Ranking the chromosomes in fitness order loses some information, but there is no need for re-scaling, and selection algorithm is simpler and more efficient. Suppose the probability of selecting the string that is ranked  $k$ th in the population is denoted by  $P[k]$ . In the case of linear ranking, we assume that

$$P[k] = \alpha + \beta k,$$

where  $\alpha$  and  $\beta$  are constants. The requirement that  $P[k]$  be a probability distribution gives us one condition:

$$\sum_{k=1}^M (\alpha + \beta k) = 1,$$

which leaves us free to choose the other parameter in a way that tunes the *selection pressure*. This term is loosely used in many papers and articles on GAs. Here, we mean the following:

**Definition 5.1** *Selection pressure*

$$\phi = \frac{\text{Prob.}[selecting fittest string]}{\text{Prob.}[selecting average string]}.$$

In the case of linear ranking, we interpret the average as meaning the *median* string, so that

$$\phi = \frac{\alpha + \beta M}{\alpha + \beta(M+1)/2}$$

(This assumes the population size is odd—however, the analysis holds *mutatis mutandis* for the case of an even number.) Some simple algebra soon establishes that

$$\beta = \frac{2(\phi - 1)}{M(M-1)} \quad \text{and} \quad \alpha = \frac{2M - \phi(M+1)}{M(M-1)}$$

which implies that  $1 \leq \phi \leq 2$ . With this framework, it is easy to see that the cumulative probability distribution can be stated in terms of the sum of an arithmetic progression, so that finding the appropriate  $k$  for a given pseudo-random number  $r$  is simply a matter of solving the quadratic equation

$$\alpha k + \beta \frac{k(k+1)}{2} = r,$$

for  $k$ , which can be done simply in  $O(1)$  time. The formula is

$$k = \frac{-(2\alpha + \beta) \pm \sqrt{(2\alpha + \beta)^2 + 4\beta r}}{2\beta}.$$

In contrast, searching for  $k$  (given a value for  $r$ ) using ordinary fitness-proportional selection needs at least  $\mathcal{O}(\log M)$  time.

Other functions can be used besides linear ranking [108, 109] but the above scheme is sufficiently flexible for most applications.

### 5.8.2 Tournament Selection

The other alternative to strict fitness-proportional selection is *tournament selection*, in which a set of  $\tau$  chromosomes are chosen and compared, the best one being selected for parenthood. This approach has similar properties to linear ranking for  $\tau = 2$ . It is easy to see that the best string will be selected every time it is compared, while the median string will be chosen with probability  $2^{-(\tau-1)}$ . Thus the selection pressure is given by  $\phi = 2^{\tau-1}$ , which for  $\tau = 2$  is similar to linear ranking when  $\alpha \rightarrow 0$ .

One potential advantage of tournament selection over all other forms is that it only needs a preference ordering between pairs or groups of strings, and it can thus cope with situations where there is no formal objective function at all—in other words, it can deal with a purely *subjective* objective function!

However, tournament selection is also subject to arbitrary stochastic effects in the same way as roulette-wheel selection—there is no guarantee that every string will appear in a given cycle. Indeed, using sampling with replacement there is a probability of approximately  $e^{-1}$  ( $\approx 0.368$ ) that a given string will not appear at all. One way of coping with this, at the expense of a little extra computation, is to use a variance reduction technique from simulation theory. Saliby [110] distinguishes between the *set* effect and the *sequence* effect in drawing items from a finite population. In applying his ideas here, we know that we need  $\tau$  items to be drawn  $M$  times, so we simply construct  $\tau$  random permutations<sup>5</sup> of the numbers  $1, \dots, M$ —the indices of the individuals in the population. These are concatenated into one long sequence which is then chopped up into  $M$  pieces, each containing the  $\tau$  indices of the individuals to be used in the consecutive tournaments. If  $M$  is not an exact multiple of  $\tau$ , there is the small chance of some distortion where the permutations join, but this is a relatively minor problem.

---

<sup>5</sup> There is a simple algorithm for doing this efficiently—see Nijenhuis and Wilf [111], for example, or look at the Stony Brook Algorithm Repository [112].

## 5.9 Crossover

Crossover is simply a matter of replacing some of the genes in one parent by the corresponding genes of the other. Suppose we have two strings  $a$  and  $b$ , each consisting of six variables, i.e.,

$$(a_1, a_2, a_3, a_4, a_5, a_6) \quad \text{and} \quad (b_1, b_2, b_3, b_4, b_5, b_6),$$

which represent two possible solutions to a problem. One-point crossover (**1X**) has been described earlier in the context of a binary alphabet. (Note that we have chosen here to leave the alphabet unspecified, to emphasize that binary representation is not a critical aspect of GAs.) Two-point crossover (denoted by **2X**) is very similar: two crosspoints are chosen at random from the numbers  $1, \dots, 5$ , and a new solution produced by combining the pieces of the original ‘parents’. For instance, if the crosspoints were 2 and 4, the ‘offspring’ solutions would be

$$(a_1, a_2, b_3, b_4, a_5, a_6) \quad \text{and} \quad (b_1, b_2, a_3, a_4, b_5, b_6)$$

A similar prescription can be given for  $m$ -point crossover where  $m > 1$ .

An early and thorough investigation of multipoint crossovers is that by Eshelman et al. [113], who examined the biasing effect of traditional one-point crossover and considered a range of alternatives. Their central argument is that two sources of bias exist to be exploited in a genetic algorithm: *positional* bias and *distributional* bias. One-point crossover has considerable positional bias, in that it relies on the building-block hypothesis, and if this is invalid, the bias may prevent the production of good solutions.

On the other hand, **1X** has no distributional bias, in that the crossover point is chosen randomly using the uniform distribution. But this lack of bias is not necessarily a good thing, as it limits the exchange of information between the parents. In [113], the possibilities of changing these biases, in particular by using multipoint crossover, were investigated and empirical evidence strongly supported the suspicion that one-point crossover is not the best option. In fact, despite some ambiguities, the evidence seemed to point to an 8-point crossover operator as the best overall, in terms of the number of function evaluations needed to reach the global optimum, averaged over a range of problem types.

Another obvious alternative, which removes any bias, is to make the crossover process completely random—the so-called uniform crossover. This can be seen most easily by observing that a crossover operator itself can be written as a binary string or *mask*—in fact, when implementing crossover in a computer algorithm, this is the obvious way to do it. For example, the mask

1 1 0 0 1 1

represents the 2-point crossover used above, where a 1 means that the alleles are taken from the first parent, while a 0 means they come from the second.

By generating the pattern of 0s and 1s stochastically (using a Bernoulli distribution) we thus get uniform crossover (**UX**), which might generate a mask such as

1 0 1 0 0 1

which implies that the 1st, 3rd and 6th alleles are taken from the first parent, the others from the second. This idea was first used by Syswerda [114], who implicitly assumed the Bernoulli parameter  $p = 0.5$ . Of course, this is not necessary: we can bias UX towards one or the other parent by choosing  $p$  appropriately.

De Jong and Spears [115] produced a theoretical analysis that was able to characterize the amount of disruption introduced by a given crossover operator exactly. In particular, the amount of disruption in UX can be tuned by choosing different values of  $p$ .

Of course, there are also many practical considerations that influence the implementation of crossover. How often do we apply it? Some always do, others use a stochastic approach, applying crossover with a probability  $\chi < 1$ . Do we generate one offspring or two? In many cases there are natural ‘twin’ offspring resulting, but in more sophisticated problems it may be that only one offspring arises. When we choose only one from two, how do we do it? In accordance with the stochastic nature of the GA, we may well decide to choose either of the offspring at random. Alternatively, we could bias the decision by making use of some other property such as the fitness of the new individuals or the loss (or gain) in diversity that results in choosing one rather than the other.

Booker [116] reported significant gains from using an adaptive crossover rate: the rate was varied according to a characteristic called *percent involvement*. This is simply the percentage of the current population that is producing offspring—too small a value is associated with loss of diversity and premature convergence.

### 5.9.1 Non-linear Crossover

In cases of non-linear encodings, crossover has to be reinterpreted. One of the most frequently occurring problems is where the solution space is the space of permutations ( $\Pi_l$ ) of the numbers  $1, \dots, l$ —well-known examples of this include many scheduling problems, and the famous travelling salesman problem (TSP).

For instance, the simple-minded application of 1X with crosspoint  $X = 2$  in the following case produces an infeasible solution:

P1	1	6	3	4	5	2		O1	1	6	1	2	6	5
			X											
P2	4	3	1	2	6	5		O2	4	3	3	4	5	2

If this represents a TSP, the first offspring visits cities 1 and 6 twice, and never gets to cities 3 or 4. A moment’s thought is enough to realize that this type of behaviour will be the rule, not an exception. Clearly we need to think of something rather smarter if we are to be able to solve such problems.

One of the first ideas for such problems was the PMX (partially mapped crossover) operator [94], which operates as follows: Two crossover points are chosen uniformly at random between 1 and  $l$ . The section between these points defines

an interchange mapping. Thus, in the example above, PMX (with crosspoints X=2 and Y=5) might proceed as follows:

P1	1	6	3	4	5	2		O1	3	5	1	2	6	4
		X		Y										
P2	4	3	1	2	6	5		O2	2	1	3	4	5	6

Here the crossover points X and Y define an interchange mapping

$$3 \leftrightarrow 1 \quad 4 \leftrightarrow 2; \quad 5 \leftrightarrow 6$$

on their respective strings, which means that the cut blocks have been swapped and now appear in different contexts from before. Another possibility is to apply a binary mask, as in linear crossover, but with a different meaning. Such a mask, generated as with UX, say, might be the following

$$1 \ 0 \ 1 \ 0 \ 0 \ 1$$

which is applied to the parents in turn. First the components corresponding to 1s are copied from one parent, and then those that correspond to 0s are taken in the order they appear from the second parent in order to fill the gaps. Thus the above example generates the following pairs of strings:

P1	1	6	3	4	5	2	->	1	_	3	_	_	2	O1	1	4	3	6	5	2
P2	4	3	1	2	6	5	->	4	_	1	_	_	5	O2	4	6	1	3	2	5

## 5.10 Mutation

First we note that in the case when crossover-OR-mutation is used, we must first decide whether any mutation is carried out at all. Assuming that it is the concept of mutation is even simpler than crossover, and again, this can easily be represented as a bit-string, so we generate a mask such as

$$0 \ 1 \ 0 \ 0 \ 0 \ 1$$

using a Bernoulli distribution at each locus—with a small value of  $p$  in this case. (The above example would then imply that the 2nd and 6th genes are assigned new allele values.) However, it appears that there are variant ways of implementing this simple idea that can make a substantial difference to the performance of a GA. The naive idea would be to draw a random number for *every* gene in the string and compare it to  $\mu$ , but this is potentially expensive in terms of computation if the strings are long and the population is large. An efficient alternative is to draw a random variate from a Poisson distribution with parameter  $\lambda$ , where  $\lambda$  is the average number of mutations per chromosome. A common value for  $\lambda$  is 1—in other words, if  $l$  is the string length, the (bit-wise) mutation rate is  $\mu = 1/l$ , which as early as 1966 [118] was shown to be in some sense an ‘optimal’ mutation rate. If our Poisson

random draw proposes that there are (say)  $m$  mutations, we draw  $m$  random numbers (without replacement) uniformly distributed between 1 and  $l$  in order to specify the loci where mutation is to take place.

In the case of binary strings, mutation simply means complementing the chosen bit(s). More generally, when there are several possible allele values for each gene, if we decide to change a particular allele, we must provide some means of deciding what its new value should be. This could be a random choice, but if (as in some cases) there is some ordinal relation between allele values, it may be more sensible to restrict the choice to alleles that are close to the current value or at least to bias the probability distribution in their favour.

It is often suggested that mutation has a somewhat secondary function, that of helping to preserve a reasonable level of population diversity—an insurance policy which enables the process to escape from sub-optimal regions of the solution space, but not all authors agree. Proponents of evolutionary programming ([119], for example), consider crossover to be an irrelevance, and mutation plays the major role. The balance between crossover and mutation is often a problem-specific one, and definite guidelines are hard to give.

However, several authors have suggested some type of adaptive mutation: for example, Fogarty [120] experimented with different mutation rates at different loci. Reeves [100] varied the mutation probability according to the diversity in the population (measured in terms of the coefficient of variation of fitnesses). More sophisticated procedures are possible, and anecdotal evidence suggests that many authors use some sort of diversity maintenance policy. In this connection, it should also be mentioned that there is interest currently in ‘parameter-less’ GAs. It is impossible to eliminate all parameter values, of course, but there has always been interest in some sort of adaptation as the search proceeds, not only for mutation rates but also for other parameters, such as population size. Eiben et al. [121] summarize some of the recent work in this area.

Finally, it should be no surprise that the values of different parameters interact with each other, in terms of the overall performance of the GA. For example, choosing a high selection pressure may mean that we also need a high mutation rate in order to avoid premature convergence. De Jong [43] has an extensive discussion on such matters.

## 5.11 New Population

Holland’s original GA assumed a *generational* approach: selection, recombination and mutation were applied to a population of  $M$  chromosomes until a new set of  $M$  individuals had been generated. This set then became the new population. From an optimization viewpoint this seems an odd thing to do—we may have spent considerable effort obtaining a good solution, only to run the risk of throwing it away and thus preventing it from taking part in further reproduction. For this reason, De Jong [5] introduced the concepts of *élitism* and *population overlaps*. His ideas

are simple—an élitist strategy ensures the survival of the best individual so far by preserving it and replacing only the remaining ( $M - 1$ ) members of the population with new strings. Overlapping populations take this a stage further by replacing only a fraction  $G$  (the *generation gap*) of the population at each generation. Finally, taking this to its logical conclusion produces the so-called steady-state or incremental strategies, in which only one new chromosome (or sometimes a pair) is generated at each stage. Davis [32] gives a good general introduction to this type of GA.

Slightly different strategies are commonly used in the ES community, which traditionally designates them either  $\lambda, \mu$  or  $\lambda + \mu$ . In the first case,  $\mu (> \lambda)$  offspring are generated from  $\lambda$  parents, and the best  $\lambda$  of these offspring are chosen to start the next generation. For the  $+$  strategy,  $\mu$  (not necessarily  $> \lambda$ ) offspring are generated and the best  $\lambda$  individuals are chosen from the combined set of parents and offspring.

In the case of incremental reproduction it is also necessary to select members of the population for deletion. Some GAs have assumed that parents are replaced by their children. Many implementations, such as Whitley's GENITOR [109], use the tactic of deleting the worst member(s) of the population, although (as Goldberg and Deb [122] have pointed out) this exerts a very strong selective pressure on the search, which may need fairly large populations and high mutation rates to prevent a rapid loss of diversity. A milder prescription is to select from the worst  $p\%$  of the population (for example, Reeves [100] used  $p = 50$ , i.e., selection from those worse than the median). This is easily implemented when rank-based selection is used. Yet another approach is to base deletion on the *age* of the strings.

### 5.11.1 Diversity Maintenance

As hinted above, one of the keys to good performance (in nature as well as in GAs) is to maintain the diversity of the population as long as possible. The effect of selection is to reduce diversity, and some methods can reduce diversity very quickly. This can be mitigated by having larger populations or by having greater mutation rates, but other techniques are also often employed.

A popular approach, commonly linked with steady-state or incremental GAs, is to use a 'no-duplicates' policy [32]. This means that the offspring are not allowed into the population if they are merely clones of existing individuals. The downside, of course, is the need to compare each current individual with the new candidate, which adds to the computational effort needed—an important consideration with large populations. (In principle, some sort of 'hashing' approach could be used to speed this process up, but whether this has ever been tried is not clear.)

We can of course take steps to reduce the chance of cloning before offspring are generated. For instance, with 1X, the two strings

1	1	0	1	0	0	1
1	1	0	0	0	1	0

will generate only clones if the crossover point is any of the first three positions. Booker [116] suggested that before applying crossover, we should examine the selected parents to find suitable crossover points. This entails computing an ‘exclusive-OR’ (XOR) between the parents, so that only positions between the outermost 1s of the XOR string (the ‘reduced surrogate’) should be considered as crossover points. Thus in the example above, the XOR string is

0 0 0 1 0 1 1

so that, as previously stated, only the last three crossover points will give rise to a different string.

## 5.12 Representation

As remarked in Section 5.1, the focus in this handbook is on using GAs as optimizers in a search space, given a suitable encoding and fitness function. We now consider how the search space  $\mathcal{S}$  might be constructed in some generic cases.

### 5.12.1 Binary Problems

In some problems a binary encoding might arise naturally. Consider the operational research problem known as the *knapsack* problem, stated as follows.

*Example 1 (The 0-1 knapsack problem)* A set of  $n$  items is available to be packed into a knapsack with capacity  $C$  units. Item  $i$  has value  $v_i$  and uses up  $c_i$  units of capacity. Determine the subset  $I$  of items which should be packed in order to maximize

$$\sum_{i \in I} v_i$$

such that

$$\sum_{i \in I} c_i \leq C.$$

If we define

$$x_i = \begin{cases} 1 & \text{if item } i \text{ is packed} \\ 0 & \text{otherwise} \end{cases}$$

the knapsack problem can be re-formulated as an *integer program*:

$$\text{maximize } \sum_{i=1}^n x_i v_i,$$

$$\text{such that } \sum_{i=1}^n x_i c_i \leq C,$$

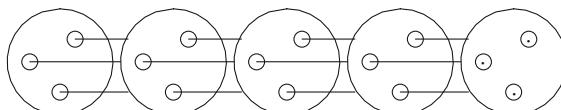
from which it is clear that we can define a solution as a binary string of length  $n$ . In this case there is thus no distinction between genotype and phenotype.

However, such problems are not necessarily easy to solve with a GA. In this case, the presence of constraints is likely to cause difficulties—two feasible parents may not produce feasible offspring, unless special crossover operators are constructed. In fact, such problems as these are really subset selection problems, which are best tackled by other means [123], despite the seductiveness of the binary encoding. It is now widely recognized that ‘natural’ binary encodings nearly always bring substantial problems for simple GAs.

### 5.12.2 Discrete (but Not Binary) Problems

There are cases in which a discrete alphabet of higher cardinality than 2 might be appropriate. The rotor-stacking problem, as originally described by McKee and Reed [124], is a good example.

*Example 2* A set of  $n$  rotors are available, each of which has  $k$  holes drilled in it. The rotors have to be assembled into a unit by stacking them and bolting them together, as in Figure 5.4. Because the rotors are not perfectly flat, stacking them in different orientations will lead to assemblies with different characteristics in terms of deviations from true symmetry, with the consequent effect (in operation) that the assembled unit will wobble as it spins. The objective is to find which of all the possible combinations of orientations produce the least deviation.



**Fig. 5.4** Rotor-stacking problem with  $n = 5$  rotors and  $k = 3$  holes.

In this case a  $k$ -ary coding is natural. A solution is represented by a string of length  $n$ , each gene corresponding to a rotor and the alleles, drawn from  $\{1, \dots, k\}$ , representing the orientation (relative to a fixed datum) of the holes. Thus, the string (1322) represents a solution to a 4-rotor problem where hole 1 of the first rotor is aligned with hole 3 of the second and hole 2 of the third and fourth. Of course, it would be possible to encode the alleles as binary strings, but there seems little point in so doing—particularly if  $k$  is not a power of 2, as there will then be some binary strings that do not correspond to any actual orientation.

This seems very straightforward, although there is a subtle point that could be overlooked. The assignment of labels to the holes is arbitrary, and this creates the

problem of ‘competing conventions’ as it has been called<sup>6</sup>. For example, given a natural order for labelling each rotor, the string (3211) represents the same solution as (1322). This can be alleviated in this case by fixing the labelling for one rotor, so that a solution can be encoded by a string of length  $(n - 1)$ .

As far as the operators are concerned, standard crossovers can be used here, but mutation needs some careful consideration in the case of  $k$ -ary coding, as outlined in Section 5.10.

### 5.12.3 Permutation Problems

There are also some problems where the ‘obvious’ choice of representation is defined, not over a set, but over a permutation. The TSP is one of many problems for which this is true. As another example, consider the permutation flowshop sequencing problem (PFSP).

*Example 3* Suppose we have  $n$  jobs to be processed on  $m$  machines, where the processing time for job  $i$  on machine  $j$  is given by  $p(i, j)$ . For a job permutation  $\{\pi_1, \pi_2, \dots, \pi_n\}$ , we calculate the completion times  $C(\pi_i, j)$  as follows:

$$\begin{aligned} C(\pi_1, 1) &= p(\pi_1, 1) \\ C(\pi_i, 1) &= C(\pi_{i-1}, 1) + p(\pi_i, 1) \quad \text{for } i = 2, \dots, n \\ C(\pi_1, j) &= C(\pi_1, j-1) + p(\pi_1, j) \quad \text{for } j = 2, \dots, m \\ C(\pi_i, j) &= \max\{C(\pi_{i-1}, j), C(\pi_i, j-1)\} + p(\pi_i, j) \\ \text{for } i &= 2, \dots, n; \ j = 2, \dots, m \end{aligned}$$

The PFSP is then to find a permutation  $\pi^*$  in the set of all permutations  $\Pi$  such that

$$f(\pi^*) \leq f(\pi) \quad \forall \pi \in \Pi.$$

(Several performance measures  $f(\cdot)$  are possible; common ones are the maximum or mean completion time.)

Here the natural encoding (although not the only one) is simply the permutation of the jobs as used to calculate the completion times. So the solution (1462537), for example, simply means that job 1 is first on each machine, then job 4, job 6, etc.

Unfortunately, the standard crossover operators patently fail to preserve the permutation except in very fortunate circumstances, as discussed in Section 5.9.1. Some solutions to this problem were outlined there; more comprehensive discussion of possible methods of attack is contained in [126, 127], while [100, 128] describe some approaches of particular relevance to the PFSP.

---

<sup>6</sup> This phenomenon is a common one whenever the coding function  $c(\cdot)$  is not injective. It has been observed in problems ranging from optimizing neural nets to the TSP. Radcliffe, who calls it ‘degeneracy’ [125], has presented the most thorough analysis of this problem and how to treat it.

### 5.12.4 Non-binary Problems

In many cases the natural variables for the problem are not binary, but integer or real-valued. In such cases a transformation to a binary string is required first. (Note that this is a different situation from the rotor-stacking example, where the integers were merely labels: here the values are assumed to be meaningful as numbers.) While the main thrust of metaheuristics research and application is directed to discrete optimization, it is perhaps appropriate to mention these other problems here.

*Example 4* It is required to maximize

$$f(x) = x^3 - 60x^2 + 900x + 100$$

over the search space  $\mathcal{X} = \{x : x \in \mathbb{Z}; x \in \{0, 31\}\}$ , i.e., the solution  $x^*$  is required to be an integer in the range  $[0, 31]$ .

To use the conventional form of genetic algorithm here, we would use a string of 5 binary digits with the standard binary to integer mapping, i.e.,  $(0, 0, 0, 0, 0) = 0, \dots, (1, 1, 1, 1, 1) = 31$ . Of course, in practice we could solve such a problem easily without recourse to encoding the decision variable in this way, but it illustrates neatly the sort of optimization problem to which GAs are often applied. Such problems assume first that we know the domain of each of our decision variables, and second that we have some idea of the precision with which we need to specify our eventual solution. Given these two ingredients, we can determine the number of bits needed for each decision variable and concatenate them to form the chromosome. More information on this topic can be found in [12].

## 5.13 Random Numbers

As GAs are stochastic in nature, it is clear that a reliable random number source is very important. Most computer systems have built-in `rand()` functions, and that is the usual method of generating random numbers. Not all random number generators are reliable, however, as Ross [129] has pointed out, and it is a good idea to use one that has been thoroughly tested, such as those described in the *Numerical Recipes* series [130].

## 5.14 Conclusions

While this exposition has covered the basic principles of GAs, the number of variations that have been suggested is enormous. Probably everybody's GA is unique! Many variations in population size, in initialization methods, in fitness definition, in selection and replacement strategies, in crossover and mutation are

obviously possible. Some have added information such as age, or artificial tags, to chromosomes; others have allowed varying population sizes or induced the formation of multiple populations in ‘niches’. It is in the nature of GAs that parallel processing can often be used to advantage, and here again, there are many possibilities, ranging from simple parallelization of function evaluations to very sophisticated implementations that add a spatial aspect to the algorithm.

The GA community has yet to reach a consensus on any of these things, and in the light of the NFLT, this is perhaps not surprising. However, some ideas do emerge as a reasonable set of recommendations. From a practitioner’s viewpoint, Levine made the following observations:

1. A steady-state (or incremental) approach is generally more effective and efficient than a generational method.
2. Don’t use simple roulette-wheel selection. Tournament selection or SUS is better.
3. Don’t use one-point crossover. UX or 2X should be preferred.
4. Make use of an adaptive mutation rate—one that is fixed throughout the search (even at  $1/l$ ) is too inflexible.
5. Hybridize wherever possible; don’t use a GA as a black box, but make use of any problem-specific information that you have.

Not everyone will agree with this particular list, and there is a conflict inherent in the first two points, since SUS functions best in a generational setting. Broadly speaking, however, it is one with which many researchers would be comfortable. Two other points could be added:

6. Make diversity maintenance a priority.
7. Don’t be afraid to run the GA several times.

Why this last point? Statements are frequently made that GAs can find global optima. Well, they can—but usually they tend to converge to some other ‘attractor’. In fact, there is some evidence [131] that even with very large populations the attractors are a subset of the local optima relating to a neighbourhood search. With practical population sizes, the attractors may not even be restricted to such a set and may be some distance from global optimality. It thus makes sense to explore several alternatives.

## References

1. Holland, J.H.: *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, Michigan; re-issued by MIT Press (1992) (1975)
2. Rechenberg, I.: *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. 2nd edn. 1993 Frommann-Holzboog Verlag, Stuttgart (1973)
3. Schwefel, H-P.: *Numerische Optimierung von Computer-modellen mittels der Evolutionstrategie*. Birkhäuser, Basel. (English edn. *Numerical Optimization of Computer Models*, John, Chichester, (1981) (1977)

4. Fogel, D.B.: Evolutionary Computation: The Fossil Record. IEEE Press, Piscataway, NJ (1998)
5. De Jong, K.A.: An Analysis of the Behavior of a Class of Genetic Adaptive Systems. Doctoral dissertation, University of Michigan, Ann Arbor, Michigan (1975)
6. Goldberg, D.E.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Reading, Massachusetts (1989)
7. De Jong, K.A.: Genetic algorithms are NOT function optimizers. In: [Whitley, L.D. (ed.): Foundations of Genetic Algorithms 2. Morgan Kaufmann, San Mateo, CA (1993)], pp. 5–18 (1993)
8. Lin, S.: Computer solutions of the traveling salesman problem. Bell Systems Tech. J. **44**, pp. 2245–2269 (1965)
9. Roberts, S.M., Flores, B.: An engineering approach to the travelling salesman problem. Man. Sci. **13**, 269–288 (1966)
10. Nugent, C.E., Vollman, T.E., Rumml, J.E.: An experimental comparison of techniques for the assignment of facilities to locations. Oper. Res. **16**, 150–173 (1968)
11. Reeves, C.R.: Genetic algorithms for the Operations Researcher. INFORMS J Comput. **9**, 231–250 (1997)
12. Reeves, C.R., Rowe, J.E.: Genetic Algorithms—Principles and Perspectives. Kluwer, Norwell, MA (2002)
13. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Trans. Evol. Comput. **1**, 67–82 (1997)
14. Reeves, C.R., Wright, C.C.: Genetic algorithms and the design of experiments. In: Davis, L.D. De Jong, K.A. Vose, M.D., Whitley, L.D. (eds.) Evolutionary Algorithms: IMA Volumes in Mathematics and its Applications, vol. 111, 207–226 Springer, New York (1999)
15. Macready, W.G., Wolpert, D.H.: Bandit problems and the exploration/exploitation tradeoff. IEEE Trans. Evol. Comput. **2**, 2–13 (1998)
16. Stephens, C.R., Zamora, A., Wright, A.H.: Perturbation theory and the renormalization group in genetic dynamics. In: [Wright, A.H., et al. (eds.): Foundations of Genetic Algorithms 8, LNCS 3469. Springer, Berlin (2005)], pp. 192–214 (2005)
17. Mitchell, M., Holland, J.H., Forrest, S.: When will a genetic algorithm outperform hill climbing? In: Cowan, J.D., Tesauro, G., Alspector, J. (eds.) (1994) Advances in Neural Information Processing Systems 6, Morgan Kaufmann, San Mateo, CA (1994)
18. Vose, M.D.: Modeling simple genetic algorithms. In [Whitley, L.D. (ed.): Foundations of Genetic Algorithms 2. Morgan Kaufmann, San Mateo, CA (1993)], CA, 63–73 (1993)
19. Whitley, D.: An executable model of a simple genetic algorithm. In: [Whitley, L.D. (ed.): Foundations of Genetic Algorithms 2. Morgan Kaufmann, San Mateo, CA (1993)], pp. 45–62 (1993)
20. Vose, M.D.: A closer look at mutation in genetic algorithms. Ann. Math. AI **10**, 423–434 (1994)
21. Vose, M.D., Wright, A.H.: Stability of vertex fixed points and applications. In: [Whitley, D., Vose, M. (eds.): Foundations of Genetic Algorithms 3. Morgan Kaufmann, San Mateo, CA (1995)], pp. 103–113 (1995)
22. De Jong, K.A., Spears, W.M., Gordon, D.F.: Using Markov chains to analyze GAFOs. In: [Whitley, D., Vose, M. (eds.): Foundations of Genetic Algorithms 3. Morgan Kaufmann, San Mateo, CA (1995)], pp. 115–137 (1995)
23. Shapiro, J.L., Prügel-Bennett, A., Rattray, M.: A statistical mechanics formulation of the dynamics of genetic algorithms. Lecture Notes in Computer Science **65**, Springer Berlin, 17–27 (1994)
24. Peck, C.C., Dhawan, A.P.: Genetic algorithms as global random search methods: An alternative perspective. Evolutionary Computation **3**, 39–80 (1995)
25. Reeves, C.R.: Predictive measures for problem difficulty. In: Proceedings of 1999 Congress on Evolutionary Computation, IEEE Press, 736–743 (1999)

26. Reeves, C.R.: Genetic algorithms and neighbourhood search. In: Fogarty, T.C., (ed.) *Evolutionary Computing: AISB Workshop*, Leeds, UK, April 1994; Selected Papers, Springer, Berlin, 115–130 (1994)
27. Jones, T.C.: Evolutionary Algorithms, Fitness Landscapes and Search. Doctoral dissertation, University of New Mexico, Albuquerque, NM (1995)
28. Culberson, J.C.: Mutation-crossover isomorphisms and the construction of discriminating functions. *Evol. Comput.* **2**, 279–311 (1995)
29. Stadler, P.F., Wagner, G.P.: Algebraic theory of recombination spaces. *Evol. Comput.* **5**, 241–275 (1998)
30. Reidys, C.M., Stadler, P.F.: Combinatorial landscapes. *SIAM Review* **44**, 3–54 (2002)
31. Reeves, C.R.: Fitness landscapes and evolutionary algorithms. In: Fonlupt, C., Hao, J-K., Lutton, E., Ronald, E., Schoenauer, M., (eds.) *Artificial Evolution: 4th European Conference; Selected Papers*. Springer, Berlin, 3–20 (2000)
32. Davis, L. (ed.): *Handbook of Genetic Algorithms*. Van Nostrand Reinhold, New York (1991)
33. Chambers, L. (ed.): *Practical Handbook of Genetic Algorithms: Applications*, Volume I. CRC Press, Boca Raton, Florida (1995)
34. Chambers, L. (ed.): *Practical Handbook of Genetic Algorithms: New Frontiers*, Volume II. CRC Press, Boca Raton, Florida (1995)
35. Alander, J.T.: An Indexed Bibliography of Genetic Algorithms in Operations Research. <ftp://garbo.uwasa.fi/cs/report94-1/gaORbib.pdf>. last accessed 16 May 2009 (2008)
36. Michalewicz, Z.: *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd edn. Springer, Berlin (1996)
37. Reeves, C.R. (ed.): *Modern Heuristic Techniques for Combinatorial Problems*. Blackwell Scientific Publications, Oxford, UK; re-issued by McGraw-Hill, London, UK (1995) (1993)
38. Mitchell, M.: *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA (1996)
39. Falkenauer, E.: *Genetic Algorithms and Grouping Problems*. John Wiley & Sons, Chichester (1998)
40. Bäck, Th.: *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, Oxford (1996)
41. Eiben, A.E., Smith, J.E.: *Introduction to Evolutionary Computing*, 2nd edn. Springer, Berlin (2007)
42. Spears, W.M.: *Evolutionary Algorithms: the Role of Mutation and Recombination*. Springer, New York (2000)
43. De Jong, K.A.: *Evolutionary Computation*. MIT Press, Cambridge, MA (2006)
44. Vose, M.D.: *The Simple Genetic Algorithm: Foundations and Theory*. MIT Press, Cambridge, MA (1999)
45. Schmitt, L., Nehaniv, C.L., Fujii, R.H.: Linear analysis of genetic algorithms. *Theor. Comput. Sci.* **200**, 101–134 (1998)
46. Schmitt, L.: Theory of genetic algorithms. *Theor. Comput. Sci.* **259**, 1–61 (2001)
47. Grefenstette, J.J. (ed.): *Proceedings of an International Conference on Genetic Algorithms and their applications*. Lawrence Erlbaum Associates, Hillsdale, NJ (1985)
48. Grefenstette, J.J. (ed.): *Proceedings of the 2nd International Conference on Genetic Algorithms*. Lawrence Erlbaum Associates, Hillsdale, NJ (1987)
49. Schaffer, J.D. (ed.): *Proceedings of 3rd International Conference on Genetic Algorithms*. Morgan Kaufmann, San Mateo, CA (1989)
50. Belew, R.K., Booker, L.B. (eds.): *Proceedings of 4th International Conference on Genetic Algorithms*. Morgan Kaufmann, San Mateo, CA (1991)
51. Forrest, S. (ed.): *Proceedings of 5th International Conference on Genetic Algorithms*. Morgan Kaufmann, San Mateo, CA (1993)
52. Eshelman, L.J. (ed.): *Proceedings of 6th International Conference on Genetic Algorithms*. Morgan Kaufmann, San Mateo, CA (1995)
53. Bäck, Th. (ed.): *Proceedings of 7th International Conference on Genetic Algorithms*. Morgan Kaufmann, San Francisco, CA (1997)

54. Angeline, P.J. (ed.): Proceedings of the 1999 Congress on Evolutionary Computation. IEEE Press, Piscataway, NJ (1999)
55. Zalzala, A. (ed.): Proceedings of the 2000 Congress on Evolutionary Computation. IEEE Press, Piscataway, NJ (2000)
56. Kim, J.-H. (ed.): Proceedings of the 2001 Congress on Evolutionary Computation. IEEE Press, Piscataway, NJ (2001)
57. Banzhaf, W., et al. (eds.): Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999). Morgan Kaufmann, San Francisco (1999)
58. Whitley, D., et al. (eds.): Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2000). Morgan Kaufmann, San Francisco (2000)
59. Spector, L., et al. (eds.): Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001). Morgan Kaufmann, San Francisco, CA (2001)
60. Langdon, W.B., et al.: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002). Morgan Kaufmann, San Francisco, CA (2002)
61. Cantú-Paz, E., et al.: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2003), LNCS 2723/2724. Springer, Berlin (2003)
62. Deb, K., et al. (eds.): Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004), LNCS 3102/3103. Springer, Berlin (2004)
63. Beyer, H-G., et al.: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2005). ACM Press, New York (2005)
64. Cattolico, M. (ed.): Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2006). ACM Press, New York (2006)
65. Lipson, H. (ed.): Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2007). ACM Press, New York (2007)
66. Ryan, C., et al.: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2008). ACM Press (2008)
67. Schwefel, H-P., Männer, R. (eds.): Parallel Problem-Solving from Nature. Springer, Berlin (1991)
68. Männer, R., Manderick, B. (eds.): Parallel Problem-Solving from Nature, 2. Elsevier Science Publishers, Amsterdam (1992)
69. Davidor, Y., Schwefel, H-P., Männer, R. (eds.): Parallel Problem-Solving from Nature, 3. Springer, Berlin (1994)
70. Voigt, H-M., et al. (eds.): Parallel Problem-Solving from Nature, 4, LNCS 1141. Springer, Berlin (1996)
71. Eiben, A.E., et al. (eds.): Parallel Problem-Solving from Nature, 5, LNCS 1498. Springer, Berlin (1998)
72. Schoenauer, M., et al. (eds.): Parallel Problem-Solving from Nature, 6, LNCS 1917. Springer, Berlin (2000)
73. Merelo Guervs, J.J., et al. (eds.): Parallel Problem-Solving from Nature, 7, LNCS 2439. Springer, Berlin (2002)
74. Yao, X., et al. (eds.): Parallel Problem-Solving from Nature, 8, LNCS 3242. Springer, Berlin (2004)
75. Runarsson, T.P., et al. (eds.): Parallel Problem-Solving from Nature, 9, LNCS 4193. Springer, Berlin (2006)
76. Rudolph, G., et al.: Parallel Problem-Solving from Nature, 10, LNCS 5199. Springer, Berlin (2008)
77. Albrecht, R.F., Reeves, C.R., Steele, N.C. (eds.): Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms. Springer, Vienna (1993)
78. Pearson, D.W., Steele, N.C., Albrecht, R.F. (eds.): Proceedings of the 2nd International Conference on Artificial Neural Networks and Genetic Algorithms. Springer, Vienna (1995)
79. Smith, G.D., Steele, N.C., Albrecht, R.F. (eds.): Proceedings of the 3rd International Conference on Artificial Neural Networks and Genetic Algorithms. Springer, Vienna (1997)

80. Dobnikar, A., Steele, N.C., Pearson, D.W., Albrecht, R.F. (eds.): Proceedings of the 4th International Conference on Artificial Neural Networks and Genetic Algorithms. Springer, Vienna (1999)
81. Kůrková, V., Steele, N.C., Neruda, R., Kárný, M. (eds.): Proceedings of the 5th International Conference on Artificial Neural Networks and Genetic Algorithms. Springer, Vienna (2001)
82. Pearson, D.W., Steele, N.C., Albrecht, R.F. (eds.): Proceedings of the 6th International Conference on Artificial Neural Networks and Genetic Algorithms. Springer, Vienna (2003)
83. Ribeiro, B., Albrecht, R.F., Steele, N.C., Dobnikar, A., Pearson, D.W., Steele, N.C. (eds.): Proceedings of the International Conference on Adaptive and Natural Computing Algorithms. Springer, Vienna (2005)
84. Beliczynski, B., Dzielinski, A., Iwanowski, M., Ribeiro, B. (eds.): Adaptive and Natural Computing Algorithms; Proceedings of ICANNGA 2007, LNCS 4431/4432. Springer, Berlin (2007)
85. Rawlins, G.J.E. (ed.): Foundations of Genetic Algorithms. Morgan Kaufmann, San Mateo, CA (1991)
86. Whitley, L.D. (ed.): Foundations of Genetic Algorithms 2. Morgan Kaufmann, San Mateo, CA (1993)
87. Whitley, D., Vose, M. (eds.): Foundations of Genetic Algorithms 3. Morgan Kaufmann, San Mateo, CA (1995)
88. Belew, R.K., Vose, M.D. (eds.): Foundations of Genetic Algorithms 4. Morgan Kaufmann, San Francisco, CA (1997)
89. Banzhaf, W., Reeves, C.R. (eds.): Foundations of Genetic Algorithms 5. Morgan Kaufmann, San Francisco, CA (1999)
90. Martin, W.N., Spears, W.M. (eds.): Foundations of Genetic Algorithms 6. Morgan Kaufmann, San Francisco, CA (2001)
91. De Jong, K.A., Poli, R., Rowe, J.E. (eds.): Foundations of Genetic Algorithms 7. Morgan Kaufmann, San Francisco, CA (2003)
92. Wright, A.H., et al. (eds.): Foundations of Genetic Algorithms 8, LNCS 3469. Springer, Berlin (2005)
93. Stephens, C.R., et al. (eds.): Foundations of Genetic Algorithms 9, LNCS 4436. Springer, Berlin (2007)
94. Goldberg, D.E.: Optimal Initial Population Size for Binary-Coded Genetic Algorithms. TCGA Report 85001. University of Alabama, Tuscaloosa (1985)
95. Goldberg, D.E.: Sizing populations for serial and parallel genetic algorithms. In: Schaffer, J.D. (ed.): Proceedings of 3rd International Conference on Genetic Algorithms. Morgan Kaufmann, San Mateo, CA, pp. 70–79 (1989)
96. Grefenstette, J.J.: Optimization of control parameters for genetic algorithms. IEEE-SMC **16**, 122–128 (1986)
97. Schaffer, J.D., Caruana, R.A., Eshelman, L.J., Das, R.: A study of control parameters affecting online performance of genetic algorithms for function optimization. In: [Schaffer, J.D. (ed.): Proceedings of 3rd International Conference on Genetic Algorithms. Morgan Kaufmann, San Mateo, CA (1989)], pp. 51–60 (1989)
98. Reeves, C.R.: Using genetic algorithms with small populations. In: [Forrest, S. (ed.): Proceedings of 5th International Conference on Genetic Algorithms. Morgan Kaufmann, San Mateo, CA (1993)], pp. 92–99 (1993)
99. Rees, J., Koehler, G.J.: An investigation of GA performance results for different cardinality alphabets. In: Davis, L.D., De Jong, K.A., Vose, M.D., Whitley, L.D. (eds.): Evolutionary Algorithms: IMA Volumes in Mathematics and its Applications, Vol. 111, Springer, New York, pp. 191–206 (1998) (1999)
100. Reeves, C.R.: A genetic algorithm for flowshop sequencing. Comput. Oper. Res. **22**, 5–13 (1995)
101. Ahuja, R.K., Orlin, J.B.: Developing fitter GAs. INFORMS J. Comput. **9**, 251–253 (1997)

102. Kapsalis, A., Smith, G.D., Rayward-Smith, V.J.: Solving the graphical steiner tree problem using genetic algorithms. *J. Oper. Res. Soc.* **44**, 397–406 (1993)
103. Levine, D.: GAs: A practitioner's view. *INFORMS J. Comput.* **9**, 256–257 (1997)
104. Ayutg, H., Koehler, G.J.: New stopping criterion for genetic algorithms. *Eur. J. Oper. Res.* **126**, 662–674 (2000)
105. Greenhalgh, D., Marshall, S.: Convergence criteria for genetic algorithms. *SIAM J. Comput.* **30**, 269–282 (2000)
106. Baker, J.E.: Reducing bias and inefficiency in the selection algorithm. In: [Grefenstette, J.J. (ed.): Proceedings of the 2nd International Conference on Genetic Algorithms. Lawrence Erlbaum Associates, Hillsdale, NJ (1987)], 14–21 (1987)
107. Lohr, S.L.: Sampling: Design and Analysis. Duxbury Press, Pacific Grove, CA (1999)
108. Hancock, P.J.B.: An empirical comparison of selection methods in evolutionary algorithms. In: Fogarty, T.C. (ed.) *Evolutionary Computing: AISB Workshop*, Leeds, UK, April 1994; Selected Papers, Springer, Berlin, pp. 80–94 (1994)
109. Whitley, D.: The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In: [Schaffer, J.D. (ed.): Proceedings of 3rd International Conference on Genetic Algorithms. Morgan Kaufmann, San Mateo, CA (1989)], pp. 116–121 (1989)
110. Saliby, E.: Descriptive sampling: A better approach to Monte Carlo simulation. *J. Oper. Res. Soc.* **41**, 1133–1142 (1990)
111. Nijenhuis, A., Wilf, H.S.: Combinatorial Algorithms for Computers and Calculators. Academic, New York (1978)
112. Skiena, S.S.: The Stony Brook Algorithm Repository. [http://www.cs.sunysb.edu/~algorith/major\\_section/1.3.shtml](http://www.cs.sunysb.edu/~algorith/major_section/1.3.shtml). last accessed 18 August 2010 (2000)
113. Eshelman, L.J., Caruana, R.A., Schaffer, J.D.: Biases in the crossover landscape. In: [Schaffer, J.D. (ed.): Proceedings of 3rd International Conference on Genetic Algorithms. Morgan Kaufmann, San Mateo, CA (1989)], pp. 10–19 (1989)
114. Syswerda, G. Uniform crossover in genetic algorithms. In: [Schaffer, J.D. (ed.): Proceedings of 3rd International Conference on Genetic Algorithms. Morgan Kaufmann, San Mateo, CA (1989)], 2–9 (1989)
115. De Jong, K.A., Spears, W.M.: A formal analysis of the role of multi-point crossover in genetic algorithms. *Ann. Math. AI* **5**, 1–26 (1992)
116. Booker, L.B.: Improving search in genetic algorithms. In: Davis, L. (ed.) (1987) *Genetic Algorithms and Simulated Annealing*. Morgan Kauffmann, Los Altos, CA, pp. 61–73 (1987)
117. Goldberg, D.E., Lingle, R.: Alleles, loci and the traveling salesman problem. In: [Grefenstette, J.J. (ed.): Proceedings of an International Conference on Genetic Algorithms and their applications. Lawrence Erlbaum Associates, Hillsdale, NJ (1985)], pp. 154–159 (1985)
118. Bremermann, H.J., Rogson, J., Salaff, S.: Global properties of evolution processes. In: Pattee, H.H. (ed.) *Natural Automata and Useful Simulations*, Spartan Books, Washington DC, pp. 3–42 (1966)
119. Fogel, D.B.: An overview of evolutionary programming. In: Davis, L.D., De Jong, K.A., Vose, M.D., Whitley, L.D. (eds.) *Evolutionary Algorithms: IMA Volumes in Mathematics and its Applications*, Vol. 111, Springer, New York, pp. 89–109 (1999)
120. Fogarty, T.C.: Varying the probability of mutation in the genetic algorithm. In: [Schaffer, J.D. (ed.): Proceedings of 3rd International Conference on Genetic Algorithms. Morgan Kaufmann, San Mateo, CA (1989)], pp. 104–109 (1989)
121. Eiben, A.E., Schut, M.C., de Wilde, A.R.: Is self-adaptation of selection pressure and population size possible? A case study. In: [Runarsson, T.P., et al. (eds.): *Parallel Problem-Solving from Nature*, 9, LNCS 4193. Springer, Berlin (2006)], pp. 900–909 (2006)
122. Goldberg, D.E., Deb, K.: A comparative analysis of selection schemes used in genetic algorithms. In: [Rawlins, G.J.E. (ed.): *Foundations of Genetic Algorithms*. Morgan Kaufmann, San Mateo, CA (1991)], pp. 69–93 (1991)

123. Radcliffe, N.J., George, F.A.W.: A study in set recombination. In: [Forrest, S. (ed.): Proceedings of 5th International Conference on Genetic Algorithms. Morgan Kaufmann, San Mateo, CA (1993)], pp. 23–30 (1993)
124. McKee, S., Reed, M.B.: An algorithm for the alignment of gas turbine components in aircraft. *IMA J. Math. Manag.* **1**, 133–144 (1987)
125. Radcliffe, N.J., Surry, P.D.: Formae and the variance of fitness. In: [Whitley, D., Vose, M. (eds.): Foundations of Genetic Algorithms 3. Morgan Kaufmann, San Mateo, CA (1995)], pp. 51–72 (1995)
126. Fox, B.R., McMahon, M.B.: Genetic operators for sequencing problems. In: [Rawlins, G.J.E. (ed.): Foundations of Genetic Algorithms. Morgan Kaufmann, San Mateo, CA (1991)], pp. 284–300 (1991)
127. Poon, P.W., Carter, J.N.: Genetic algorithm crossover operators for ordering applications. *Comput. Oper. Res.* **22**, 135–147 (1995)
128. Reeves, C.R., Yamada, T.: Genetic algorithms, path relinking and the flowshop sequencing problem. *Evol. Comput.* **6**, 45–60 (1998)
129. Ross, P. `srandom()` anomaly. *Genetic Algorithms Digest*, [http://www.aridolan.com/maillists/mlframes\\_ns.html](http://www.aridolan.com/maillists/mlframes_ns.html). last accessed 18 August 2010, **11:23** (1997)
130. Press, W.H., Teukolsky, S.A., Vetterling, W.T., Flannery, B.P.: *Numerical Recipes in C : The Art of Scientific Computing*. Cambridge University Press, Cambridge, UK (1992)
131. Reeves, C.R.: The ‘crossover landscape’ and the Hamming landscape for binary search spaces. In: [De Jong, K.A., Poli, R., Rowe, J.E. (eds.): Foundations of Genetic Algorithms 7. Morgan Kaufmann, San Francisco, CA (2003)], pp. 81–97 (2002)



# Chapter 6

## A Modern Introduction to Memetic Algorithms

Pablo Moscato and Carlos Cotta

**Abstract** Memetic algorithms are optimization techniques based on the synergistic combination of ideas taken from different algorithmic solvers, such as population-based search (as in evolutionary techniques) and local search (as in gradient-ascent techniques). After providing some historical notes on the origins of memetic algorithms, this work shows the general structure of these techniques, including some guidelines for their design. Some advanced topics such as multiobjective optimization, self-adaptation, and hybridization with complete techniques (e.g., branch-and-bound) are subsequently addressed. This chapter finishes with an overview of the numerous applications of these techniques and a sketch of the current development trends in this area.

### 6.1 Introduction and Historical Notes

The generic denomination of “memetic algorithms” (MAs) is used to encompass a broad class of metaheuristics (i.e., general purpose methods aimed to guide an underlying heuristic). The method is based on a population of agents and proved to be of practical success in a variety of problem domains and in particular for the approximate solution of NP-hard optimization problems.

Unlike traditional evolutionary computation (EC) methods, MAs are intrinsically concerned with exploiting *all available knowledge* about the problem under study.

---

Pablo Moscato

Centre for Bioinformatics, Biomarker Discovery and Information-based Medicine, The University of Newcastle, University Drive, Callaghan, NSW 2308, Australia  
e-mail: pablo.moscato@newcastle.edu.au

Carlos Cotta

Departamento de Lenguajes y Ciencias de la Computación, Escuela Técnica Superior de Ingeniería Informática, Universidad de Málaga, Campus de Teatinos, 29071 - Málaga, Spain  
e-mail: ccottap@lcc.uma.es

The incorporation of problem domain knowledge is not an optional mechanism, but a fundamental feature that characterizes MAs. This functioning philosophy is perfectly illustrated by the term “memetic.” Coined by R. Dawkins [62], the word “meme” denotes an analogous to the gene in the context of cultural evolution [177]. In Dawkins’ words:

Examples of memes are tunes, ideas, catch-phrases, clothes fashions, ways of making pots or of building arches. Just as genes propagate themselves in the gene pool by leaping from body to body via sperms or eggs, so memes propagate themselves in the meme pool by leaping from brain to brain via a process which, in the broad sense, can be called imitation.

This characterization of a meme suggests that in cultural evolution processes, information is not simply transmitted unaltered between individuals. In contrast, it is processed and enhanced by the communicating parts. This enhancement is accomplished in MAs by incorporating heuristics, approximation algorithms, local search techniques, specialized recombination operators, truncated exact methods, etc. In essence, most MAs can be interpreted as a search strategy in which a population of optimizing agents cooperate and compete [202]. The success of MAs can probably be explained as being a direct consequence of the *synergy* of the different search approaches they incorporate.

The most crucial and distinctive feature of MAs, the inclusion of problem knowledge mentioned above, is also supported by strong theoretical results. As Hart and Belew [108] initially stated and Wolpert and Macready [276] later popularized in the so-called No-Free-Lunch Theorem, a search algorithm strictly performs in accordance with the amount and quality of the problem knowledge they incorporate. This fact clearly underpins the exploitation of problem knowledge intrinsic to MAs. Given that the term *hybridization* is often used to denote the process of incorporating problem knowledge [39], it is not surprising that MAs are sometimes called “Hybrid Evolutionary Algorithms” [61] (hybrid EAs) as well. One of the first algorithms to which the MA label was assigned dates from 1988 [202] and was regarded by many as a hybrid of *traditional* genetic algorithms (GAs) and *simulated annealing* (SA). Part of the initial motivation was to find a way out of the limitations of both techniques on a well-studied combinatorial optimization problem, the MIN EUCLIDEAN TRAVELING SALESMAN problem (MIN ETSP). According to the authors, the original inspiration came from *computer game tournaments* [111] used to study “the evolution of cooperation” [8, 190]. That approach had several features which anticipated many current algorithms in practice today. The competitive phase of the algorithm was based on the new allocation of search points in configuration phase, a process involving a “battle” for survival followed by the so-called cloning, which has a strong similarity with “go with the winners” algorithms [4, 213]. The cooperative phase followed by local search may be better named “go with the local winners” since the optimizing *agents* were arranged with a topology of a two-dimensional toroidal lattice. After initial computer experiments, an insight was derived on the particular relevance that the “spatial” organization, when coupled with an appropriate set of rules, had for the overall performance of population search processes. A few months later, Moscato and Norman discovered that they shared similar views with other researchers [100, 185] and other authors proposing

“island models” for GAs. Spacialization is now being recognized as the “catalyzer” responsible for a variety of phenomena [189, 190]. This is an important research issue, currently only understood in a rather heuristic way. However, some proper undecidability results have been obtained for related problems [102] giving some hope to a more formal treatment.

Less than a year later, in 1989, Moscato and Norman identified several authors who were also pioneering the introduction of heuristics to improve the solutions before recombining them [99, 186] (see other references and the discussion in [177]). Particularly coming from the GA field, several authors were introducing *problem domain knowledge* in a variety of ways. In [177] the denomination of “memetic algorithms” was introduced for the first time. It was also suggested that *cultural evolution* can be a better working metaphor for these metaheuristics to avoid “biologically constrained” thinking that was restricting progress at that time.

Ten years later, albeit unfortunately under different names, MAs have become an important optimization approach, with several successes in a variety of classical NP-hard optimization problems. We aim to provide an updated and self-contained introduction to MAs, focusing on their technical innards and formal features, but without loosing the perspective of their practical application and open research issues.

## 6.2 Memetic Algorithms

Before proceeding to the description of MAs, it is necessary to provide some basic concepts and definitions. Several notions introduced in Section 6.1 are strongly related to the field of computational complexity. Nevertheless, they may be presented in a slightly different way and pace for the sake of the subsequent development. These basic concepts will give rise to the notions of local search and population-based search, upon which MAs are founded. This latter class of search settles the scenario for *recombination*, a crucial mechanism in the functioning of MAs that will be studied to some depth. Finally, a basic algorithmic template and some guidelines for designing MAs will be presented.

### 6.2.1 Basic Concepts

An *algorithm* is a detailed step-by-step procedure for solving a *computational problem*. A computational problem  $P$  denotes a class of algorithmically doable tasks, and it has an input domain set of *instances* denoted  $I_P$ . For each instance  $x \in I_P$ , there is an associated set  $\text{sol}_P(x)$  which denotes the *feasible* solutions for problem  $P$  given instance  $x$ . The set  $\text{sol}_P(x)$  is also known as the set of *acceptable* or *valid* solutions.

We are expected to deliver an algorithm that solves problem  $P$ ; this means that our algorithm, given instance  $x \in I_P$ , must return at least one element  $y$  from a set of answers  $ansp(x)$  (also called *given solutions*) that satisfies the requirements of the problem. This is the first design issue to face. To be precise, depending on the kind of answers expected, computational problems can be classified into different categories; for instance:

- finding *all* solutions in  $solp(x)$ , i.e., *enumeration* problems.
- counting *how many* solutions exist in  $solp(x)$ , i.e., *counting* problems.
- determining whether the set  $solp(x)$  is *empty or not*, i.e., *decision* problems.
- finding a solution in  $solp(x)$  maximizing or minimizing a given function, i.e., *optimization* problems.

In this chapter, we will focus on the last possibility, that is, a problem will be considered *solved* by finding a certain feasible solution, i.e., either finding an *optimal*  $y \in solp(x)$  or giving an indication that no such feasible solution exists. It is thus convenient in many situations to define a Boolean *feasibility* function  $feasible_p(x, y)$  in order to identify whether a given solution  $y \in ansp(x)$  is acceptable for an instance  $x \in I_P$  of a computational problem  $P$ , i.e., checking if  $y \in solp(x)$ .

An algorithm is said to *solve* problem  $P$  if it can fulfill this condition for any given instance  $x \in I_P$ . This definition is certainly too broad, so a more restrictive characterization for our problems of interest is necessary. This characterization is provided by restricting ourselves to the so-called combinatorial optimization problems. These constitute a special subclass of computational problems in which for each instance  $x \in I_P$ :

- the cardinality of  $solp(x)$  is finite.
- each solution  $y \in solp(x)$  has a *goodness integer value*  $m_p(y, x)$  obtained by means of an associated *objective function*  $m_p$ .
- a partial order  $\prec_p$  is defined over the set of goodness values returned by the objective function, allowing determining which of two goodness values is preferable.

An instance  $x \in I_P$  of a combinatorial optimization problem  $P$  is solved by finding the best solution  $y^* \in solp(x)$ , i.e., finding a solution  $y^*$  such that no other solution  $y \prec_p y^*$  exists if  $solp(x)$  is not empty. It is very common to have  $\prec_p$  defining a total order. In this case, the best solution is the one that maximizes (or minimizes) the objective function.

As an example of a combinatorial optimization problem consider the 0-1 MULTIPLE KNAPSACK PROBLEM (0-1 MKP). Each instance  $x$  of this problem is defined by a vector of profits  $V = \{v_0, \dots, v_{n-1}\}$ , a vector of capacities  $C = \{c_0, \dots, c_{m-1}\}$ , and a matrix of capacity constraints  $M = \{m_{ij} : 0 \leq i < m, 0 \leq j < n\}$ . Intuitively, the problem consists in selecting a set of objects so as to maximize the profit of this set without violating the capacity constraints. If the objects are indexed with the elements of the set  $\mathbb{N}_n = \{0, 1, \dots, n-1\}$ , the answer set  $ansp(x)$  for an instance  $x$  is simply the power set of  $\mathbb{N}_n$ , that is, each subset of  $\mathbb{N}_n$  is a possible answer. Furthermore, the set of feasible answers  $solp(x)$  is composed of those subsets whose

incidence vector  $B$  verifies  $M \cdot B \leq C$ . Finally, the objective function is defined as  $m_P(y, x) = \sum_{i \in y} v_i$ , i.e., the sum of profits for all selected objects, the goal being to maximize this value.

Note that, associated with a combinatorial optimization problem, we can define its *decisional* version. To formulate the decision problem, an integer goodness value  $K$  is considered, and instead of trying to find the best solution of instance  $x$ , we ask whether  $x$  has a solution whose goodness is equal or better than  $K$ . In the above example, we could ask whether a feasible solution  $y$  exists such that its associated profit is equal or better than  $K$ .

### 6.2.2 Search Landscapes

As mentioned above, having defined the concept of combinatorial optimization problem the goal is finding at least one of the optimal solutions for a given instance. For this purpose, a search algorithm must be used. Before discussing search algorithms, three entities must be discussed. These are the *search space*, the *neighborhood relation*, and the *guiding function*. It is important to consider that, for any given computational problem, these three entities can be instantiated in several ways, giving rise to different optimization tasks.

Let us start by defining the concept of search space for a combinatorial problem  $P$ . To do so, we consider a set  $\mathcal{S}_P(x)$ , whose elements have the following properties:

- Each element  $s \in \mathcal{S}_P(x)$  represents at least one answer in  $ans_P(x)$ .
- For decision problems: at least one element of  $sol_P(x)$  that stands for a “Yes” answer must be represented by one element in  $\mathcal{S}_P(x)$ .
- For optimization problems: at least one *optimal* element  $y^*$  of  $sol_P(x)$  is represented by one element in  $\mathcal{S}_P(x)$ .

Each element of  $\mathcal{S}_P(x)$  will be termed a *configuration*, being related to an answer in  $ans_P(x)$  by a *growth function*  $g : \mathcal{S}_P(x) \rightarrow ans_P(x)$ . Note that the first requirement refers to  $ans_P(x)$  and not to  $sol_P(x)$ , i.e., some configurations in the search space may correspond to infeasible solutions. Thus, the search algorithm may need to be prepared to deal with this fact. If these requirements have been achieved, we say that we have a *valid representation* or *valid formulation* of the problem. For simplicity, we will just write  $\mathcal{S}$  to refer to  $\mathcal{S}_P(x)$  when  $x$  and  $P$  are clear from the context. People using biologically inspired metaphors like to call  $\mathcal{S}_P(x)$  the *genotype space* and  $ans_P(x)$  denotes the *phenotype space*, so we appropriately refer to  $g$  as the *growth function*.

To illustrate this notion of search space, consider again the case of the 0-1 MKP. Since solutions in  $ans_P(x)$  are subsets of  $\mathbb{N}_n$ , we can define the search space as the set of  $n$ -dimensional binary vectors. Each vector will represent the incidence vector of a certain subset, i.e., the growth function  $g$  is defined as  $g(s) = g(b_0 b_1 \cdots b_{n-1}) = \{i \mid b_i = 1\}$ . As mentioned above, many binary vectors may correspond to infeasible sets of objects. Another possibility is defining the search space as the set of

permutations of elements in  $\mathbb{N}_n$  [101]. In this case, the growth function may consist of applying a greedy construction algorithm, considering objects in the order provided by the permutation. Unlike the binary search space previously mentioned, all configurations represent feasible solutions in this case.

The role of the search space is to provide a “ground” where the search algorithm will act. Important properties of the search space that affect the dynamics of the search algorithm are related to the accessibility relationships between the configurations. These relationships are dependent on a *neighborhood function*  $\mathcal{N} : \mathcal{S} \rightarrow 2^{\mathcal{S}}$ . This function assigns to each element  $s \in S$  a set  $\mathcal{N}(s) \subseteq S$  of neighboring configurations of  $s$ . The set  $\mathcal{N}(s)$  is called the *neighborhood* of  $s$  and each member  $s' \in \mathcal{N}(s)$  is called a *neighbor* of  $s$ .

It must be noted that the neighborhood depends on the instance, so the notation  $\mathcal{N}(s)$  is a simplified form of  $\mathcal{N}_P(s, x)$  since it is clear from the context. The elements of  $\mathcal{N}(s)$  need not be listed explicitly. In fact, it is very usual to define them *implicitly* by referring to a set of possible *moves*, which define *transitions* between configurations. Moves are usually defined as “local” modifications of some part of  $s$ , where “locality” refers to the fact that the move is done on a single solution to obtain another single solution. This “locality” is one of the key ingredients of *local search*, and actually it has also given the name to the whole search paradigm.

As examples of concrete neighborhood definitions, consider the two representations of solutions for the 0-1 MKP presented above. In the first case (binary representation), moves can be defined as changing the values of a number of bits. If just one bit is modified at a time, the resulting neighborhood structure is the  $n$ -dimensional binary hypercube. In the second case (permutation representation), moves can be defined as the interchange of two positions in the permutation. Thus, two configurations are neighboring if, and only if, they differ in exactly two positions.

This definition of locality presented above is not necessarily related to “closeness” under some kind of distance relationship between configurations (except in the tautological situation in which the distance between two configurations  $s$  and  $s'$  is defined as the number of moves needed to reach  $s'$  from  $s$ ). As a matter of fact, it is possible to give common examples of very complex neighborhood definitions unrelated to intuitive distance measures.

An important feature that must be considered when selecting the class of moves to be used in the search algorithm is its “ergodicity,” that is the ability, given any  $s \in S$ , to find a sequence of moves that can reach *all other* configurations  $s' \in S$ . In many situations this property is self-evident and no explicit demonstration is required. It is important since even if we have a valid representation (recall the definition above), it is necessary to guarantee a priori that at least one optimal solution is reachable from any given initial solution. Again, consider the binary representation of solutions for a 0-1 MKP instance. If moves are defined as single bit-flips, it is easily seen that any configuration  $s'$  can be reached from another configuration  $s$  in exactly  $h$  moves, where  $h$  is the Hamming distance between these configurations. This is not always the case though.

The last entity that must be defined is the *guiding function*. To do so, we require a set  $\mathcal{F}$  whose elements are termed *fitness* values (typically  $\mathcal{F} \equiv \mathbb{R}$ ) and a partial order  $\prec_{\mathcal{F}}$  on  $\mathcal{F}$  (typically, but not always,  $\prec_{\mathcal{F}} \equiv <$ ). The guiding function is defined as a function  $F_g : \mathcal{S} \rightarrow \mathcal{F}$  that associates to each configuration  $s \in \mathcal{S}$  a value  $F_g(s)$  that assesses the quality of the solution. The behavior of the search algorithm will be “controlled” by these fitness values.

Note that for optimization problems there is an obvious direct connection between the guiding function  $F_g$  and the objective function  $m_P$  (and hence between partial orders  $\prec_P$  and  $\prec_{\mathcal{F}}$ ). As a matter of fact, it is very common to enforce this relationship to the point that both terms are usually considered equivalent. However, this equivalence is not necessary and, in many situations, not even desirable. For decision problems, since a solution is a “Yes” or “No” answer, associated guiding functions usually take the form of *distance to satisfiability*.

A typical example is the BOOLEAN SATISFIABILITY PROBLEM, i.e., determining whether a Boolean expression in conjunctive normal form is satisfiable. In this case, solutions are assignments of Boolean values to variables, and the objective function  $m_P$  is a binary function returning 1 if the solution satisfies the Boolean expression, and returning 0 otherwise. This objective function could be used as guiding function. However, a much more typical choice is to use the number of satisfied clauses in the current configuration as guiding function, i.e.,  $F_g(s) = \sum_i f_i(s)$ , the sum over clause indexes  $i$  of  $f_i(s)$ , defined as  $f_i(s) = 0$  for a yet unsatisfied clause  $i$ , and  $f_i(s) = 1$  if the clause  $i$  is satisfied. Hence, the goal is to maximize this number. Note that the guiding function in this case is the objective function of the associated NP-hard optimization problem called MAX SAT.

The above differentiation between objective function and guiding function is also very important in the context of constrained optimization problems, i.e., problems for which, in general,  $\text{sol}_P(x)$  is chosen to be a proper subset of  $\text{ans}_P(x)$ . Since the growth function establishes a mapping from  $\mathcal{S}$  to  $\text{ans}_P(x)$ , the search algorithm might need processing both feasible solutions (whose goodness values are well defined) and infeasible solutions (whose goodness values are ill-defined in general). In many implementations of MAs for these problems, a guiding function is defined as a weighted sum of the value of the objective function and the distance to feasibility (which accounts for the constraints). Typically, a higher weight is assigned to the constraints, so as to give preference to feasibility over optimality. Several other remedies to this problem abound, including resorting to multiobjective techniques.

The combination of a certain problem instance and the three entities defined above induces a so-called fitness landscape [127]. Essentially, a fitness landscape can be defined as a weighted digraph, in which the vertices are configurations of the search space  $\mathcal{S}$ , and the arcs connect neighboring configurations. The weights are the differences between the guiding function values of the two endpoint configurations. The search can thus be seen as the process of “navigating” the fitness landscape using the information provided by the guiding function. This is a very powerful metaphor; it allows interpretations in terms of well-known topographical objects such as *peaks*, *valleys*, and *mesas*; it is of great utility to visualize the search progress and to grasp factors affecting the performance of the process. In particular,

the important notion of *local* optimum is associated with this definition of fitness landscape. To be precise, a local optimum is a vertex of the fitness landscape whose guiding function value is better than the values of all its neighbors. Note that different moves define different neighborhoods and hence different fitness landscapes, even when the same problem instance is considered. For this reason, the notion of local optimum is not intrinsic to a problem instance as it is, sometimes, erroneously considered.

### 6.2.3 Local vs. Population-Based Search

The definitions presented in Section 6.2.2 naturally lead to the notion of *local search algorithm*. A local search algorithm starts from a configuration  $s_0 \in \mathcal{S}$ , generated at random or constructed by some other algorithm. Subsequently, it iterates using at each step a transition based on the neighborhood of the current configuration. Transitions leading to preferable (according to the partial order  $\prec_{\mathcal{F}}$ ) configurations are accepted, i.e., the newly generated configuration turns to be the current configuration in the next step. Otherwise, the current configuration is kept. This process is repeated until a certain termination criterion is met. Typical criteria are the realization of a pre-specified number of iterations, not having found any improvement in the last  $m$  iterations, or even more complex mechanisms based on estimating the probability of being at a local optimum [44]. Due to these characteristics, the approach is metaphorically called “hill climbing.” The whole process is sketched in Algorithm 1.

---

**Algorithm 1** A local search algorithm

---

```

1 Procedure Local-Search-Engine (current);
2 begin
3   repeat
4     new  $\leftarrow$  GenerateNeighbor(current);
5     if  $F_g(\text{new}) \prec_{\mathcal{F}} F_g(\text{current})$  then
6       | current  $\leftarrow$  new;
7     endif
8   until TerminationCriterion();
9   return current;
10 end

```

---

The selection of the particular type of moves (also known as *mutation* in the context of GAs) to use does certainly depend on the specific characteristics of the problem and the representation chosen. There is no general advice for this, since it is a matter of the available computer time for the whole process as well as other algorithmic decisions that include ease of coding, etc. In some cases some moves are conspicuous, for example, it can be the change of the value of one single variable or

the swap of the values of two different variables. Sometimes the “step” may also be composed of a chain of transitions. For instance, in relation with MAs, Radcliffe and Surry introduced the concept of *Binomial Minimal Mutation*, where the number of mutations to perform is selected according to a certain binomial distribution [229]. In the context of fitness landscapes, this is equivalent to a redefinition of the neighborhood relation, considering two configurations as neighbors when there exists a chain of transitions connecting them.

Local search algorithms are thus characterized by keeping a single configuration at a time. The immediate generalization of this behavior is the simultaneous maintenance of  $k$  ( $k \geq 2$ ) configurations. The term *population-based* search algorithms has been coined to denote search techniques behaving this way.

The availability of several configurations at a time allows the use of new powerful mechanisms for traversing the fitness landscape in addition to the standard mutation operator. The most popular of these mechanisms, the recombination operator, will be studied in more depth in Section 6.2.4. In any case, note that the general functioning of population-based search techniques is very similar to the pseudocode depicted in Algorithm 1. As a matter of fact, a population-based algorithm can be imagined as a procedure in which we sequentially visit vertices of a hypergraph. Each vertex of the hypergraph represents a set of configurations in  $\mathcal{S}_P(x)$ , i.e., a population. The next vertex to be visited, i.e., the new population, can be established according to the composition of the neighborhoods of the different transition mechanisms used in the population algorithm. Despite the analogy with local search, it is widely accepted in the scientific literature to apply the denomination “local” just to one-configuration-at-a-time search algorithms. For this reason, the term “local” will be used with this interpretation in the remainder of the chapter.

### 6.2.4 Recombination

As mentioned in Section 6.2.3, local search is based on the application of a mutation operator to a single configuration. Despite the apparent simplicity of this mechanism, “mutation-based” local search has revealed itself to be a very powerful mechanism for obtaining good quality solutions for NP-hard problems. For this reason, some researchers have tried to provide a more theoretically solid background to this class of search. In this line, it is worth mentioning the definition of the *Polynomial Local Search* class (PLS) by Johnson et al. [126]. Basically, this complexity class comprises a problem and an associated search landscape such that we can decide in polynomial time if we can find a better solution in the neighborhood. Unfortunately, it is very likely that no NP-hard problem is contained in class PLS, since that would imply that  $\text{NP}=\text{co-NP}$  [279], a conjecture usually assumed to be false. This fact has justified the quest for additional search mechanisms to be used as stand-alone operators or as complements to standard mutation.

In this line, recall that population-based search allowed the definition of generalized move operators termed *recombination* operators. In essence, recombination can

be defined as a process in which a set  $S_{\text{par}}$  of  $n$  configurations (informally referred to as “parents”) are manipulated to create a set  $S_{\text{desc}} \subseteq \text{sol}_P(x)$  of  $m$  new configurations (informally termed “descendants”). The creation of these descendants involves the identification and combination of features extracted from the parents.

At this point, it is possible to consider properties of interest that can be exhibited by recombination operators [229]. The first property, *respect*, represents the exploitation side of recombination. A recombination operator is said to be *respectful*, regarding a particular type of features of the configurations, if, and only if, it generates descendants carrying all basic features common to all parents. Note that, if all parent configurations are identical, a respectful recombination operator is forced to return the same configuration as a descendant. This property is termed *purity* and can be achieved even when the recombination operator is not generally respectful.

On the other hand, *assortment* represents the exploratory side of recombination. A recombination operator is said to be *properly assorting* if, and only if, it can generate descendants carrying any combination of compatible features taken from the parents. The assortment is said to be *weak* if it is necessary to perform several recombinations within the offspring to achieve this effect.

Finally, *transmission* is a very important property that captures the intuitive role of recombination. An operator is said to be transmitting if every feature exhibited by the offspring is present in at least one of the parents. Thus, a transmitting recombination operator combines the information present in the parents but does not introduce new information. This latter task is usually left to the mutation operator. For this reason, a non-transmitting recombination operator is said to introduce *implicit mutation*.

The three properties above suffice to describe the abstract input/output behavior of a recombination operator regarding some particular features. It provides a characterization of the possible descendants that can be produced by the operator. Nevertheless, there exist other aspects of the functioning of recombination that must be studied. In particular, it is interesting to consider how the construction of  $\mathcal{S}_{\text{desc}}$  is approached.

First of all, a recombination operator is said to be *blind* if it has no other input than  $\mathcal{S}_{\text{par}}$ , i.e., it does not use any information from the problem instance. This definition is certainly very restrictive and hence is sometimes relaxed as to allow the recombination operator to use information regarding the problem constraints (so as to construct feasible descendants) and possibly the fitness values of configurations  $y \in \mathcal{S}_{\text{par}}$  (so as to bias the generation of descendants toward the best parents). A typical example of a blind recombination operator is the classical *Uniform crossover* [253]. This operator is defined on search spaces  $\mathcal{S} \equiv \Sigma^n$ , i.e., strings of  $n$  symbols taken from an alphabet  $\Sigma$ . The construction of the descendant is done by randomly selecting at each position one of the symbols appearing in that position in any of the parents. This random selection can be totally uniform or can be biased according to the fitness values of the parents as mentioned before. Furthermore, the selection can be done so as to enforce feasibility (e.g., consider the binary representation of solutions in the 0-1 MKP). Note that, in this case, the resulting operator is neither respectful nor transmitting in general.

The use of blind recombination operators has been usually justified on the grounds of not introducing excessive bias in the search algorithm, thus preventing extremely fast convergence to suboptimal solutions. This is questionable though. First, note that the behavior of the algorithm is in fact biased by the choice of representation and the mechanics of the particular operators. Second, there exist widely known mechanisms (e.g., spatial isolation) to hinder these problems. Finally, it can be better to quickly obtain a suboptimal solution and restart the algorithm than using blind operators for a long time in pursuit of an asymptotically optimal behavior (not even guaranteed in most cases).

Recombination operators that use problem knowledge are commonly termed *heuristic* or *hybrid*. In these operators, problem information is utilized to guide the process of constructing the descendants. This can be done in a plethora of ways for each problem, so it is difficult to provide a taxonomy of heuristic recombination operators. Nevertheless, there exist two main aspects into which problem knowledge can be injected: the selection of the parental features that will be transmitted to the descendant and the selection of non-parental features that will be added to it. A heuristic recombination operator can focus in one of these aspects or in both of them simultaneously.

As an example of a heuristic recombination operator focusing on the first aspect, dynamically optimal recombination (DOR) [53] can be mentioned. This operator explores the *dynamic potential* (i.e., the set of possible children) of the configurations being recombined, so as to find the best member of this set (note that, since configurations in the dynamic potential are entirely composed of features taken from any of the parents, this is a transmitting operator). This exploration is done using a subordinate complete algorithm, and its goal is thus to find the best combination of parental features giving rise to a feasible child. Hence, this operator is monotonic in the sense that any child generated is at least as good as the best parent.

As examples of heuristic recombination operators concentrating on the selection of non-parental features, one can cite the *patching-by-forma-completion* operators proposed by Radcliffe and Surry [228]. These operators are based on generating an incomplete child using a non-heuristic procedure (e.g., the RAR $\omega$  operator [227]) and then completing the child using either a local hill climbing procedure restricted to non-specified features (*locally optimal forma completion*) or a global search procedure that finds the globally best solution carrying the specified features (*globally optimal forma completion*). Note the similarity of this latter approach with DOR.

Finally, there exist some operators trying to exploit knowledge in both of the above aspects. A distinguished example is the *Edge Assembly Crossover* (EAX) [188]. EAX is a specialized operator for the TSP (both for symmetric and for asymmetric instances) in which the construction of the child comprises two phases: the first one involves the generation of an incomplete child via the so-called E-sets (subtours composed of alternating edges from each parent); subsequently, these subtours are merged into a single feasible subtour using a greedy repair algorithm. The authors of this operator reported impressive results in terms of accuracy and speed. It has some similarities with the recombination operator proposed in [178].

A final comment must be made in relation to the computational complexity of recombination. It is clear that combining the features of several solutions is in general computationally more expensive than modifying a single solution (i.e., a mutation). Furthermore, the recombination operation will be usually invoked a large number of times. For this reason, it is convenient (and in many situations mandatory) to keep it at a low computational cost. A reasonable guideline is to consider an  $O(N \log N)$  upper bound for its complexity, where  $N$  is the size of the input (the set  $S_{par}$  and the problem instance  $x$ ). Such limit is easily affordable for blind recombination operators, which are called *crossover*, a reasonable name to convey their low complexity (yet not always used in this context). However, this limit can be relatively astringent in the case of heuristic recombination, mainly when epistasis (non-additive inter-feature influence on the fitness value) is involved. This admits several solutions depending on the particular heuristic used. For example, DOR has exponential worst case behavior, but it can be made affordable by picking larger pieces of information from each parent (the larger the size of these pieces of information, the lower the number of them needed to complete the child) [52]. In any case, consider that heuristic recombination operators provide better solutions than blind recombination operators, and hence they need not be invoked the same number of times.

### 6.2.5 A Memetic Algorithm Template

In light of the above considerations, it is possible to provide a general template for a memetic algorithm. As mentioned in Section 6.2.3, this template is very similar to that of a local search procedure acting on a set of  $|pop| \geq 2$  configurations. This is shown in Algorithm 2.

---

#### Algorithm 2 A population-based search algorithm

---

```

1 Procedure Population-Based-Search-Engine;
2 begin
3   Initialize pop using GenerateInitialPopulation();
4   repeat
5     newpop ← GenerateNewPopulation(pop);
6     pop ← UpdatePopulation (pop, newpop);
7     if pop has converged then
8       | pop ← RestartPopulation(pop);
9     endif
10    until TerminationCriterion();
11 end

```

---

This template requires some explanation. First of all, the `GenerateInitialPopulation` procedure is responsible for creating the initial set of  $|pop|$  configurations. This can be done by simply generating  $|pop|$  random configurations or by using a more sophisticated seeding mechanism (for instance, some constructive heuristic),

by means of which high-quality configurations are injected in the initial population [252]. Another possibility is to use the Local-Search-Engine presented in Section 6.2.3 as shown in Algorithm 3.

---

**Algorithm 3** Injecting high-quality solutions in the initial population.

---

```

1 Procedure GenerateInitialPopulation;
2 begin
3   Initialize pop using EmptyPopulation();
4   for  $j \leftarrow 1$  to popsize do
5      $i \leftarrow$  GenerateRandomConfiguration();
6      $i \leftarrow$  Local-Search-Engine ( $i$ );
7     InsertInPopulation individual  $i$  to pop;
8   endfor
9   return pop;
10 end
```

---

As for the TerminationCriterion function, it can be defined very similarly to the case of Local Search, i.e., setting a limit on the total number of iterations, reaching a maximum number of iterations without improvement, or having performed a certain number of population restarts.

The GenerateNewPopulation procedure is at the core of memetic algorithms. Essentially, this procedure can be seen as a pipelined process comprising  $n_{op}$  stages. Each of these stages consists of taking  $arity_{in}^j$  configurations from the previous stage, generating  $arity_{out}^j$  new configurations by applying an operator  $op^j$ . This pipeline is restricted to have  $arity_{in}^1 = popsize. The whole process is sketched in Algorithm 4.$

---

**Algorithm 4** The pipelined GenerateNewPopulation procedure.

---

```

1 Procedure GenerateNewPopulation (pop);
2 begin
3    $buffer^0 \leftarrow pop$ ;
4   for  $j \leftarrow 1$  to  $n_{op}$  do
5     Initialize  $buffer^j$  using EmptyPopulation();
6   endfor
7   for  $j \leftarrow 1$  to  $n_{op}$  do
8      $S_{par}^j \leftarrow$  ExtractFromBuffer ( $buffer^{j-1}$ ,  $arity_{in}^j$ );
9      $S_{desc}^j \leftarrow$  ApplyOperator ( $op^j$ ,  $S_{par}^j$ );
10    for  $z \leftarrow 1$  to  $arity_{out}^j$  do
11      InsertInPopulation individual  $S_{desc}^j[z]$  to  $buffer^j$ ;
12    endfor
13  endfor
14  return  $buffer^{n_{op}}$ ;
15 end
```

---

This template for the `GenerateNewPopulation` procedure is usually instantiated in GAs by letting  $n_{\text{op}} = 3$ , using a selection, a recombination, and a mutation operator. Traditionally, mutation is applied after recombination, i.e., on each child generated by the recombination operator. However, if a heuristic recombination operator is being used, it may be more convenient to apply mutation *before* recombination. Since the purpose of mutation is simply to introduce new features in the configuration pool, using it in advance is also possible. Furthermore, the *smart* feature combination performed by the heuristic operator would not be disturbed this way.

This situation is slightly different in MAs. In this case, it is very common to let  $n_{\text{op}} = 5$ , inserting a Local-Search-Engine right after applying  $op^2$  and  $op^4$  (respectively, recombination and mutation). Due to the local optimization performed after mutation, their combined effect (i.e., mutation + local search) cannot be regarded as a simple disruption of a computationally demanding recombination. Note also that the interplay between mutation and local search requires the former to be different than the neighborhood structure used in the latter; otherwise mutations can be readily reverted by local search, and their usefulness would be negligible.

The `UpdatePopulation` procedure is used to reconstruct the current population using the old population  $pop$  and the newly generated population  $newpop$ . Borrowing the terminology from the evolution strategy [230, 238] community, there exist two main possibilities to carry on this reconstruction: the *plus* strategy and the *comma* strategy. In the former, the current population is constructed by taking the best  $popsize$  configurations from  $pop \cup newpop$ . As to the latter, the best  $popsize$  configurations are taken just from  $newpop$ . In this case, it is required to have  $|newpop| > popszie$ , so as to put some selective pressure on the process (the bigger the  $|newpop|/popszie$  ratio, the stronger the pressure). Otherwise, the search would reduce to a random wandering through  $\mathcal{S}$ .

There are a number of studies regarding appropriate choices for the `UpdatePopulation` procedure (see, e.g., [9]). As a general guideline, the comma strategy is usually regarded as less prone to stagnation, with the ratio  $|newpop|/popszie \simeq 6$  being a common choice [10]. Nevertheless, this option can be somewhat computationally expensive if the guiding function is complex and time consuming. Another common alternative is using a plus strategy with a low value of  $|newpop|$ , analogous to the so-called steady-state replacement strategy in GAs [274]. This option usually provides a faster convergence to high-quality solutions. However, care has to be taken with premature convergence to suboptimal regions of the search space, i.e., all configurations in the population being very similar to each other, hence hindering the exploration of other regions of  $\mathcal{S}$ .

The above consideration about premature convergence leads to the last component of the template shown in Algorithm 2, the restarting procedure. First of all, it must be decided whether the population has degraded or has not. To do so, it is possible to use some measure of information diversity in the population such as Shannon's entropy [60]. If this measure falls below a predefined threshold, the population is considered to be in a degenerate state. This threshold depends upon the representation (number of values per variable, constraints, etc.) and hence must be determined in an ad hoc fashion. A different possibility is using a probabilistic

approach to determine with a desired confidence that the population has converged. For example, in [119] a Bayesian approach is presented for this purpose.

Once the population is considered to be at a degenerate state, the restart procedure is invoked. Again, this can be implemented in a number of ways. A very typical strategy is to keep a fraction of the current population (this fraction can be as small as one solution, the current best) and substituting the remaining configurations with newly generated (from scratch) solutions, as shown in Algorithm 5.

---

**Algorithm 5** The RestartPopulation procedure.

---

```

1 Procedure RestartPopulation (pop);
2 begin
3   Initialize newpop using EmptyPopulation();
4   #preserved  $\leftarrow$  popsize · %preserve;
5   for j  $\leftarrow$  1 to #preserved do
6     i  $\leftarrow$  ExtractBestFromPopulation(pop);
7     InsertInPopulation individual i to newpop;
8   endfor
9   for j  $\leftarrow$  #preserved + 1 to popsize do
10    i  $\leftarrow$  GenerateRandomConfiguration();
11    i  $\leftarrow$  Local-Search-Engine (i);
12    InsertInPopulation individual i to newpop;
13  endfor
14  return newpop;
15 end
```

---

The procedure shown in Algorithm 5 is also known as the *random-immigrant* strategy [33]. Another possibility is to activate a *strong* or *heavy* mutation operator in order to drive the population away from its current location in the search space.

Both options have their advantages and disadvantages. For example, when using the random-immigrant strategy, one has to take some caution to prevent the preserved configurations to take over the population (this can be achieved by putting a low selective pressure, at least in the first iterations after a restart). As to the heavy mutation strategy, one has to achieve a trade-off between an excessively strong mutation that would destroy any information contained in the current population and a not so strong mutation that would cause the population to converge again in a few iterations.

### 6.2.6 Designing an Effective Memetic Algorithm

The general template of MAs depicted in Section 6.2.5 must be instantiated with precise components in order to be used for solving a specific problem. This instantiation has to be done carefully so as to obtain an effective optimization tool. We will address some design issues in this section.

A first obvious remark is that there exists no general approach for the design of effective MAs. This observation is based on different proofs depending on the precise definition of *effective* in the previous statement. Such proofs may involve classical complexity results and conjectures if “effective” is understood as “polynomial-time,” the NFL Theorem if we consider a more general set of performance measures, and even Computability Theory if we relax the definition to arbitrary decision problems. For these reasons, we can only define several *design heuristics* that will likely result in good-performing MAs, but without explicit guarantees for this.

This said, MAs are commonly implemented as evolutionary algorithms endowed with a local search component (recall Section 6.2.5), and as such can benefit from the theoretical corpus available for EAs. This is particularly applicable to some basic aspects such as the representation of solutions in terms of meaningful information units [59, 228]. Focusing now on more specific aspects of MAs, the first consideration that must be clearly taken into account is the interplay among the local search component and the remaining operators, mostly with respect to the characteristics of the search landscape. A good example of this issue can be found in the work of Merz and Freisleben on the TSP [85]. They consider the use of a highly intensive local search procedure—the Lin–Kernighan heuristic [157]—and note that the average distance between local optima is similar to the average distance between a local optimum and the global optimum. For this reason, they introduce a distance-preserving crossover (DPX) operator that generates offspring whose distance from the parents is the same as the distance between the parents themselves. Such an operator is likely to be less effective if a not-so-powerful local improvement method, e.g., 2-opt, was used, inducing a different distribution of local optima.

In addition to the particular choice (or choices) of local search operator, there remains the issue of determining an adequate parameterization for the procedure, namely, how much effort must be spent on each local search, how often the local search must be applied, and—were it not applied to every new solution generated—how to select the solutions that will undergo local improvement. Regarding the first two items, there exists theoretical evidence [143, 251] that an inadequate parameter setting can turn the algorithmic solution from easily solvable to non-polynomially solvable. Besides, there are obvious practical limitations in situations where the local search and/or the fitness function is computationally expensive. This fact admits different solutions. On the one hand, the use of surrogates (i.e., fast approximate models of the true function) to accelerate evolution is an increasingly popular option in such highly demanding problems [104, 155, 272, 273, 283]. On the other hand, partial lamarckism [42, 112, 212], where not every individual is subject to local search, is commonly used as well. The precise value for the local search application probability (or multiple values when more than one local search procedure is available) largely depends on the problem under consideration [123], and its determination is in many cases an art. For this reason, adaptive and self-adaptive mechanisms have been defined in order to let the algorithm learn what the most appropriate setting is (see Section 6.3.2).

As to the selection of individuals that will undergo local search, most common options are random selection and fitness-based selection, where only the best individuals are subject to local improvement. Nguyen et al. [197] also consider a “stratified” approach, in which the population is sorted and divided into  $n$  levels ( $n$  being the number of local search applications), and one individual per level is randomly selected. Their experimentation on some continuous functions indicates that this strategy and improve-the-best (i.e., applying local search to the best  $n$  individuals) provide better results than random selection. Such strategies can be readily deployed on a structured MA as defined by Moscato et al. [15, 21, 83, 169, 172], where good solutions flow upward within a tree-structured population, and layers are explicitly available. Other population management strategies are nevertheless possible, see [19, 218, 219, 249].

## 6.3 Algorithmic Extensions of Memetic Algorithms

The algorithmic template and design guidelines described in Section 6.2.6 can characterize most basic incarnations of MAs, namely population-based algorithms endowed with static local search for single-objective optimization. However, more sophisticated approaches can be conceived, and certainly required, in certain applications. This section is aimed at providing an overview of more advanced algorithmic extensions used in the MA realm.

### 6.3.1 Multiobjective Memetic Algorithms

Multiobjective problems are frequent in real-world applications. Rather than having a single objective to be optimized, the solver is faced with multiple, partially conflicting objectives. As a result, there is no a priori single optimal solution, but rather a collection of optimal solutions, providing different trade-offs among the objectives considered. In this scenario, the notion of Pareto-dominance is essential: given two solutions  $s, s' \in \text{sol}_P(x)$ ,  $s$  is said to dominate  $s'$  if it is better than  $s'$  in at least one of the objectives, and it is no worse in the remaining ones. This clearly induces a partial order  $\prec_P$ , since given two solutions it may be the case that none of them dominates the other. This collection of optimal solutions is termed the optimal Pareto front or the optimal non-dominated front.

Population-based search techniques, in particular evolutionary algorithms (EAs), are naturally fit to deal with multiobjective problems, due to the availability of a population of solutions which can approach the optimal Pareto front from different directions. There is extensive literature on the deployment of EAs in multiobjective settings, and the reader is referred to [35, 36, 63, 287], among others, for more information on this topic. MAs can obviously benefit from this corpus of knowledge. However, MAs typically incorporate a local search mechanism, and it has

to be adapted to the multiobjective setting as well. This can be done in different ways [132], which can be roughly classified into two major classes: scalarizing approaches and Pareto-based approaches. The scalarizing approaches are based on the use of some aggregation mechanism to combine the multiple objectives into a single scalar value. This is usually done using a linear combination of the objective values, with weights that are either fixed (at random or otherwise) for the whole execution of the local search procedure [266] or adapted as the local search progresses [106]. As to Pareto-based approaches, they consider the notion of Pareto-dominance for deciding transitions among neighboring solutions, typically coupled with the use of some measure of crowding to spread the search, e.g., [133].

A full-fledged multiobjective MA (MOMA) is obtained by appropriately combining population-based and local search-based components for multiobjective optimization. Again, the strategy used in the local search mechanism can be used to classify most MOMAs. Thus, two proposals due to Ishibuchi and Murata [121, 122] and to Jaszkiewicz [124, 125] are based on the use of random scalarization each time a local search is to be used. Alternatively, a single-objective local search could be used to optimize individual objectives [120]. Ad hoc mating strategies based on the particular weights chosen at each local search invocation (whereby the solutions to be recombined are picked according to these weights) are used as well. A related approach—including the online adjustment of scalarizing weights—is followed by Guo et al. [105–107]. On the other hand, a MA based on PAES (Pareto archived evolution strategy) was defined by Knowles and Corne [134, 135]. More recently, a MOMA based on particle swarm optimization (PSO) has been defined by Liu et al. [152, 162]. In this algorithm, an archive of non-dominated solutions is maintained and randomly sampled to obtain reference points for particles. A different approach is used by Schuetze et al. [237] for numerical optimization problems. The continuous nature of solution variables allows using their values for computing search directions. This fact is exploited in their local search procedure (HCS for Hill Climber with Sidestep) for directing the search toward specific regions (e.g., along the Pareto front) when required.

### ***6.3.2 Adaptive Memetic Algorithms***

When some design guidelines were given in Section 6.2.6, the fact that these were heuristics that ultimately relied on the problem knowledge available was stressed. This is not a particular feature of MAs, but affects the field of metaheuristics as a whole. Indeed, one of the keystones in practical metaheuristic problem solving is the necessity of customizing the solver for the problem at hand [51]. Therefore, it is not surprising that attempts to transfer a part of this tuning effort to the metaheuristic technique itself have been common. Such attempts can take place at different levels or can affect different components of the algorithm. The first—and more intuitive one—is the parametric level involving the numerical values of parameters, such as the operator application rates. Examples of this can be found in early EAs, see for

example [61]. A good up-to-date overview of these approaches (actually broader in scope, covering more advanced topics than parameter adaptation) can be found in [247]. Focusing specifically on MAs, this kind of adaptation has been applied in [11, 164, 175, 176].

A slightly more general approach—termed “meta-lamarckian learning” [204] by Ong and Keane—takes place at the algorithmic level. They consider a setting in which the MA has a collection of local search operators available, and how the selection of the particular operator(s) to be applied to a specific solution can be done on the basis of past performance of the operator, or on the basis of the similarity of the solution to previous successful cases of operator application. Some analogies can also be drawn here with hyperheuristics [54], a high-level heuristic that controls the application of a set of low-level heuristics to solutions, using strategies ranging from pure random to performance-based rules. See [28] for a recent comprehensive overview of hyperheuristics.

In general terms, the approaches mentioned before are based on static, hard-wired mechanisms that the MA uses to react to the environment. Hence, they can be regarded as adaptive, but not as self-adaptive [205]. In the latter case, the actual definition of the search mechanisms can evolve during the search. This is a goal that has been pursued for long in MAs. Back in the early days of the field, it was already envisioned that future generations of MAs would work in at least two levels and two timescales [179]. During the short timescale, a set of agents would be searching in the search space associated with the problem. The long timescale would *adapt the algorithms* associated with the agents. Here we encompass individual search strategies, recombination operators, etc. A simple example of this kind of self-adaptation can be found in the so-called multi-memetic algorithms, in which each solution carries a gene that indicates which local search has to be applied on it. This can be a simple pointer to an existing local search operator or even the parametrization of a general local search template, with items such as the neighborhood to use and acceptance criterion. [141]. Going beyond, a grammar can be defined to specify a more complex local search operator [140, 142]. At an even higher level, this evolution of local search operators can be made fully symbiotic, rather than merely endosymbiotic. For this purpose, two co-evolving populations can be considered: a population of solutions and a population of local search operators. These two populations co-operate by means of an appropriate pairing mechanism that associates solutions with operators. The latter receive fitness in response to their ability to improve solutions, thus providing a fully self-adaptive strategy for exploring the search landscape [244–246].

### 6.3.3 Complete Memetic Algorithms

The combination of exact techniques with metaheuristics is an increasingly popular approach. Focusing on local search techniques, Dumitrescu and Stützle [73] have provided a classification of methods in which exact algorithms are used to

strengthen local search, i.e., to explore large neighborhoods, to solve exactly some subproblems, to provide bounds and problem relaxations to guide the search. Some of these combinations can be also found in the literature on population-based methods. For example, exact techniques—such as branch-and-bound (BnB) [53] or dynamic programming [90]—have been used to perform recombination (recall Section 6.2.4), and approaches in which exact techniques solved some subproblems provided by EAs date back to 1995 [45]. See also [76] for a large list of references regarding local search/exact hybrids.

Puchinger and Raidl [220] have provided a classification of this kind of hybrid techniques in which algorithmic combinations are either collaborative (sequential or intertwined execution of the combined algorithms) or integrative (one technique works inside the other one, as a subordinate). Some of the exact/metaheuristic hybrid approaches defined before are clearly integrative—i.e., using an exact technique to explore neighborhoods. Further examples are the use of BnB in the decoding process [221] of a genetic algorithm (i.e., exact method within a metaheuristic technique) or the use of evolutionary techniques for the strategic guidance of BnB [139] (metaheuristic approach within an exact method).

As to collaborative combinations, a sequential approach in which the execution of a MA is followed by a branch-and-cut method can be found in [131]. Intertwined approaches are also popular. For example, Denzinger and Offerman [66] combine genetic algorithms and BnB within a parallel multi-agent system. These two algorithms also cooperate in [45, 88], the exact technique providing partial promising solutions, and the metaheuristic returning improved bound. A related approach involving beam search and full-fledged MAs can be found in [89, 92, 93].

It must be noted that most hybrid algorithms defined so far that involve exact techniques and metaheuristics are not complete, in the sense that they do not guarantee an optimal solution (an exception is the proposal of French et al. [86], combining an integer-programming BnB approach with GAs for MAX-SAT). Thus, the term “complete MA” may be not fully appropriate. Nevertheless, many of these hybrids can be readily adapted for completeness purposes, although obviously time and/or space requirements will grow faster-than-polynomial in general.

## 6.4 Applications of Memetic Algorithms

This section will provide an overview of the numerous applications of MAs. This overview is far from exhaustive since new applications are being developed continuously. However, it is intended to illustrate the practical impact of these optimization techniques. We have focused on recent applications, namely in the last 5 years (that is, from 2004 onward). Readers interested in earlier applications (which are also manifold) can refer to [109, 180–182]. We have organized references in five major areas: machine learning and knowledge discovery (Table 6.1); traditional combinatorial optimization (Table 6.2); planning, scheduling, and timetabling (Table 6.3); bioinformatics (Table 6.4); and electronics, engineering, and telecommunications

**Table 6.1** Applications in machine learning and knowledge discovery.

Data mining and knowledge discovery	Image analysis	[37, 67, 68, 77, 211]
	Fuzzy clustering	[70]
	Feature selection	[243, 286]
	Pattern recognition	[94]
Machine learning	Decision trees	[144]
	Inductive learning	[69]
	Neural networks	[64, 65, 103, 110, 159, 168, 195, 262]

**Table 6.2** Applications in combinatorial optimization.

Binary and set problems	Binary quadratic programming	[173]
	Knapsack problem	[87, 88, 105, 107, 222]
	Low autocorrelation sequences	[91]
	MAX-SAT	[18, 223]
	Set covering	[125]
Graph-based problems	Crossdock optimization	[2, 154]
	Graph coloring	[38]
	Graph matching	[12]
	Hamiltonian cycle	[32]
	Maximum cut	[270]
	Quadratic assignment	[72, 255]
	Routing problems	[19, 20, 56, 57, 74] [80, 145–147] [218, 259, 263]
	Spanning tree	[79, 231]
	Steiner tree	[131]
	TSP	[21, 161, 163, 196, 271]
Constrained optimization	Golomb ruler	[46, 48]
	Social golfer	[47]
	Maximum density still life	[89, 90]

**Table 6.3** Applications in planning, scheduling, timetabling, and manufacturing. (check also [49])

Manufacturing	Assembly line	[226, 257, 265]
	Flexible manufacturing	[5, 31, 187, 258]
	Lot sizing	[16]
	Multi-tool milling	[13]
	Supply chain network	[280]
Planning	Temporal planning	[235]
Scheduling	Flowshop scheduling	[82, 84, 152, 158, 160, 184, 209, 240, 241]
	Job-shop	[27, 96–98, 224, 267, 268, 278]
	Parallel machine scheduling	[184, 277]
	Project scheduling	[29]
	Single machine scheduling	[166, 184]
Timetabling	Driver scheduling	[153]
	Examination timetabling	[216]
	Rostering	[3, 22, 206]
	Sport league	[236]
	Train timetabling	[239]
	University course	[151, 215, 233]

**Table 6.4** Applications in bioinformatics.

Phylogeny	Phylogenetic inference	[43, 93, 275]
	Consensus tree	[217]
Microarrays	Biclustering	[208]
	Feature selection	[55, 284, 285]
	Gene ordering	[169, 183]
Sequence analysis	Shortest common supersequence	[42, 92]
	DNA sequencing	[71]
Protein science	Sequence assignment	[269]
	Structure comparison	[140]
	Structure prediction	[14, 40, 203, 234, 281]
Systems biology	Gene regulatory networks	[200, 250]
	Cell models	[232]
Biomedicine	Drug therapy design	[194, 264]

**Table 6.5** Applications in electronics, telecommunications, and engineering.

Electronics	Analog circuit design	[58, 170]
	Circuit partitioning	[34]
	Electromagnetism	[23, 104, 210]
	Filter design	[254]
	VLSI design	[7, 171, 256]
Engineering	Chemical kinetics	[136, 137]
	Crystallography	[212]
	Drive design	[24, 25]
	Power systems	[26]
	Structural optimization	[129]
	System modeling	[1, 260]
Computer Science	Code optimization	[207]
	Information forensics	[242]
	Information theory	[41]
	Software engineering	[6]
Telecommunications	Antenna design	[114–117]
	Mobile networks	[128, 225]
	P2P networks	[174, 191, 192]
	Wavelength assignment	[78]
	Wireless networks	[113, 118, 130, 138]

(Table 6.5). As mentioned before, we have tried to be illustrative rather than exhaustive, pointing out some selected references from these well-known application areas.

Although these fields encompass the vast majority of applications of MAs, it must be noted that success stories are not restricted to these major fields. To cite an example, there are several applications of MAs in economics, e.g., in portfolio optimization [165], risk analysis [167], and labor-market delineation [81]. For further information about MA applications we suggest querying bibliographical databases or web browsers for the keywords “memetic algorithms” and “hybrid genetic algorithms.”

## 6.5 Challenges and Future Directions

The future seems promising for MAs. This is the combination of several factors. First, MAs (less frequently disguised under different names) are showing a remarkable record of efficient implementations, providing very good results in practical problems. Second, there are reasons to believe that some new attempts to do theoretical analysis can be conducted. This includes the worst-case and average-case computational complexity of recombination procedures. Third, the ubiquitous nature of distributed systems, like networks of workstations for example, plus the inherent asynchronous parallelism of MAs and the existence of web-conscious languages like Java, all together are an excellent combination to develop highly portable and extendable object-oriented frameworks allowing algorithmic reuse. These frameworks might allow the users to solve subproblems using commercial codes or well-tested software from other users who might be specialists in another area. Fourth, an important and pioneering group of MAs, that of *Scatter Search* [95, 148], is challenging the role of randomization in recombination. We expect that, as a healthy reaction, we will soon see new types of powerful MAs that blend in a more appropriate way both exhaustive (either truncated or not) and systematic search methods.

### 6.5.1 Learning from Experience

In 1998, Applegate, Bixby, Cook, and Chvatal established new breakthrough results for the MIN TSP. They solved to optimality an instance of the TSP of 13,509 cities corresponding to all US cities with populations of more than 500 people. The approach, according to Bixby, "...involves ideas from polyhedral combinatorics and combinatorial optimization, integer and linear programming, computer science data structures and algorithms, parallel computing, software engineering, numerical analysis, graph theory, and more." The solution of this instance demanded the use of three Digital AlphaServer 4100s (with a total of 12 processors) and a cluster of 32 Pentium-II PCs. The complete calculation took approximately 3 months of computer time. The code has certainly more than 1,000 pages and is based on state-of-the-art techniques from a wide variety of scientific fields.

The philosophy is the same in the case of MAs, that of a synergy of different approaches. Actually, their approach can possibly be classified as the most complex MA ever built for a given combinatorial optimization problem. One of the current challenges is to develop simpler algorithms that achieve these impressive results. The approach of running a local search algorithm (Chained Lin–Kernighan) to produce a collection of tours, followed by the dynamically optimal recombination method called *tour merging*, produced a non-optimal tour only 0.0002% above the proved optimal tour for the 13,509 cities instance. We take this as a clear proof of the benefits of the MA approach and that more work is needed in developing good strategies for *complete memetic algorithms*, i.e., those that systematically and synergistically use randomized and deterministic methods and can prove optimality.

An open line for the design of this kind of algorithms may be the exploitation of FPT (fixed-parameter tractability) results, see Section 6.5.2. Related to this, it must be noted that we still lack a formal framework for recombination, similar for instance to the one for Local Search [126, 279]. In this sense, an interesting new direction for theoretical research arose after the introduction of two computational complexity classes: the PMA class (for Polynomial Merger Algorithms problems) and its unconstrained analogue, the uPMA class (see [180]). These classes are defined analogously to the class of polynomial local search (PLS). Conducting research to identify problems, and their associated recombination procedures, such that membership, in either PMA or uPMA, can be proved is a definitely important task. It is also hoped that after some initial attempts on challenging problems, completeness and reductions for these classes can be properly defined [50].

### 6.5.2 Exploiting FPT results

An interesting new avenue of research can be established by appropriately linking results from the theory of fixed-parameter tractability (FPT) and the development of recombination algorithms. A parameterized problem can be generally viewed as a problem with two input components, i.e., a pair  $\langle x, k \rangle$ . The former is generally an instance (i.e.,  $x \in I_P$ ) of some other decision problem  $P$  and the latter is some numerical aspect of the former (generally a positive integer assumed  $k \ll |x|$ , where  $|x|$  is the size of instance  $x$ ) that constitutes a *parameter*, for example, the maximum node degree in a certain graph-based problem, the maximum number of elements in the solution of a subset-selection problem. If there exists an algorithm solving the problem in time  $O(f(k)|x|^\alpha)$ , where  $f(k)$  is an *arbitrary* function depending on  $k$  only, and  $\alpha$  a constant independent of  $k$  or  $n$ , the parameterized problem is said to be *fixed-parameter tractable* and the decision problem belongs to the computational complexity class FPT. Note that by following this parameterized approach, the complexity analysis becomes multidimensional, in contrast to the classical one-dimensional approach, in which only the instance size is considered (thus failing to distinguish structural properties that may make a particular problem instance hard or easy).

To illustrate this topic, consider one of the most emblematic FPT problems, namely VERTEX COVER: given a graph  $G(V, E)$ , find a subset  $S \subseteq V$  of  $k$  vertices, such that for every edge  $(u, v) \in E$ , at least  $u$  or  $v$  is a member of  $S$ . Here, the number  $k$  of vertices in  $S$  is taken as a parameter and factored out from the problem input. In general, efficient FPT algorithms are based on the techniques of *reduction to a problem kernel* and *bounded search trees*. To understand the techniques, the reader may check a method by Chen et al. [30]. This method can solve the parameterized version of vertex cover in time  $O(1.271^k k^2 + kn)$ . Furthermore, using this method together with the speed-up method proposed by Neidermeier and Rossmanith [199], the problem can be solved in  $O(1.271^k + n)$ , i.e., linear in  $n$  for fixed  $k$ . The relevance of this result is more evident by noting that VERTEX COVER is an NP-hard

problem. Thus, FPT results provide an efficient way for provably solving NP-hard problems for fixed-parameter values.

The combination of FPT results and recombination operators is an avenue that goes both ways. In one direction, efficient (i.e., polynomial-time), fixed-parameter algorithms can be used as “out of the box” tools to create efficient recombination procedures, i.e., recall some of the procedures mentioned in Section 6.3.3. Conversely, since MAs are typically designed to deal with large instances and scale pretty well with problem size, using both techniques together can produce *complete MAs*, thus extending the benefits of fixed-parameter tractability. From a software engineering perspective, the combination is perfect both from code and from algorithmic reuse.

### 6.5.3 Belief Search in Memetic Algorithms

As a logical consequence of the possible directions that MAs can take, it is reasonable to affirm that more complex schemes evolving solutions, agents, as well as representations, will soon be implemented. Some theoretical computer science researchers dismiss heuristics and metaheuristics since they are not scholarly structured as a formal paradigm. However, their achievements are well recognized. From [150]:

Explaining and predicting the impressive empirical success of some of these algorithms is one of the most challenging frontiers of the theory of computation today.

This comment is even more relevant for MAs since they generally present even better results than single-agent methods. Though metaheuristics are extremely powerful in practice, we agree that one problem with the current trend in applied research is that it allows the introduction of increasingly more complex heuristics, unfortunately most of the time parameterized by ad hoc values. Moreover, some metaheuristics, like some *ant-systems* implementations, can basically be viewed as particular types of MAs. This is the case if you allow the “ants” to use *branch-and-bound* or *local search* methods. In addition, these methods for distributed recombination of information (or beliefs) have some points in common with *blackboard systems* [75], as it has been recognized in the past, yet it is hardly being mentioned in the current metaheuristics literature [180].

To illustrate how Belief Search can work in an MA setting, consider for example  $\mathbf{PL}_n^{\otimes}$ , a multi-agent epistemic logic introduced by Boldrin and Saffiotti [17]. According to this formalism, the opinions shared by a set of  $n$  agents can be *recombined* in a distributed belief. Using it, we can deduce the distributed belief about properties of solutions, and this can be stronger than any individual belief about it (see [50] for detailed examples with numerical values).

One interesting application of these new MAs is due to Lamma et al. [149] for diagnosing digital circuits. In their approach, they differentiate between genes and “memes.” The latter group codes for the agent beliefs and assumptions. Using a

logic-based technique, they modify the memes according to how the present beliefs are contradicted by integrity constraints that express observations and laws. Each agent keeps a population of chromosomes and finds a solution to the belief revision problem by means of a genetic algorithm. A *Lamarckian* operator is used to modify a chromosome using belief revision directed mutations, oriented by tracing logical derivations. As a consequence, a chromosome will satisfy a larger number of constraints. The evolution provided by the Darwinian operators allows agents to improve the chromosomes by gaining on the experience of other agents. Central to this approach is the Lamarckian operator appropriately called *Learn*. It takes a chromosome and produces a revised chromosome as output. To achieve that, it eliminates some derivation paths that lead to contradictions.

Surprisingly enough (and here we remark the first possibility of using the theory of *fixed-parameter tractability*), the learning is achieved by finding a *hitting set* which is not necessarily minimal. The authors make this point clear by saying that: “a hitting set generated from these support sets is not necessarily a contradiction removal set and therefore is not a solution to the belief revision problem.” The authors might not be aware of the  $O(2.311^k + n)$  exact algorithm for MIN 3-HITTING SET [198]. They might be able to use it, but that is anecdotal at the moment. What is important is that algorithms like this one might be used *out-of-the-box* if a proper, worldwide based, algorithmic framework was created.

On the other hand, we noted how results of logic programming and belief revision might help improve the current status of metaheuristics. The current situation where everybody comes with new names for the same basic techniques, and where most contributions are just the addition of new parameters to guide the search, is a futile research direction. It is possible that belief-search-guided MAs will prove to be a valid tool to help systematize the construction of these guided metaheuristics. In particular, the discussion is based on which multi-agent logic performs better, rather than which parameters work better for specific problems or instances. To this end, we hope to convince researchers in logic programming to address these issues and to face the difficult task of guiding MAs for large-scale combinatorial optimization.

## 6.6 Conclusions

We believe that the future looks good for MAs. This belief is based on the following. First of all, MAs are showing a great record of efficient implementations, providing very good results in practical problems, as the reader may have noted in Section 6.4. We also have reasons to believe that we are close to some major leaps forward in our theoretical understanding of these techniques, including, for example, the worst-case and average-case computational complexity of recombination procedures. On the other hand, the ubiquitous nature of distributed systems is likely to boost the deployment of MAs on large-scale, computationally demanding optimization problems.

We also see as a healthy sign the systematic development of other particular optimization strategies. If any of the simpler metaheuristics (SA, TS, VNS, GRASP, etc.) performs the same as a more complex method (GAs, MAs, Ant Colonies, etc.), an “elegance design” principle should prevail and we must either resort to the simpler method, or to the one that has less free parameters, or to the one that is easier to implement. Such a fact should defy us to adapt the complex methodology to beat a simpler heuristic or to check if that is possible at all. An unhealthy sign of current research, however, is the attempts to encapsulate metaheuristics on stretched confinements. Fortunately, such attempts are becoming increasingly less frequent. Indeed, combinations of MAs with other metaheuristics such as differential evolution [193, 201, 261], particle swarm optimization [152, 158, 159, 161, 162, 209, 214, 248, 282], or ant-colony optimization [156] are not unusual nowadays. As stated before, the future looks promising for MAs.

**Acknowledgments** This chapter is an updated second edition of [180], refurbished with new references and the inclusion of sections on timely topics which were not fully addressed in the first edition. Carlos Cotta acknowledges the support of Spanish Ministry of Science and Innovation, under project TIN2008-05941.

## References

1. Ahmad, R., Jamaluddin, H., Hussain, M.A.: Application of memetic algorithm in modelling discrete-time multivariable dynamics systems. *Mech. Syst. Signal Process.* **22**(7), 1595–1609 (2008)
2. Aickelin, U., Adewunmi, A.: Simulation optimization of the crossdock door assignment problem. In: UK Operational Research Society Simulation Workshop 2006 (SW 2006), Leamington Spa, UK, March 11 2006
3. Aickelin, U., White, P.: Building better nurse scheduling algorithms. *Ann. Oper. Res.* **128**, 159–177 (2004)
4. Aldous, D., Vazirani, U.: “Go with the winners” algorithms. In: Proceedings of the 35th Annual Symposium on Foundations of Computer Science, pp. 492–501. IEEE Press, Los Alamitos, CA, (1994)
5. Amaya, J.E., Cotta, C., Fernández, A.J.: A memetic algorithm for the tool switching problem. In: Blesa, M.J., et al. (eds.) Hybrid metaheuristics 2008, vol. 5296, Lecture notes in computer science, pp. 190–202. Springer, Heidelberg (2008)
6. Arcuri, A., Yao, X.: A memetic algorithm for test data generation of object-oriented software. In: Srinivasan, D., Wang, L. (eds.) 2007 IEEE Congress on Evolutionary Computation, pp. 2048–2055, Singapore, 25–28 September 2007. IEEE Computational Intelligence Society, IEEE Press (2007)
7. Areibi, S., Yang, Z.: Effective Memetic Algorithms for VLSI design = genetic algorithms plus local search plus multi-level clustering. *Evol. Comput.* **12**(3), 327–353 (2004)
8. Axelrod, R., Hamilton, W.D.: The evolution of cooperation. *Science* **211**(4489), 1390–1396 (1981)
9. Bäck, T.: Evolutionary Algorithms in Theory and Practice. Oxford University Press, New York (1996)
10. Bäck, T., Hoffmeister, F.: Adaptive search by evolutionary algorithms. In: Ebeling, W., Peschel, M., Weidlich, W. (eds.) Models of Self-organization in Complex Systems, number 64 in Mathematical Research, pp. 17–21. Akademie, Berlin (1991)

11. Bambha, N.K., Bhattacharyya, S.S., Teich, J., Zitzler, E.: Systematic integration of parameterized local search into evolutionary algorithms. *IEEE Trans. Evol. Comput.* **8**(2), 137–155 (2004)
12. Bärecke, T., Detyniecki, M.: Memetic algorithms for inexact graph matching. In: Srinivasan, D., Wang, L. (eds.) 2007 IEEE Congress on Evolutionary Computation, pp. 4238–4245, Singapore, 25–28 September 2007. IEEE Computational Intelligence Society, IEEE Press (2007)
13. Baskar, N., Asokan, P., Saravanan, R., Prabhaharan, G.: Selection of optimal machining parameters for multi-tool milling operations using a memetic algorithm. *J. Mater. Process. Tech.* **174**(1–3), 239–249 (2006)
14. Bazzoli, A., Tettamanzi, A.G.B.: A memetic algorithm for protein structure prediction in a 3D-Lattice HP model. In: Raidl, G.R., et al. (eds.) Applications of Evolutionary Computing, vol. 3005, *Lecture Notes in Computer Science*, pp. 1–10, Berlin, 2004. Springer.
15. Berretta, R., Cotta, C., Moscato, P.: Enhancing the performance of memetic algorithms by using a matching-based recombination algorithm: Results on the number partitioning problem. In: Resende, M., Pinho de Sousa, J., (eds.) Metaheuristics: Computer-Decision Making, pp. 65–90. Kluwer, Boston MA (2003)
16. Berretta, R., Rodrigues, L.F.: A memetic algorithm for a multistage capacitated lot-sizing problem. *Int. J. Prod. Econ.* **87**(1), 67–81 (2004)
17. Boldrin, L., Saffiotti, A.: A modal logic for merging partial belief of multiple reasoners. *J. Logic Comput.* **9**(1), 81–103 (1999)
18. Borschbach, M., Exeler, A.: A tabu history driven crossover operator design for memetic algorithm applied to max-2SAT-problems. In: Keijzer, M. et al. (eds.) GECCO '08: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, pp. 605–606, Atlanta, GA, USA, 12–16 July 2008. ACM Press.
19. Boudia, M., Prins, C., Reghioui, M.: An effective memetic algorithm with population management for the split delivery vehicle routing problem. In: Bartz-Beielstein, T., et al. (eds.) Hybrid Metaheuristics 2007, vol. 4771, Lecture Notes in Computer Science, pp. 16–30. Springer, Berlin, Heidelberg (2007)
20. Bouly, H., Dang, D.-C., Moukrim, A.: A memetic algorithm for the team orienteering problem. In: Giacobini, M., et al. (eds.) Applications of Evolutionary Computing vol. 4974, Lecture Notes in Computer Science, pp. 649–658. Springer, Berlin, Heidelberg (2008)
21. Buriol, L., França, P.M., Moscato, P.: A new memetic algorithm for the asymmetric traveling salesman problem. *J. Heuristics* **10**(5), 483–506 (2004)
22. Burke, E.K., De Causmaecker, P., van den Berghe, G.: Novel metaheuristic approaches to nurse rostering problems in belgian hospitals. In: Leung, J. (ed.) Handbook of Scheduling: Algorithms, Models, and Performance Analysis, chapter 44, pp. 44.1–44.18. Chapman Hall/CRC Press, Boca Raton, FL (2004)
23. Caorsi, S., Massa, A., Pastorino, M., Randazzo, A.: Detection of PEC elliptic cylinders by a memetic algorithm using real data. *Microwave Optical Technol. Lett.* **43**(4), 271–273 (2004)
24. Caponio, A., Leonardo Cascella, G., Neri, F., Salvatore, N., Sumner, M.: A fast adaptive memetic algorithm for online and offline control design of pmsm drives. *IEEE Trans. Syst. Man Cybernet. Part B* **37**(1), 28–41 (2007)
25. Caponio, A., Neri, F., Cascella, G.L., Salvatore, N.: Application of memetic differential evolution frameworks to PMSM drive design. In: Wang, J. (ed.) 2008 IEEE World Congress on Computational Intelligence, pp. 2113–2120, Hong Kong, 1–6 June 2008. IEEE Computational Intelligence Society, IEEE Press (2008)
26. Carrano, E.G., Souza, B.B., Neto, O.M.: An immune inspired memetic algorithm for power distribution system design under load evolution uncertainties. In: Wang, J. (ed.) 2008 IEEE World Congress on Computational Intelligence, pp. 3251–3257, Hong Kong, 1–6 June 2008. IEEE Computational Intelligence Society, IEEE Press (2008)
27. Caumond, A., Lacomm, P., Tchernev, N.: A memetic algorithm for the job-shop with time-lags. *Computers & Or*, **35**(7), 2331–2356 (2008)

28. Chakhlevitch, K., Cowling, P.: Hyperheuristics: Recent developments. In: Cotta, C., Sevaux, M., Sørensen, K. (eds.) *Adaptive and Multilevel Metaheuristics*, vol. 136, *Studies in Computational Intelligence*, pp. 3–29. Springer, Berlin (2008)
29. Chen, A.H.L., Chyu, C.-C.: A memetic algorithm for maximizing net present value in resource-constrained project scheduling problem. In: Wang, J. (ed.) *2008 IEEE World Congress on Computational Intelligence*, pp. 2401–2408, Hong Kong, 1–6 June 2008. IEEE Computational Intelligence Society, IEEE Press (2008)
30. Chen, J., Kanj, I.A., Jia, W.: Vertex cover: further observations and further improvements. In: *Proceeding of 25th International Workshop Graph-Theoretic Concepts in Computer Science*, vol. 1665, *Lecture Notes in Computer Science*, pp. 313–324. Springer, Berlin, Heidelberg (1999)
31. Chen, J.-H., Chen, J.-H.: Multi-objective memetic approach for flexible process sequencing problems. In: Ebner, M., et al. (eds.) *GECCO-2008 Late-Breaking Papers*, pp. 2123–2128, Atlanta, GA, USA, 12–16 July 2008. ACM Press (2008)
32. Chen, X.S., Lim, M.H., Wunsch II, D.C.: A memetic algorithm configured via a problem solving environment for the hamiltonian cycle problems. In: Srinivasan, D., Wang, L. (eds.) *2007 IEEE Congress on Evolutionary Computation*, pp. 2766–2773, Singapore, 25–28 September 2007. IEEE Computational Intelligence Society, IEEE Press (2007)
33. Cobb, H.G., Grefenstette, J.J.: Genetic algorithms for tracking changing environments. In: Forrest, S. (ed.) *Proceedings of the Fifth International Conference on Genetic Algorithms*, pp. 529–530, San Mateo, CA, 1993. Morgan Kaufmann (1993)
34. Coe, S., Areibi, S., Moussa, M.: A hardware memetic accelerator for VLSI circuit partitioning. *Comput. Eng.* **33**(4), 233–248 (2007)
35. Coello Coello, C.A., Lamont, G.B.: *Applications of Multi-Objective Evolutionary Algorithms*. World Scientific, New York (2004)
36. Coello Coello, C.A., Van Veldhuizen, D.A., Lamont, G.B.: *Evolutionary Algorithms for Solving Multi-Objective Problems*, volume 5 of *Genetic Algorithms and Evolutionary Computation*. Kluwer, Boston, MA (2002)
37. Córdón, O., Damas, S., Santamaría, J.: A scatter search algorithm for the 3D image registration problem. In: Yao, X., et al. (eds.) *Parallel Problem Solving From Nature VIII*, vol. 3242, *Lecture Notes in Computer Science*, pp. 471–480, Berlin, 2004. Springer, Berlin, Heidelberg (2004)
38. Cosmin, D., Hao, J.-K., Kuntz, P.: Diversity control and multi-parent recombination for evolutionary graph coloring. In: Cotta, C., Cowling, P. (eds.) *Evolutionary Computation in Combinatorial Optimization*, vol. 5482, *Lecture Notes in Computer Science*, pp. 121–132, Tübingen, 2009. Springer, Berlin, Heidelberg (2009)
39. Cotta, C.: A study of hybridisation techniques and their application to the design of evolutionary algorithms. *AI Commun.* **11**(3–4), 223–224 (1998)
40. Cotta, C.: Hybrid evolutionary algorithms for protein structure prediction in the HPNX model. In: Reusch, B. (ed.) *Computational intelligence, Theory and Applications, Advances in Soft Computing*, pp. 525–534, Springer, Heidelberg (2004)
41. Cotta, C.: Scatter search and memetic approaches to the error correcting code problem. In: Gottlieb, J., Raidl, G.R. (eds.) *Evolutionary Computation in Combinatorial Optimization*, vol. 3004, *Lecture Notes in Computer Science*, pp. 51–60. Springer, Berlin (2004)
42. Cotta, C.: Memetic algorithms with partial lamarckism for the shortest common supersequence problem. In: Mira, J., Álvarez, J.R. (eds.) *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*, vol. 3562, *Lecture Notes in Computer Science*, pp. 84–91. Springer, Berlin (2005)
43. Cotta, C.: Scatter search with path relinking for phylogenetic inference. *Eur. J. Oper. Res.* **169**(2), 520–532, 2005
44. Cotta, C., Alba, E., Troya, J.M.: Stochastic reverse hillclimbing and iterated local search. In: *Proceedings of the 1999 Congress on Evolutionary Computation*, pp. 1558–1565, Washington DC, 1999. IEEE (1999)

45. Cotta, C., Aldana, J.F., Nebro, A.J., Troya, J.M.: Hybridizing genetic algorithms with branch and bound techniques for the resolution of the TSP. In: Pearson, D.W., Steele, N.C., Albrecht, R.F. (eds.) *Artificial Neural Nets and Genetic Algorithms 2*, pp. 277–280. Springer, New York (1995)
46. Cotta, C., Dotú, I., Fernández, A.J., Van Hentenryck, P.: A memetic approach to Golomb rulers. In: Runarsson, T.P., et al. (eds.) *Parallel Problem Solving from Nature IX*, vol. 4193, Lecture Notes in Computer Science, pp. 252–261. Springer, Berlin (2006)
47. Cotta, C., Dotú, I., Fernández, A.J., Van Hentenryck, P.: Scheduling social golfers with memetic evolutionary programming. In: *Hybrid Metaheuristic 2006*, vol. 4030, Lecture Notes in Computer Science, pp. 150–161. Springer, Berlin, Heidelberg (2006)
48. Cotta, C., Fernández, A.: A hybrid GRASP – evolutionary algorithm approach to golomb ruler search. In: Yao, X., et al. (eds.) *Parallel Problem Solving From Nature VIII*, vol. 3242, Lecture Notes in Computer Science, pp. 481–490. Springer, Berlin (2004)
49. Cotta, C., Fernández, A.J.: Memetic algorithms in planning, scheduling, and timetabling. In K.P. Dahal, K.C. Tan, and P.I. Cowling, editors, *Evolutionary Scheduling*, volume 49 of *Studies in Computational Intelligence*, pages 1–30. Springer-Verlag, 2007.
50. Cotta, C., Moscato, P.: Evolutionary computation: Challenges and duties. In: Menon, A. (ed.) *Frontiers of Evolutionary Computation*, pp. 53–72. Kluwer, Boston, MA (2004)
51. Cotta, C., Sevaux, M., Sørensen, K.: Adaptive and Multilevel Metaheuristics, volume 136 of *Studies in Computational Intelligence*. Springer, Berlin (2008)
52. Cotta, C., Troya, J.M.: On the influence of the representation granularity in heuristic forma recombination. In: Carroll, J., Damiani, E., Haddad, H., Oppenheim, D. (eds.) *ACM Symposium on Applied Computing 2000*, pp. 433–439. ACM Press, Como, Italy (2000)
53. Cotta, C., Troya, J.M.: Embedding branch and bound within evolutionary algorithms. *Appl. Intell.* **18**(2), 137–153, 2003.
54. Cowling, P., Kendall, G., Soubeiga, E.: A hyperheuristic approach to schedule a sales submit. In: Burke, E., Erben, W. (eds.) *PATAT 2000*, vol. 2079, Lecture Notes in Computer Science, pp. 176–190. Springer, Berlin (2008)
55. Cox, M., Bowden, N., Moscato, P., Berretta, R., Scott, R.I., Lechner-Scott, J.S.: Memetic algorithms as a new method to interpret gene expression profiles in multiple sclerosis. *Mult. Scler.* **13**(Suppl. 2), S205 (2007)
56. Créput, J.-C., Koukam, A.: The memetic self-organizing map approach to the vehicle routing problem. *Soft Comput.* **12**(11), 1125–1141 (2008)
57. Cruz-Chavez, M.A., Díaz-Parra, O., Juárez-Romero, D., Martínez-Rangel, M.G.: Memetic algorithm based on a constraint satisfaction technique for VRPTW. In: Rutkowski, L., et al. (eds.) *9th Artificial Intelligence and Soft Computing Conference*, vol. 5097, Lecture Notes in Computer Science, pp. 376–387. Springer, Berlin, Heidelberg (2008)
58. Dantas, M.J., da, L., Brito, C., de Carvalho, P.H.: Multi-objective Memetic Algorithm applied to the automated synthesis of analog circuits. In: Simão Sichman, J., Coelho, H., Oliveira Rezende, S. (eds.) *Advances in Artificial Intelligence*, vol. 4140, Lecture Notes in computer Science, pp. 258–267. Springer, Berlin, Heidelberg (2006)
59. Davidor, Y.: Epistasis Variance: Suitability of a Representation to Genetic Algorithms. *Complex Syst.* **4**(4), 369–383 (1990)
60. Davidor, Y., Ben-Kiki, O.: The interplay among the genetic algorithm operators: Information theory tools used in a holistic way. In: Männer, R., Manderick, B. (eds.) *Parallel Problem Solving From Nature II*, pp. 75–84. Elsevier, Amsterdam (1992)
61. Davis, L.: *Handbook of Genetic Algorithms*. Van Nostrand Reinhold Computer Library, New York (1991)
62. Dawkins, R.: *The Selfish Gene*. Clarendon, Oxford (1976)
63. Deb, K.: *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley, Chichester, UK (2001)
64. Delgado, M., Cuellar, M.P., Pegalajar, M.C.: Multiobjective hybrid optimization and training of recurrent neural networks. *IEEE Trans. Syst. Man Cybernet. Part B* **38**(2), 381–403 (2008)

65. Delgado, M., Pegalajar, M.C., Cuellar, M.P.: Memetic evolutionary training for recurrent neural networks: an application to time-series prediction. *Expert. Syst.* **23**(2), 99–115 (2006)
66. Denzinger, J., Offermann, T.: On cooperation between evolutionary algorithms and other search paradigms. In: 6th International Conference on Evolutionary Computation, pp. 2317–2324. IEEE Press, Washington, DC (1999)
67. di Gesù, V., Lo Bosco, G., Millonzi, F., Valenti, C.: Discrete tomography reconstruction through a new memetic algorithm. In: Giacobini, M., et al. (eds.) *Applications of Evolutionary Computing*, vol. 4974, Lecture Notes in Computer Science, pp. 347–352. Springer, Berlin, Heidelberg (2008)
68. di Gesù, V., Lo Bosco, G., Millonzi, F., Valenti, C.: A memetic algorithm for binary image reconstruction. In: Brimkov, V.E., Barneva, R.P., Hauptman, H.A. (eds.) *Combinatorial Image Analysis*, pp. 384–395. Springer, Berlin, Heidelberg (2008)
69. Divina, F.: Hybrid genetic relational search for inductive learning. PhD thesis, Department of Computer Science, Vrije Universiteit, Amsterdam, the Netherlands (2004)
70. Do, A.-D., Cho, S.Y.: Memetic algorithm based fuzzy clustering. In: Srinivasan, D., Wang, L. (eds.) *2007 IEEE Congress on Evolutionary Computation*, pp. 2398–2404, Singapore, 25–28 September 2007. IEEE Computational Intelligence Society, IEEE Press (2007)
71. Dorronsoro, B., Alba, E., Luque, G., Bouvry, P.: A self-adaptive cellular memetic algorithm for the DNA fragment assembly problem. In: Wang, J. (ed.) *2008 IEEE World Congress on Computational Intelligence*, pp. 2656–2663, Hong Kong, 1–6 June 2008. IEEE Computational Intelligence Society, IEEE Press (2008)
72. Drezner, Z.: Extensive experiments with hybrid genetic algorithms for the solution of the quadratic assignment problem. *Comput. Oper. Res.* **35**(3), 717–736 Mar (2008)
73. Dumitrescu, I., Stützle, T.: Combinations of local search and exact algorithms. In: Raidl, G.R., et al. (eds.) *Applications of Evolutionary Computing: EvoWorkshops 2003*, vol. 2611, LNCS, pp. 212–224. Springer, Berlin, Heidelberg (2003)
74. El-Fallahi, A., Prins, C., Wolfler Calvo, R.: A memetic algorithm and a tabu search for the multi-compartment vehicle routing problem. *Comput. Or* **35**(5), 1725–1741 (2008)
75. Englemore, R., Morgan, T. (eds.): *Blackboard Systems*. Addison-Wesley, Reading, MA (1988)
76. Fernandes, S., Lourenço, H.: Hybrids combining local search heuristics with exact algorithms. In: Almeida, F., et al. (eds.) *V Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados*, pp. 269–274, Las Palmas, Spain (2007)
77. Fernández, E., Graña, M., Ruiz-Cabello, J.: An instantaneous memetic algorithm for illumination correction. In: *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pp. 1105–1110, Portland, Oregon, 20–23 June 2004. IEEE Press (2004)
78. Fischer, T., Bauer, K., Merz, P.: Distributed memetic algorithm for the routing and wavelength problem. In: Rudolph, G., et al. (eds.) *Parallel Problem Solving from Nature X*, vol. 5199, Lecture Notes in Computer Science, pp. 879–888. Springer, Berlin (2008)
79. Fischer, T., Merz, P.: A memetic algorithm for the optimum communication spanning tree problem. In: Bartz-Beielstein, T., et al. (eds.) *Hybrid Metaheuristics 2007*, vol. 4771, Lecture Notes in Computer Science, pp. 170–184. Springer, Berlin, Heidelberg (2007)
80. Fleury, G., Lacomme, P., Prins, C.: Evolutionary algorithms for stochastic arc routing problems. In: Raidl, G.R., et al. (eds.) *Applications of Evolutionary Computing*, vol. 3005, Lecture Notes in Computer Science, pp. 501–512. Springer, Berlin (2004)
81. Flórez-Revuelta, F., Casado-Díaz, J.M., Martínez-Bernabeu, L., Gómez-Hernández, R.: A memetic algorithm for the delineation of local labour markets. In: Rudolph, G., et al. (eds.) *Parallel Problem Solving from Nature X*, vol. 5199, Lecture Notes in Computer Science, pp. 1011–1020. Springer, Berlin (2008)
82. França, P.M., Gupta, J.N.D., Mendes, A.S., Moscato, P., Veltnik, K.J.: Evolutionary algorithms for scheduling a flowshop manufacturing cell with sequence dependent family setups. *Comput. Indus. Eng.* **48**, 491–506 (2005)

83. França, P.M., Mendes, A.S., Moscato, P.: A memetic algorithm for the total tardiness single machine scheduling problem. *Eur. J. Oper. Res.* **132**, 224–242 (2001)
84. França, P.M., Tin, G., Buriol, L.S.: Genetic algorithms for the no-wait flowshop sequencing problem with time restrictions. *Int. J. Prod. Res.* **44**(5), 939–957 (2006)
85. Freisleben, B., Merz, P.: A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems. In: Proceedings of the 1996 IEEE International Conference on Evolutionary Computation, pp. 616–621, Nagoya, Japan, 20–22 May 1996. IEEE Press (1996)
86. French, A.P., Robinson, A.C., Wilson, J.M.: Using a hybrid genetic-algorithm/branch and bound approach to solve feasibility and optimization integer programming problems. *J. Heuristics* **7**(6), 551–564 (2001)
87. Gallardo, J.E., Cotta, C., Fernández, A.J.: A hybrid model of evolutionary algorithms and branch-and-bound for combinatorial optimization problems. In: 2005 Congress on Evolutionary Computation, pp. 2248–2254, Edinburgh, UK, 2005. IEEE Press (2005)
88. Gallardo, J.E., Cotta, C., Fernández, A.J.: Solving the multidimensional knapsack problem using an evolutionary algorithm hybridized with branch and bound. In: Mira, J., Álvarez, J.R. (eds.) *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*, vol. 3562, Lecture Notes in Computer Science, pp. 21–30. Springer, Berlin (2005)
89. Gallardo, J.E., Cotta, C., Fernández, A.J.: A multi-level memetic/exact hybrid algorithm for the still life problem. In: Runarsson, T.P., et al. (eds.) *Parallel Problem Solving from Nature IX*, vol. 4193, Lecture Notes in Computer Science, pp. 212–221. Springer, Berlin (2006)
90. Gallardo, J.E., Cotta, C., Fernández, A.J.: A memetic algorithm with bucket elimination for the still life problem. In: Gottlieb, J., Raidl, G.R. (eds.) *Evolutionary Computation in Combinatorial Optimization*, vol. 3906, Lecture Notes in Computer Science, pp. 73–84, Budapest, 10–12 April 2006. Springer, Berlin, Heidelberg (2006)
91. Gallardo, J.E., Cotta, C., Fernández, A.J.: A memetic algorithm for the low autocorrelation binary sequence problem. In: Lipson, H. (ed.) *GECCO '07: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation Conference*, pp. 1226–1233. ACM Press, London, UK (2007)
92. Gallardo, J.E., Cotta, C., Fernández, A.J.: On the hybridization of memetic algorithms with branch-and-bound techniques. *IEEE Trans. Syst. Man Cybernet. Part B* **37**(1), 77–83 (2007)
93. Gallardo, J.E., Cotta, C., Fernández, A.J.: Reconstructing phylogenies with memetic algorithms and branch-and-bound. In: Bandyopadhyay, S., Maulik, U., Tsong-Li Wang, J., (eds.) *Analysis of Biological Data: A Soft Computing Approach*, pp. 59–84. World Scientific, Singapore (2007)
94. García, S., Cano, J.R., Herrera, F.: A memetic algorithm for evolutionary prototype selection: A scaling up approach. *Pattern Recogn.* **41**(8), 2693–2709 August (2008)
95. Glover, F., Laguna, M., Martí, R.: Fundamentals of scatter search and path relinking. *Control Cybernet.* **39**(3), 653–684 (2000)
96. González, M.A., Vela, C.R., Sierra, M.R., González Rodríguez, I., Varela, R.: Comparing schedule generation schemes in memetic algorithms for the job shop scheduling problem with sequence dependent setup times. In: Gelbukh, A.F., Reyes García, C.A. (eds.) *5th Mexican International Conference on Artificial Intelligence*, vol. 4293, Lecture Notes in Computer Science, pp. 472–482. Springer, Berlin, Heidelberg (2006)
97. González, M.A., Vela, C.R., Varela, R.: Scheduling with memetic algorithms over the spaces of semi-active and active schedules. In: *Artificial Intelligence and Soft Computing*, vol. 4029, Lecture Notes in computer Science, pp. 370–379. Springer, Berlin (2006)
98. González-Rodríguez, I., Vela, C.R., Puente, J.: A memetic approach to fuzzy job shop based on expectation model. In: *2007 IEEE International Conference on Fuzzy Systems*, pp. 1–6, London, UK, 23–26 July, (2007)
99. Gorges-Schleuter, M.: *ASPARAGOS: An asynchronous parallel genetic optimization strategy*. In: David Schaffer, J. (ed.) *Proceedings of the 3rd International Conference on Genetic Algorithms*, pp. 422–427. Morgan Kaufmann, San Francisco, CA (1989)

100. Gorges-Schleuter, M.: Explicit Parallelism of Genetic Algorithms through Population Structures. In: Schwefel, H.-P., Männer, R. (eds.) *Parallel Problem Solving from Nature*, pp. 150–159. Springer, Berlin, Heidelberg (1991)
101. Gottlieb, J.: Permutation-based evolutionary algorithms for multidimensional knapsack problems. In: Carroll, J., Damiani, E., Haddad, H., Oppenheim, D. (eds.) *ACM Symposium on Applied Computing 2000*, pp. 408–414. ACM Press, Como, Italy (2000)
102. Grim, P.: The undecidability of the spatialized prisoner's dilemma. *Theor. Decis.* **42**(1), 53–80 (1997)
103. Guillén, A., Pomares, H., González, J., Rojas, I., Herrera, L.J., Prieto, A.: Parallel multi-objective memetic RBFNNs design and feature selection for function approximation problems. In: Sandoval, F., Prieto, A., Cabestany, J., Graña, M. (eds.) *9th International Work-Conference on Artificial Neural Networks*, vol. 4507, Lecture Notes in Computer Science, pp. 341–350. Springer, Berlin, Heidelberg (2007)
104. Guimaraes, F.G., Campelo, F., Igarashi, H., Lowther, D.A., Ramírez, J.A.: Optimization of cost functions using evolutionary algorithms with local learning and local search. *IEEE Trans. Magn.* **43**(4), 1641–1644 (2007)
105. Guo, X.P., Wu, Z.M., Yang, G.K.: A hybrid adaptive multi-objective memetic algorithm for 0/1 knapsack problem. In: *AI 2005: Advances in Artificial Intelligence*, vol. 3809, Lecture Notes in Artificial Intelligence, pp. 176–185. Springer, Berlin (2005)
106. Guo, X.P., Yang, G.K., Wu, Z.M.: A hybrid self-adjusted memetic algorithm for multi-objective optimization. In: *4th Mexican International Conference on Artificial Intelligence*, vol. 3789, Lecture Notes in Computer Science, pp. 663–672. Springer, Berlin (2005)
107. Guo, X.P., Yang, G.K., Wu, Z.M., Huang, Z.H.: A hybrid fine-timed multi-objective memetic algorithm. *IEICE Trans. Fund. Electr. Commun. Comput. Sci.* **E89A**(3), 790–797 (2006)
108. Hart, W.E., Belew, R.K.: Optimizing an arbitrary function is hard for the genetic algorithm. In: Belew, R.K., Booker, L.B. (eds.) *Proceedings of the Fourth International Conference on Genetic Algorithms*, pp. 190–195. Morgan Kaufmann, San Mateo CA (1991)
109. Hart, W.E., Krasnogor, N., Smith, J.E.: *Recent Advances in Memetic Algorithms*, vol. 166, *Studies in Fuzziness and Soft Computing*. Springer, Berlin, Heidelberg (2005)
110. Hervás, C., Silva, M.: Memetic algorithms-based artificial multiplicative neural models selection for resolving multi-component mixtures based on dynamic responses. *Chemometr. Intell. Lab. Syst.* **85**(2), 232–242 (2007)
111. Hofstadter, D.R.: Computer tournaments of the prisoners-dilemma suggest how cooperation evolves. *Sci. Am.* **248**(5), 16–23 (1983)
112. Houck, C., Joines, J.A., Kay, M.G., Wilson, J.R.: Empirical investigation of the benefits of partial lamarckianism. *Evol. Comput.* **5**(1), 31–60 (1997)
113. Hsu, C.-H.: Uplink MIMO-SDMA optimisation of smart antennas by phase-amplitude perturbations based on memetic algorithms for wireless and mobile communication systems. *IET Commun.* **1**(3), 520–525 (2007)
114. Hsu, C.-H., Chou, P.-H., Shyr, W.-J., Chung, Y.-N.: Optimal radiation pattern design of adaptive linear array antenna by phase and amplitude perturbations using memetic algorithms. *Int. J. Innovat. Comput. Infor. Control* **3**(5), 1273–1287 (2007)
115. Hsu, C.-H., Shyr, W.-J.: Memetic algorithms for optimizing adaptive linear array patterns by phase-position perturbations. *Circuits Syst. Signal Process.* **24**(4), 327–341 (2005)
116. Hsu, C.-H., Shyr, W.-J.: Optimizing linear adaptive broadside array antenna by amplitude-position perturbations using memetic algorithms. In: Khosla, R., Howlett, R.J., Jain, L.C. (eds.) *9th International Conference on Knowledge-Based Intelligent Information and Engineering Systems*, vol. 3681, Lecture Notes in Computer Science, pp. 568–574. Springer, Berlin, Heidelberg (2005)
117. Hsu, C.-H., Shyr, W.-J., Chen, C.-H.: Adaptive pattern nulling design of linear array antenna by phase-only perturbations using memetic algorithms. In *First International Conference on Innovative Computing, Information and Control*, pp. 308–311, Beijing, China, 2006. IEEE Computer Society (2006)

118. Huang, D., Leung, C., Miao, C.: Memetic algorithm for dynamic resource allocation in multiuser OFDM based cognitive radio systems. In: Wang, J. (ed.) 2008 IEEE World Congress on Computational Intelligence, pp. 3861–3866, Hong Kong, 1–6 June 2008. IEEE Computational Intelligence Society, IEEE Press (2008)
119. Hulin, M.: An optimal stop criterion for genetic algorithms: A bayesian approach. In: Bäck, T. (ed.) Proceedings of the Seventh International Conference on Genetic Algorithms, pp. 135–143, Morgan Kaufmann, San Mateo, CA (1997)
120. Ishibuchi, H., Hitotsuyanagi, Y., Tsukamoto, N., Nojima, Y.: Use of heuristic local search for single-objective optimization in multiobjective memetic algorithms. In: Rudolph, G., et al. (eds.) Parallel Problem Solving from Nature X, vol. 5199, Lecture Notes in Computer Science, pp. 743–752. Springer Berlin, Berlin (2008)
121. Ishibuchi, H., Murata, T.: Multi-objective genetic local search algorithm. In: Fukuda, T., Furuhashi, T. (eds.) 1996 International Conference on Evolutionary Computation, pp. 119–124, Nagoya, Japan, 1996. IEEE Press (1996)
122. Ishibuchi, H., Murata, T.: Multi-objective genetic local search algorithm and its application to flowshop scheduling. *IEEE Trans. Syst. Man Cybernet.* **28**(3), 392–403 (1998)
123. Ishibuchi, H., Yoshida, T., Murata, T.: Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling. *IEEE Trans. Evol. Comput.* **7**(2), 204–223 (2003)
124. Jaszkiewicz, A.: Genetic local search for multiple objective combinatorial optimization. *Eur. J. Oper. Res.* **137**(1), 50–71 (2002)
125. Jaszkiewicz, A.: A comparative study of multiple-objective metaheuristics on the bi-objective set covering problem and the Pareto memetic algorithm. *Ann. Oper. Res.* **131**(1–4), 135–158 (2004)
126. Johnson, D.S., Papadimitriou, C.H., Yannakakis, M.: How easy is local search? *J. Comput. Syst. Sci.* **37**, 79–100 (1988)
127. Jones, T.C.: Evolutionary Algorithms, Fitness Landscapes and Search. PhD thesis, University of New Mexico (1995)
128. Karaoğlu, B., Topcuoğlu, H., Gürgen, F.: Evolutionary algorithms for location area management. In: Rothlauf, F., et al. (eds.) Applications of Evolutionary Computing, vol. 3449 LNCS, pp. 175–184, Lausanne, Switzerland, 30 March–1 April 2005. Springer, Berlin, Heidelberg (2005)
129. Kaveh, A., Shahrouzi, M.: Graph theoretical implementation of memetic algorithms in structural optimization of frame bracing layouts. *Eng. Comput.* **25**(1–2), 55–85 (2008)
130. Kim, S.-S., Smith, A.E., Lee, J.-H.: A memetic algorithm for channel assignment in wireless FDMA systems. *Comput. Or* **34**(6), 1842–1856 (2007)
131. Klau, G.W., Ljubić, I., Moser, A., Mutzel, P., Neuner, P., Pferschy, U., Raidl, G.R., Weiskircher, R.: Combining a memetic algorithm with integer programming to solve the prize-collecting Steiner tree problem. GECCO 04: Genet. Evol. Comput. Conf. **3102**(Part 1), 1304–1315 (2004)
132. Knowles, J., Corne, D.: Memetic Algorithms for Multiobjective Optimization: Issues, Methods and Prospects. In: Hart, W.E., Krasnogor, N., Smith, J. E. (eds.) Recent Advances in Memetic Algorithms, vol. 166, Studies in Fuzziness and Soft Computing, pp. 313–352. Springer, Berlin, Heidelberg (2005)
133. Knowles, J., Corne, D.W.: Approximating the non-dominated front using the pareto archived evolution strategy. *Evol. Comput.* **8**(2), 149–172 (2000)
134. Knowles, J.D., Corne, D.W.: M-PAES: A Memetic Algorithm for Multiobjective Optimization. In: Proceedings of the 2000 Congress on Evolutionary Computation (CEC00), pp. 325–332, Piscataway, NJ, 2000. IEEE Press (2000)
135. Knowles, J.D., Corne, D.W.: A Comparison of Diverse Approaches to Memetic Multiobjective Combinatorial Optimization. In: Wu, A.S. (ed.) Proceedings of the 2000 Genetic and Evolutionary Computation Conference Workshop Program, pp. 103–108, July 8–12, 2000, Las Vegas, Nevada (2000)

136. Kononova, A.V., Hughes, K.J., Pourkashanian, M., Ingham, D.B.: Fitness diversity based adaptive memetic algorithm for solving inverse problems of chemical kinetics. In: Srinivasan, D., Wang, L. (eds.) 2007 IEEE Congress on Evolutionary Computation, pp. 2366–2373, Singapore, 25–28 September 2007. IEEE Computational Intelligence Society, IEEE Press (2007)
137. Kononova, A.V., Ingham, D.B., Pourkashanian, M.: Simple scheduled memetic algorithm for inverse problems in higher dimensions: Application to chemical kinetics. In: Wang, J. (ed.) 2008 IEEE World Congress on Computational Intelligence, pp. 3906–3913, Hong Kong, 1–6 June 2008. IEEE Computational Intelligence Society, IEEE Press (2008)
138. Konstantinidis, A., Yang, K., Chen, H.-H., Zhang, Q.: Energy-aware topology control for wireless sensor networks using memetic algorithms. *Comput. Commun.* **30**(14–15), 2753–2764 (2007)
139. Kostikas, K., Fragakis, C.: Genetic programming applied to mixed integer programming. In: Keijzer, M., et al. (eds.) 7th European Conference on Genetic Programming, vol. 3003, Lecture Notes in Computer Science, pp. 113–124. Springer, Berlin (2004)
140. Krasnogor, N.: Self generating metaheuristics in bioinformatics: The proteins structure comparison case. *Genet. Program. Evol. Mach.* **5**(2), 181–201 June (2004)
141. Krasnogor, N., Blackburne, B.P., Burke, E.K., Hirst, J.D.: Multimeme algorithms for protein structure prediction. In: Merelo, J.J., et al. (eds.) Parallel Problem Solving From Nature VII, vol. 2439, Lecture Notes in Computer Science, pp. 769–778. Springer, Berlin (2002)
142. Krasnogor, N., Gustafson, S.M.: A study on the use of “self-generation” in memetic algorithms. *Nat. Comput.* **3**(1), 53–76 (2004)
143. Krasnogor, N., Smith, J.: Memetic algorithms: The polynomial local search complexity theory perspective. *J. Math. Model. Algorithms* **7**(1), 3–24 (2008)
144. Kretowski, M.: A memetic algorithm for global induction of decision trees. In: Geffert, V., et al. (eds.) 34th Conference on Current Trends in Theory and Practice of Computer Science, vol. 4910, Lecture Notes in Computer Science, pp. 531–540. Springer, Berlin, Heidelberg (2008)
145. Kubiak, M., Wesolek, P.: Accelerating local search in a memetic algorithm for the capacitated vehicle routing problem. In: Cotta, C., van Hemert, J.I. (eds.) Evolutionary Computation in Combinatorial Optimization, vol. 4446, Lecture Notes in Computer Science, pp. 96–107. Springer, Berlin, Heidelberg (2007)
146. Lacomme, P., Prins, C., Ramdane-Cherif, W.: Competitive memetic algorithms for arc routing problems. *Ann. Oper. Res.* **131**(1–4), 159–185 (2004)
147. Lacomme, P., Prins, C., Ramdane-Cherif, W.: Evolutionary algorithms for periodic arc routing problems. *Eur. J. Oper. Res.* **165**(2), 535–553 (2005)
148. Laguna, M., Martí, R.: Scatter Search. Methodology and Implementations in C. Kluwer, Boston, MA (2003)
149. Lamma, E., Pereira, L.M., Riguzzi, F.: Multi-agent logic aided lamarckian learning. Technical Report DEIS-LIA-00-004, Dipartimento di Elettronica, Informatica e Sistemistica, University of Bologna (Italy) (2000)
150. Lewis, H.R., Papadimitriou, C.H.: Elements of the Theory of Computation. Prentice-Hall, Inc., Upper Saddle River, NJ (1998)
151. Lewis, R., Paechter, B.: Finding feasible timetables using group-based operators. *IEEE Trans. Evol. Comput.* **11**(3), 397–413 (2007)
152. Li, B.-B., Wang, L., Liu, B.: An effective PSO-based hybrid algorithm for multiobjective permutation flow shop scheduling. *IEEE Trans. Syst. Man Cybernet. Part B* **38**(4), 818–831 (2008)
153. Li, J., Kwan, R.S.K.: A self adjusting algorithm for driver scheduling. *J. Heuristics* **11**(4), 351–367 (2005)
154. Lim, A., Rodrigues, B., Zhu, Y.: Airport gate scheduling with time windows. *Artifi. Intell. Rev.* **24**(1), 5–31 (2005)
155. Lim, D., Ong, Y.-S., Jin, Y., Sendhoff, B.: A study on metamodeling techniques, ensembles, and multi-surrogates in evolutionary computation. In: Thierens, D., et al. (eds.) GECCO '07:

- Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, vol. 2, pp. 1288–1295, London, 7–11 July 2007. ACM Press (2007)
- 156. Lim, K.K., Ong, Y.-S., Lim, M.H., Chen, X., Agarwal, A.: Hybrid ant colony algorithms for path planning in sparse graphs. *soft Comput.* **12**(10), 981–994 (2008)
  - 157. Lin, S., Kernighan, B.: An Effective Heuristic Algorithm for the Traveling Salesman Problem. *Oper. Res.* **21**, 498–516 (1973)
  - 158. Liu, B., Wang, L., Jin, Y.: An effective PSO-based memetic algorithm for flow shop scheduling. *IEEE Trans. Syst. Man Cybernet. Part B* **37**(1), 18–27 (2007)
  - 159. Liu, B., Wang, L., Jin, Y., Huang, D.: Designing neural networks using PSO-based memetic algorithm. In: Liu, D., Fei, S., Hou, Z.-G., Zhang, H., Sun, C. (eds.) 4th International Symposium on Neural Networks, vol. 4493, Lecture Notes in Computer Science, pp. 219–224. Springer, Berlin, Heidelberg (2007)
  - 160. Liu, B., Wang, L., Jin, Y.-H.: An effective hybrid particle swarm optimization for no-wait flow shop scheduling. *Int. J. Adv. Manuf. Tech.* **31**(9–10), 1001–1011 (2007)
  - 161. Liu, B., Wang, L., Jin, Y.-H., Huang, D.-X.: An effective PSO-based memetic algorithm for TSP. In: Intelligent Computing in Signal Processing and Pattern Recognition, vol. 345, Lecture Notes in Control and Information Sciences, pp. 1151–1156. Springer, Berlin, Heidelberg (2006)
  - 162. Liu, D., Tan, K.C., Goh, C.K., Ho, W.K.: A multiobjective memetic algorithm based on particle swarm optimization. *IEEE Trans. Syst. Man Cybernet., Part B* **37**(1), 42–50 (2007)
  - 163. Liu, Y.-H.: A memetic algorithm for the probabilistic traveling salesman problem. In: Wang, J. (ed.) 2008 IEEE World Congress on Computational Intelligence, pp. 146–152, Hong Kong, 1–6 June 2008. IEEE Computational Intelligence Society, IEEE Press (2008)
  - 164. Lozano, M., Herrera, F., Krasnogor, N., Molina, D.: Real-coded memetic algorithms with crossover hill-climbing. *Evol. Comput.* **12**(3), 273–302 (2004)
  - 165. Lumanpauw, E., Pasquier, M., Quek, C.: MNFS-FPM: A novel memetic neuro-fuzzy system based financial portfolio management. In: Srinivasan, D., Wang, L. (eds.) 2007 IEEE Congress on Evolutionary Computation, pp. 2554–2561, Singapore, 25–28 September 2007. IEEE Computational Intelligence Society, IEEE Press (2007)
  - 166. Maheswaran, R., Ponnambalam, S.G., Aravindan, C.: A meta-heuristic approach to single machine scheduling problems. *Int. J. Adv. Manuf. Tech.* **25**(7–8), 772–776 (2005)
  - 167. Maringer, D.G.: Finding the relevant risk factors for asset pricing. *Comput. Stat. Data Anal.* **47**(2), 339–352 (2004)
  - 168. Martínez-Estudillo, F.J., Hervás-Martínez, C., Martínez-Estudillo, A.C., Ortiz-Boyer, D.: Memetic algorithms to product-unit neural networks for regression. In: Cabestany, J., Prieto, A., Sandoval Hernández, F. (eds.) 8th International Work-Conference on Artificial Neural Networks, vol. 3512, Lecture Notes in Computer Science, pp. 83–90. Springer, Berlin, Heidelberg (2005)
  - 169. Mendes, A., Cotta, C., Garcia, V., França, P.M., Moscato, P.: Gene ordering in microarray data using parallel memetic algorithms. In: Skie, T., Yang, C.-S. (eds.) Proceedings of the 2005 International Conference on Parallel Processing Workshops, pp. 604–611, Oslo, Norway, 2005. IEEE Press (2005)
  - 170. Mendes, A., França, P.M., Lyra, C., Pissarra, C., Cavellucci, C.: Capacitor placement in large-sized radial distribution networks. *IEE Proceed.* **152**(4), 496–502 (2005)
  - 171. Mendes A., Linhares, A.: A multiple-population evolutionary approach to gate matrix layout-out. *Int. J. Syst. Sci.* **35**(1), 13–23 (2004)
  - 172. Mendes, A.S., França, P.M., Moscato, P.: Fitness landscapes for the total tardiness single machine scheduling problem. *Neural Netw. World* **2**(2), 165–180 (2002)
  - 173. Merz, P., Katayama, K.: Memetic algorithms for the unconstrained binary quadratic programming problem. *Biosystems* **78**(1–3), 99–118 (2004)
  - 174. Merz, P., Wolf, S.: Evolutionary local search for designing peer-to-peer overlay topologies based on minimum routing cost spanning trees. In: Runarsson, T.P., et al. (eds.) Parallel Problem Solving from Nature IX, vol. 4193, Lecture Notes in Computer Science, pp. 272–281. Springer, Berlin (2006)

175. Molina, D., Herrera, F., Lozano, M.: Adaptive local search parameters for real-coded memetic algorithms. In: Corne, D., et al. (eds.) *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, vol. 1, pp. 888–895, Edinburgh, Scotland, UK, 2–5 September 2005. IEEE Press (2005)
176. Molina, D., Lozano, M., Herrera, F.: Memetic algorithms for intense continuous local search methods. In: Blesa, M.J., et al. (eds.) *Hybrid Metaheuristics 2008*, vol. 5296, Lecture Notes in Computer Science, pp. 58–71. Springer, Berlin (2008)
177. Moscato, P.: On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms. Technical Report Caltech Concurrent Computation Program, Report. 826, California Institute of Technology, Pasadena, California, USA (1989)
178. Moscato, P.: An Introduction to Population Approaches for Optimization and Hierarchical Objective Functions: The Role of Tabu Search. *Ann. Oper. Res.* **41**(1–4), 85–121 (1993)
179. Moscato, P.: Memetic algorithms: A short introduction. In: Corne, D., Dorigo, M., Glover, F. (eds.) *New Ideas in Optimization*, pp. 219–234. McGraw-Hill, London, UK (1999)
180. Moscato, P., Cotta, C.: A gentle introduction to memetic algorithms. In: Glover, F., Kochenberger, G. (eds.) *Handbook of Metaheuristics*, pp. 105–144. Kluwer, Boston, MA (2003)
181. Moscato, P., Cotta, C.: Memetic algorithms. In: González, T. (ed.) *Handbook of Approximation Algorithms and Metaheuristics*, Chapter 22. Taylor & Francis, Boca Raton, FL (2006)
182. Moscato, P., Cotta, C., Mendes, A.: Memetic algorithms. In: Onwubolu, G.C., Babu, B.V. (eds.) *New Optimization Techniques in Engineering*, pp. 53–85. Springer, Berlin (2004)
183. Moscato, P., Mendes, A., Berretta, R.: Benchmarking a memetic algorithm for ordering microarray data. *Biosystems* **88**(1–2), 56–75 (2007)
184. Moscato, P., Mendes, A., Cotta, C.: Scheduling and production and control. In: Onwubolu, G.C., Babu, B.V. (eds.) *New Optimization Techniques in Engineering*, pp. 655–680. Springer, Berlin (2004)
185. Mühlenbein, H.: Evolution in time and space – The parallel genetic algorithm. In: Rawlins, J.E. (ed.) *Foundations of Genetic Algorithms*, pp. 316–337. Morgan Kaufmann Publishers, San Mateo, CA (1991)
186. Mühlenbein, H., Gorges-Schleuter, M., Krämer, O.: Evolution Algorithms in Combinatorial Optimization. *Parallel Comput.* **7**, 65–88 (1988)
187. Muruganandam, A., Prabhaharan, G., Asokan, P., Baskaran, V.: A memetic algorithm approach to the cell formation problem. *Int. J. Adv. Manuf. Tech.* **25**(9–10), 988–997 (2005)
188. Nagata, Y., Kobayashi, S.: Edge assembly crossover: A high-power genetic algorithm for the traveling salesman problem. In: Bäck, T. (ed.) *Proceedings of the Seventh International Conference on Genetic Algorithms*, pp. 450–457, San Mateo, CA, 1997. Morgan Kaufmann (1997)
189. Nakamaru, M., Matsuda, H., Iwasa, Y.: The evolution of social interaction in lattice models. *Sociol. Theor. Method.* **12**(2), 149–162 (1998)
190. Nakamaru, M., Nogami, H., Iwasa, Y.: Score-dependent fertility model for the evolution of cooperation in a lattice. *J. Theor. Biol.* **194**(1), 101–124 (1998)
191. Neri, F., Kotilainen, N., Vapa, M.: An adaptive global-local memetic algorithm to discover resources in P2P networks. In: Giacobini, M. et al. (eds.) *Applications of Evolutionary Computing*, vol. 4448, Lecture Notes in Computer Science, pp. 61–70. Springer, Berlin, Heidelberg (2007)
192. Neri, F., Kotilainen, N., Vapa, M.: A memetic-neural approach to discover resources in P2P networks. In: Cotta, C., van Hemert, J. (eds.) *Recent Advances in Evolutionary Computation for Combinatorial Optimization*, vol. 153, Studies in Computational Intelligence, pp. 113–129. Springer, Berlin (2008)
193. Neri, F., Tirronen, V.: On memetic differential evolution frameworks: A study of advantages and limitations in hybridization. In: Wang, J. (ed.) *2008 IEEE World Congress on Computational Intelligence*, pp. 2135–2142, Hong Kong, 1–6 June 2008. IEEE Computational Intelligence Society, IEEE Press (2008)

194. Neri, F., Toivanen, J., Casella, G.L., Ong, Y.-S.: An adaptive multimeme algorithm for designing HIV multidrug therapies. *IEEE/ACM Trans. Comput. Biol. Bioinfo.* **4**(2), 264–278 April (2007)
195. Neruda, R., Slusny, S.: Variants of memetic and hybrid learning of perceptron networks. In: 18th International Workshop on Database and Expert Systems Applications, pp. 158–162. IEEE Computer Society, Washington, DC (2007)
196. Nguyen, H.D., Yoshihara, I., Yamamori, K., Yasunaga, M.: Implementation of an effective hybrid GA for large-scale traveling salesman problems. *IEEE Trans. syst. Man Cybernet. Part B* **37**(1), 92–99 (2007)
197. Nguyen, Q.H., Ong, Y.-S., Krasnogor, N.: A study on the design issues of memetic algorithm. In: Srinivasan, D., Wang, L. (eds.) 2007 IEEE Congress on Evolutionary Computation, pp. 2390–2397, Singapore, 25–28 September 2007. IEEE Computational Intelligence Society, IEEE Press (2007)
198. Niedermeier, R., Rossmanith, P.: An efficient fixed parameter algorithm for 3-hitting set. Technical Report WSI-99-18, Universität Tübingen, Wilhelm-Schickard-Institut für Informatik, 1999. Technical Report, Revised version accepted in J. Discrete Algo. August (2000)
199. Niedermeier, R., Rossmanith, P.: A general method to speed up fixed-parameter-tractable algorithms. *Info. Process. Lett.* **73**, 125–129 (2000)
200. Noman, N., Iba, H.: Inferring gene regulatory networks using differential evolution with local search heuristics. *IEEE/ACM Trans. Comput. Biol. Bioinfo.* **4**(4), 634–647 October (2007)
201. Noman, N., Iba, H.: Accelerating differential evolution using an adaptive local search. *IEEE Trans. Evol. Comput.* **12**(1), 107–125 (2008)
202. Norman, M.G., Moscato, P.: A competitive and cooperative approach to complex combinatorial search. In: Proceedings of the 20th Informatics and Operations Research Meeting, pp. 3.15–3.29, Buenos Aires (1989)
203. Oakley, M.T., Barthel, D., Bykov, Y., Garibaldi, J.M., Burke, E.K., Krasnogor, N., Hirst, J.D.: Search strategies in structural bioinformatics. *Curr. Protein Peptide Sci.* **9**(3), 260–274 (2008)
204. Ong, Y.-S., Keane, A.J.: Meta-lamarckian learning in memetic algorithms. *IEEE Trans. Evol. Comput.* **8**(2), 99–110 (2004)
205. Ong, Y.-S., Lim, M.-H., Zhu, N., Wong, K.W.: Classification of adaptive memetic algorithms: a comparative study. *IEEE Trans. Syst. Man Cybernet. Part B* **36**(1), 141–152 (2006)
206. Özcan, E.: Memetic algorithms for nurse rostering. In: Yolum, P. et al. (eds.) Computer and Information Sciences – ISCIS 2005, 20 International Symposium (ISCIS), vol. 3733, Lecture Notes in Computer Science, pp. 482–492, Berlin Heidelberg, October 2005. Springer, Berlin, Heidelberg (2005)
207. Özcan, E., Onbaşıoglu, E.: Memetic algorithms for parallel code optimization. *Int. J. Parallel Program.* **35**(1), 33–61 (2007)
208. Palacios, P., Pelta, D., Blanco, A.: Obtaining biclusters in microarrays with population-based heuristics. In: Rothlauf, F., et al. (eds.) Applications of Evolutionary Computing, vol. 3907, Lecture Notes in Computer Science, pp. 115–126. Springer, Berlin (2006)
209. Pan, Q.-K., Wang, L., Qian, B.: A novel multi-objective particle swarm optimization algorithm for no-wait flow shop scheduling problems. *J. Eng. Manuf.* **222**(4), 519–539 (2008)
210. Pastorino, M.: Stochastic optimization methods applied to microwave imaging: A review. *IEEE Trans. Antennas Propag.* **55**(3, Part 1), 538–548 (2007)
211. Pastorino, M., Caorsi, S., Massa, A., Randazzo, A.: Reconstruction algorithms for electromagnetic imaging. *IEEE Trans. Instrument. Measure.* **53**(3), 692–699 (2004)
212. Paszkowicz, W.: Properties of a genetic algorithm extended by a random self-learning operator and asymmetric mutations: A convergence study for a task of powder-pattern indexing. *Anal. Chim. Acta* **566**(1), 81–98 (2006)
213. Peinado, M., Lengauer, T.: Parallel “go with the winners algorithms” in the LogP Model. In: Proceedings of the 11th International Parallel Processing Symposium, pp. 656–664, Los Alamitos, California, 1997. IEEE Computer Society Press (1997)

214. Petalas, Y.G., Parsopoulos, K.E., Vrahatis, M.N.: Memetic particle swarm optimization. *Ann. Oper. Res.* **156**(1), 99–127 (2007)
215. Petrovic, S., Burke, E.K.: University timetabling. In: Leung, J. (ed.) *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, Chapter 45. Chapman Hall/CRC Press, Boca Raton, FL (2004)
216. Petrovic, S., Patel, V., Yang, Y.: Examination timetabling with fuzzy constraints. In: *Practice and Theory of Automated Timetabling V*, vol. 3616, Lecture Notes in Computer Science, pp. 313–333. Springer, Berlin (2005)
217. Pirkwieser, S., Raidl, G.R.: Finding consensus trees by evolutionary, variable neighborhood search, and hybrid algorithms. In: Keijzer, M., et al. (eds.) *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pp. 323–330, Atlanta, GA, USA, 12–16 July 2008. ACM Press (2008)
218. Prins, C., Prodhon, C., Calvo, R.W.: A memetic algorithm with population management (MA|PM) for the capacitated location-routing problem. In: Gottlieb, J., Raidl, G.R. (eds.) *Evolutionary Computation in Combinatorial Optimization*, vol. 3906, Lecture Notes in Computer Science, pp. 183–194. Springer, Budapest, 10–12 April (2006)
219. Prodhom, C., Prins, C.: A memetic algorithm with population management (MA|PM) for the periodic location-routing problem. In: Blesa, M.J., et al. (eds.) *Hybrid Metaheuristics 2008*, vol. 5296, Lecture Notes in Computer Science, pp. 43–57. Springer-Verlag, Berlin (2008)
220. Puchinger, J., Raidl, G.R.: Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In: Mira, J., Álvarez, J.R. (eds.) *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*, vol. 3562, Lecture Notes in Computer Science, pp. 41–53. Springer, Berlin, Heidelberg (2005)
221. Puchinger, J., Raidl, G.R., Koller, G.: Solving a real-world glass cutting problem. In: Gottlieb, J., Raidl, G.R., (eds.) *4th European Conference on Evolutionary Computation in Combinatorial Optimization*, vol. 3004, Lecture Notes in Computer Science, pp. 165–176. Springer, Berlin (2004)
222. Puchinger, J., Raidl, G.R., Pferschy, U.: The core concept for the Multidimensional Knapsack Problem. In: Gottlieb, J., Raidl, G.R. (eds.) *Evolutionary Computation in Combinatorial Optimization*, vol. 3906, Lecture Notes in Computer Science, 10–12, April 2006 pp. 195–208. Springer, Budapest.
223. Qasem, M., Prugel-Bennett, A.: Complexity of Max-SAT using stochastic algorithms. In: Keijzer, M., et al. (eds.) *GECCO '08: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, pp. 615–616, Atlanta, GA, USA, 12–16 July 2008. ACM Press (2008)
224. Qian, B., Wang, L., Huang, D.-X., Wang, X.: Scheduling multi-objective job shops using a memetic algorithm based on differential evolution. *Int. J. Adv. Manuf. Tech.* **35**(9–10), 1014–1027 January (2008)
225. Quintero, A., Pierre, S.: On the design of large-scale cellular mobile networks using multi-population memetic algorithms. In: Abraham, A., et al. (eds.) *Engineering Evolutionary Intelligent Systems*, vol. 82, *Studies in Computational Intelligence*, pp. 353–377. Springer, Berlin, Heidelberg (2008)
226. Rabbani, M., Rahimi-Vahed, A., Torabi, S.A.: Real options approach for a mixed-model assembly line sequencing problem. *Int. J. Adv. Manuf. Tech.* **37**(11–12), 1209–1219 (2008)
227. Radcliffe, N.J.: The algebra of genetic algorithms. *Ann. Math. Artif. Intell.* **10**, 339–384 (1994)
228. Radcliffe, N.J., Surry, P.D.: Fitness Variance of Formae and Performance Prediction. In: Whitley, L.D., Vose, M.D. (eds.) *Proceedings of the 3rd Workshop on Foundations of Genetic Algorithms*, pp. 51–72, San Francisco, 1994. Morgan Kaufmann (1994)
229. Radcliffe, N.J., Surry, P.D.: Formal memetic algorithms. In: Fogarty, T., (ed.) *Evolutionary Computing: AISB Workshop*, vol. 865, Lecture Notes in Computer Science, pp. 1–16. Springer, Berlin (1994)

230. Rechenberg, I.: *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog Verlag, Stuttgart (1973)
231. Rocha, D.A.M., Goldberg, E.F.G., Goldberg, M.C.: A memetic algorithm for the biobjective minimum spanning tree problem. In: Gottlieb, J., Raidl, G.R. (eds.) *Evolutionary Computation in Combinatorial Optimization*, vol. 3906, Lecture Notes in Computer Science, pp. 222–233. Springer, Berlin, Heidelberg (2006)
232. Romero-Campero, F.J., Cao, H., Camara, M., Krasnogor, N.: Structure and parameter estimation for cell systems biology models. In: Keijzer, M., et al. (eds.) *GECCO '08: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, pp. 331–338, Atlanta, GA, USA, 12–16 July 2008. ACM Press (2008)
233. Rossi-Doria, O., Paechter, B.: A memetic algorithm for university course timetabling. In: *Combinatorial Optimisation 2004 Book of Abstracts*, pp. 56. Lancaster University Lancaster, UK (2004)
234. Santos, E.E., Santos, Jr, E.: Effective computational reuse for energy evaluations in protein folding. *Int. J. Artif. Intell. Tools* **15**(5), 725–739 (2006)
235. Schoenauer, M., Saveant, P., Vidal, V.: Divide-and-evolve: A new memetic scheme for domain-independent temporal planning. In: Gottlieb, J., Raidl, G.R. (eds.) *Evolutionary Computation in Combinatorial Optimization*, vol. 3906, Lecture Notes in Computer Science, pp. 247–260. Springer, Budapest.
236. Schönberger, J., Mattfeld, D.C., Kopfer, H.: Memetic algorithm timetabling for non-commercial sport leagues. *Eur. J. Oper. Res.* **153**, 102–116 (2004)
237. Schuetze, O., Sanchez, G., Coello Coello, C.A.: A new memetic strategy for the numerical treatment of multi-objective optimization problems. In: Keijzer, M., et al. (eds.) *GECCO '08: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, pp. 705–712, Atlanta, GA, USA, 12–16 July 2008. ACM Press (2008)
238. Schwefel, H.-P.: Evolution strategies: A family of non-linear optimization techniques based on imitating some principles of natural evolution. *Ann. Oper. Res.* **1**, 165–167 (1984)
239. Semet, Y., Schoenauer, M.: An efficient memetic, permutation-based evolutionary algorithm for real-world train timetabling. In: *Proceedings of the 2005 Congress on Evolutionary Computation*, pp. 2752–2759, Edinburgh, UK, 2005. IEEE Press (2005)
240. Sevaux, M., Jouplet, A., Oğuz, C.: Combining constraint programming and memetic algorithm for the hybrid flowshop scheduling problem. In: *ORBEL 19th Annual Conference of the SOGESCI-BVWB*, Louvain-la-Neuve, Belgium (2005)
241. Sevaux, M., Jouplet, A., Oğuz, C.: MLS+CP for the hybrid flowshop scheduling problem. In: *Workshop on the Combination of Metaheuristic and Local Search with Constraint Programming Techniques*. Nantes, France (2005)
242. Sheng, W., Howells, G., Fairhurst, M., Deravi, F.: A memetic fingerprint matching algorithm. *IEEE Trans. Info. Forensics Security* **2**(3, Part 1), 402–412 (2007)
243. Sheng, W., Liu, X., Fairhurst, M.: A niching memetic algorithm for simultaneous clustering and feature selection. *IEEE Trans. Knowl. Data Eng.* **20**(7), 868–879 (2008)
244. Smith, J.E.: Co-evolution of memetic algorithms: Initial investigations. In: Merelo, J.J., et al. (eds.) *Parallel Problem Solving From Nature VII*, vol. 2439, Lecture Notes in Computer Science, pp. 537–548. Springer, Berlin, Heidelberg (2002)
245. Smith, J.E.: Credit assignment in adaptive memetic algorithms. In: Lipson, H. (ed.) *GECCO '07: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation Conference*, pp. 1412–1419. ACM Press (2007)
246. Smith, J.E.: Coevolving memetic algorithms: A review and progress report. *IEEE Trans. Syst. Man Cybernet. Part B* **37**(1), 6–17 (2007)
247. Smith, J.E.: Self-adaptation in evolutionary algorithms for combinatorial optimization. In: Cotta, C., Sevaux, M., Sörensen, K. (eds.) *Adaptive and Multilevel Metaheuristics*, vol. 136, *Studies in Computational Intelligence*, pp. 31–57. Springer, Berlin (2008)
248. Soak, S.-M., Lee, S.-W., Mahalik, N.P., Ahn, B.-H.: A new memetic algorithm using particle swarm optimization and genetic algorithm. In: *Knowledge-based Intelligent Information and Engineering Systems*, vol. 4251, Lecture Notes in Artificial Intelligence, pp. 122–129. Springer, Berlin, Heidelberg (2006)

249. Sørensen, K., Sevaux, M.: MA | PM: memetic algorithms with population management. *Comput. Or.* **33**, 1214–1225 (2006)
250. Spieth, C., Streichert, F., Supper, J., Speer, N., Zell, A.: Feedback memetic algorithms for modeling gene regulatory networks. In: Proceedings of the IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB 2005), pp. 61–67, La Jolla, CA, 2005. IEEE Press (2005)
251. Sudholt, D.: Memetic algorithms with variable-depth search to overcome local optima. In: Keijzer, M., et al. (eds.) GECCO '08: Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation, pp. 787–794, Atlanta, GA, USA, 12–16 July 2008. ACM Press (2008)
252. Surry, P.D., Radcliffe, N.J.: Inoculation to initialise evolutionary search. In: Fogarty, T.C., (ed.) Evolutionary Computing: AISB Workshop, vol. 1143, Lecture Notes in Computer Science, pp. 269–285. Springer, Berlin, Heidelberg (1996)
253. Syswerda, G.: Uniform crossover in genetic algorithms. In: Schaffer, J.D. (ed.) Proceedings of the 3rd International Conference on Genetic Algorithms, pp. 2–9, San Mateo, CA, 1989. Morgan Kaufmann (1989)
254. Tagawa, K., Matsuoka, M.: Optimum design of surface acoustic wave filters based on the Taguchi's quality engineering with a memetic algorithm. In: Runarsson, T.P., et al. (eds.) Parallel Problem Solving from Nature IX, vol. 4193, Lecture Notes in Computer Science, pp. 292–301. Springer, Berlin (2006)
255. Tang, J., Lim, M.H., Ong, Y.-S., Er, M.J.: Parallel memetic algorithm with selective local search for large scale quadratic assignment problems. *Int. J. Innov. Comput. Info. Control* **2**(6), 1399–1416 (2006)
256. Tang, M., Yao, X.: A memetic algorithm for VLSI floorplanning. *IEEE Trans. Syst. Man Cybernet. Part B* **37**(1), 62–69 (2007)
257. Tavakkoli-Moghaddam, R., Rahimi-Vahed, A.R.: A memetic algorithm for multi-criteria sequencing problem for a mixed-model assembly line in a JIT production system. In: 2006 IEEE Congress on Evolutionary Computation (CEC'2006), pp. 10350–10355, Vancouver, BC, Canada, July 2006. IEEE (2006)
258. Tavakkoli-Moghaddam, R., Safaei, N., Babakhani, M.: Solving a dynamic cell formation problem with machine cost and alternative process plan by memetic algorithms. In: International Symposium on Stochastic Algorithms: Foundations and Applications, LNCS, vol. 3, Springer, Berlin, Heidelberg (2005)
259. Tavakkoli-Moghaddam, R., Saremi, A.R., Ziaeef, M.S.: A memetic algorithm for a vehicle routing problem with backhauls. *Appl. math. Comput.* **181**(2), 1049–1060 (2006)
260. Tenne, Y., Armfield, S.W.: A memetic algorithm using a trust-region derivative-free optimization with quadratic modelling for optimization of expensive and noisy black-box functions. In: Yang, S., Ong, Y.-S., Jin, Y. (eds.) Evolutionary Computation in Dynamic and Uncertain Environments, vol. 51, Studies in Computational Intelligence, pp. 389–415. Springer, Berlin, Heidelberg (2007)
261. Tirronen, V., Neri, F., Kärkkäinen, T., Majava, K., Rossi, T.: A memetic differential evolution in filter design for defect detection in paper production. In: Giacobini, M., et al. (eds.) *Applications of Evolutionary Computing*, volume 4448 of *Lecture Notes in Computer Science*, pages 320–329. Springer-Verlag (2007)
262. Togelius, J., Schaul, T., Schmidhuber, J., Gómez, F.: Countering poisonous inputs with memetic neuroevolution. In: Rudolph, G., et al. (eds.) *Parallel Problem Solving from Nature X*, volume 5199 of *Lecture Notes in Computer Science*, pages 610–619, Berlin Heidelberg, 2008. Springer-Verlag.
263. Tricoire, F.: Vehicle and personnel routing optimization in the service sector: application to water distribution and treatment. *4OR-A Quart. J. Oper. Res.* **5**(2), 165–168 (2007)
264. Tse, S.-M., Liang, Y., Leung, K.-S., Lee, K.-H., Mok, T.S.K.: A memetic algorithm for multiple-drug cancer chemotherapy schedule optimization. *IEEE Trans. Syst. Man Cybernet. Part B* **37**(1), 84–91 (2007)
265. Tseng, H.E., Wang, W.P., Shih, H.Y.: Using memetic algorithms with guided local search to solve assembly sequence planning. *Expert. Syst. Appl.* **33**(2), 451–467 (2007)

266. Ulungu, E.L., Teghem, J., Fortemps, P., Tuyttens, D.: MOSA method: A tool for solving multiobjective combinatorial optimization problems. *J. Multi-Criteria Deci. Anal.* **8**(4), 221–236 (1999)
267. Varela, R., Puente, J., Vela, C.R.: Some issues in chromosome codification for scheduling with genetic algorithms. In: Castillo, L., Borrajo, D., Salido, M.A., Oddi, A. (eds.) *Planning, Scheduling and Constraint Satisfaction: From Theory to Practice*, vol. 117, *Frontiers in Artificial Intelligence and Applications*, pp. 1–10. IOS Press (2005)
268. Varela, R., Serrano, D., Sierra, M.: New codification schemas for scheduling with genetic algorithms. In: Mira, J., Álvarez, J.R. (eds.) *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach*, vol. 3562, *Lecture Notes in Computer Science*, pp. 11–20. Springer, Berlin (2005)
269. Volk, J., Herrmann, T., Wuethrich, K.: Automated sequence-specific protein NMR assignment using the memetic algorithm match. *J. Biomol. NMR* **41**(3), 127–138 (2008)
270. Wang, J.: A memetic algorithm with genetic particle swarm optimization and neural network for maximum cut problems. In: Li, K., Fei, M., Irwin, G.W., Ma, S. (eds.) *International Conference on Life System Modeling and Simulation*, vol. 4688, *Lecture Notes in Computer Science*, pp. 297–306. Springer, Berlin, Heidelberg (2007)
271. Wang, Y., Qin, J.: A memetic-clustering-based evolution strategy for traveling salesman problems. In: Yao, J., et al. (eds.) *2nd International Conference on Rough Sets and Knowledge Technology*, vol. 4481, *Lecture Notes in Computer Science*, pp. 260–266. Springer, Berlin, Heidelberg (2007)
272. Wanner, E.F., Guimarães, F.G., Takahashi, R.H.C., Fleming, P.J.: Local search with quadratic approximations into memetic algorithms for optimization with multiple criteria. *Evol. Comput.* **16**(2), 185–224 (2008)
273. Wanner, E.F., Guimarães, F.G., Takahashi, R.H.C., Lowther, D.A., Ramírez, J.A.: Multiobjective memetic algorithms with quadratic approximation-based local search for expensive optimization in electromagnetics. *IEEE Trans. Magnet.* **44**(6), 1126–1129 (2008)
274. Whitley, D.: Using reproductive evaluation to improve genetic search and heuristic discovery. In: Grefenstette, J.J. (ed.) *Proceedings of the 2nd International Conference on Genetic Algorithms and their Applications*, pp. 108–115, Cambridge, MA, July 1987. Lawrence Erlbaum Associates (1987)
275. Williams, T.L., Smith, M.L.: The role of diverse populations in phylogenetic analysis. In: Keijzer, M., et al. (eds.) *GECCO 2006: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, vol. 1, pp. 287–294, Seattle, Washington, USA, 8–12 July 2006. ACM Press (2006)
276. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. *IEEE Trans. Evol. Comput.* **1**(1), 67–82 (1997)
277. Xhafa, F., Duran, B.: Parallel memetic algorithms for independent job scheduling in computational grids. In: Cotta, C., van Hemert, J. (eds.) *Recent Advances in Evolutionary Computation for Combinatorial Optimization*, vol. 153, *Studies in Computational Intelligence*, pp. 219–239. Springer, Berlin (2008)
278. Yang, J.-H., Sun, L., Lee, H.P., Qian, Y., Liang, Y.-C.: Clonal selection based memetic algorithm for job shop scheduling problems. *J. Bionic Eng.* **5**(2), 111–119 (2008)
279. Yannakakis, M.: Computational complexity. In: Aarts, E.H.L., Lenstra, J.K. (eds.) *Local Search in Combinatorial Optimization*, pp. 19–55. Wiley, Chichester (1997)
280. Yeh, W.-C.: An efficient memetic algorithm for the multi-stage supply chain network problem. *Int. J. Adv. Manuf. Tech.* **29**(7–8), 803–813 (2006)
281. Zhao, X.: Advances on protein folding simulations based on the lattice HP models with natural computing. *Appl. Soft Comput.* **8**(2), 1029–1040 (2008)
282. Zhen, Z., Wang, Z., Gu, Z., Liu, Y.: A novel memetic algorithm for global optimization based on PSO and SFLA. In: Kang, L., Liu, Y., Zeng, S.Y. (eds.) *2nd International Symposium on Advances in Computation and Intelligence*, vol. 4683, *Lecture Notes in Computer Science*, pp. 127–136. Springer (2007)

283. Zhou, Z., Ong, Y.-S., Lim, M.-H., Lee, B.-S.: Memetic algorithm using multi-surrogates for computationally expensive optimization problems. *Soft Comput.* **11**(10), 957–971 (2007)
284. Zhu, Z., Ong, Y.-S.: Memetic algorithms for feature selection on microarray data. In: Liu, D., et al. (eds.) 4th International Symposium on Neural Networks, vol. 4491, Lecture Notes in Computer Science, pp. 1327–1335. Springer, Berlin, Heidelberg (2007)
285. Zhu, Z., Ong, Y.-S., Dash, M.: Markov blanket-embedded genetic algorithm for gene selection. *Pattern Recogn.* **40**(11), 3236–3248 (2007)
286. Zhu, Z., Ong, Y.-S., Dash, M.: Wrapper-filter feature selection algorithm using a memetic framework. *IEEE Trans. Syst. Man Cybernet. Part B* **37**(1), 70–76 (2007)
287. Zitzler, E., Laumanns, M., Bleuler, S.: A Tutorial on Evolutionary Multiobjective Optimization. In: Gandibleux, X., et al. (eds.) Metaheuristics for Multiobjective Optimisation, vol. 535, Lecture Notes in Economics and Mathematical Systems. Springer, Berlin, Heidelberg (2004)



# Chapter 7

## Genetic Programming

William B. Langdon, Robert I. McKay and Lee Spector

**Abstract** Welcome to genetic programming, where the forces of nature are used to automatically evolve computer programs. We give a flavour of where GP has been successfully applied (it is far too wide an area to cover everything) and interesting current and future research but start with a tutorial of how to get started and finish with common pitfalls to avoid.

### 7.1 Introduction

Getting computers to automatically solve problems is central to artificial intelligence, machine learning and the broad area covered by what Turing called “machine intelligence” [1]. As we shall show, this is what genetic programming is actually doing today.

Genetic programming [2] works by applying the power of evolution by natural selection [3] to artificial populations inside your computer, cf. Figure 7.1. Unlike in nature, you decide who is fit, who survives and who has children. Like nature, children are not identical to their parents but suffer random mutations and can be created by fusing together the genetic characteristics of their parents. Unlike other approaches to evolving expressions, genetic programming works because it has defined a way of representing expressions whereby they can be randomly mutated

---

William B. Langdon

Department of Computer Science, University College London, Gower Street, London WC1E 6BT, UK

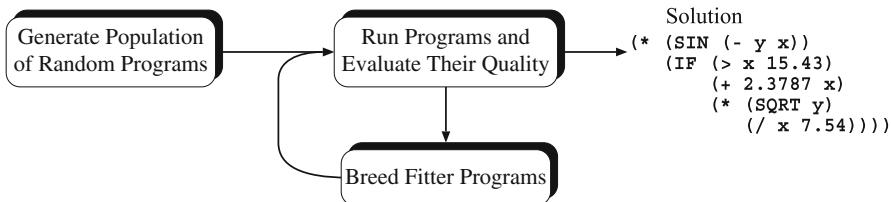
e-mail: w.langdon@cs.ucl.ac.uk

Robert I. McKay

School of Computer Science and Engineering, Seoul National University, Seoul, Korea  
e-mail: rimsnucse@gmail.com

Lee Spector

School of Cognitive Science, Hampshire College, Amherst, MA, USA  
e-mail: lspector@hampshire.edu



**Fig. 7.1** The basic control flow for GP, where survival of the fittest is used to find solutions.

and still be syntactically correct expression which can be evaluated. Like nature, many mutants are not as fit as their parents but, like nature, every so often, a mutant is created which is better. Similarly children produced by sex have genes which are a random combination of parental genes. Again, every once in a while an improved combination is found and the offspring program is selected for prospers and in subsequent generations copies of it spread through the evolving population.

Genetic programming can be thought of as like domesticated animals and plants, where improvements have been made by breeders progressively selecting preferred characteristics. (Darwin studied the records of breeders of domesticated pigeons.) Thus you too must impose a direction on evolution, e.g. to control a robot, design a radio aerial or find a genetic component of breast cancer survival, you must select programs that are better at doing it. For example, given the cause of death and life span of 253 Swedish women cancer patients, you might select a program which correctly predicted more cases of survival for more than 8 years after surgery than one which was less accurate.

With large populations and/or many generations, selecting individual programs becomes too tedious to do by hand. Instead we pass the job to a computerized automatic “fitness function”. On your behalf, it prefers better programmes over the less good. Ultimately it is your fitness function which guides the evolution of your population by selecting who will survive and who will have children. The fitness function is literally a matter of life or death.

There are several fine books on GP ([2, 4] and [5] leap to mind); however, we strongly encourage *doing* GP as well as reading about it. There are many good free (unsupported) GP implementations (e.g. *lilGP*, *ECJ*, *Beagle*<sup>1</sup> and *TinyGP*) but it is not so hard to write your own.

### 7.1.1 Overview

The next section describes the main parts of genetic programming, while Section 7.3 describes how you put them together to get a working system. Next Sections 7.4 and 7.5 describe advanced GP techniques. We survey the enormous variety of

<sup>1</sup> Darwin was the naturalist onboard HMS Beagle for 5 years [6].

applications of GP in Section 7.6. This is followed by a collection of troubleshooting suggestions (Section 7.7) and by our conclusions (Section 7.8).

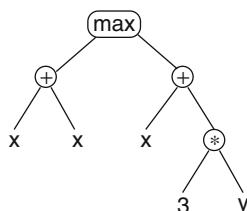
## 7.2 Representation, Initialization and Operators in Tree-Based GP

### 7.2.1 Representation

In artificial intelligence it has become accepted wisdom that how information about the application and its solution is stored (i.e. represented internally within the computing system) and manipulated by it, is crucial to successful implementation. Huge effort is spent by very clever people on designing the correct representation.

Genetic programming has ignored this. In GP, the evolved program contains the solution and “representation” refers to the language evolution uses to write the program. The same representation might be used in a program evolved to predict breast cancer survival as one evolved to find insider trading in a stock market.

In GP, programs are usually expressed as *syntax trees* rather than as lines of code. For example Figure 7.2 shows the tree representation of the program  $\max(x+x, x+3*y)$ . The variables and constants in the program ( $x$ ,  $y$  and 3) are leaves of the tree. In GP they are called *terminals*, while the arithmetic operations (+, \* and max) are internal nodes called *functions*. The sets of allowed functions and terminals together form the *primitive set* of a GP system.



**Fig. 7.2** GP syntax tree representing  $\max(x+x, x+3*y)$ .

It is common to represent expressions in *prefix* notation, e.g.  $\max(x+x, x+3*y)$  becomes  $(\max (+ x x) (+ x (* 3 y)))$ . Usually the number of arguments a function takes is known, e.g.  $\sin$  has one argument but  $*$  has two. When the arity is known, the brackets in prefix-notation expressions are not needed. This means trees can be represented as simple linear sequences. Usually this is much more efficient than tree-based representation of programs, which require the storage and management of numerous pointers. In effect, the function’s name gives its arity and from the arities the brackets can be inferred. For example, the

expression `(max (+ x x) (+ x (* 3 y)))` can be written unambiguously as `max + x x + x * 3 y`.

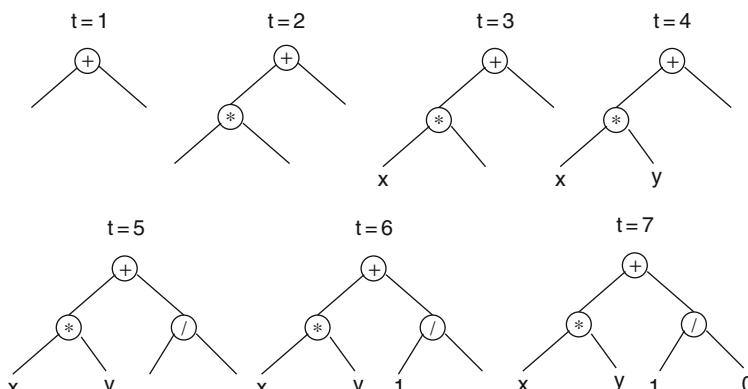
The choice of whether to use such a linear representation or an explicit tree representation is typically guided by convenience, efficiency, the genetic operations being used (some may be more easily or more efficiently implemented in one representation), and other data one may wish to collect during runs. (It is sometimes useful to attach additional information to nodes, which may be easier to implement if they are explicitly represented).

Tree representations are the most common in GP. However, there are other important representations including linear [4, 7–9] and graph [10–12] based programs.

### 7.2.2 Initializing the Population

As with other evolutionary algorithms, in GP the individuals in the initial population are typically randomly generated. There are a number of different approaches to generating this random initial population, e.g. [13]. However, we will describe two of the simplest methods (the *full* and *grow* methods) and the most widely used combination of the two known as *Ramped half-and-half* [2].

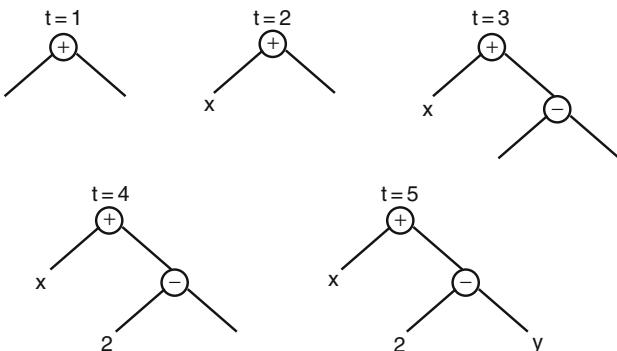
In both the *full* and the *grow* methods, the initial individuals are generated so that they do not exceed a maximum depth you decide. The *depth* of a node is the number of edges that need to be traversed to reach the node starting from the tree's root node (depth 0). The depth of a tree is the depth of its deepest leaf (e.g. the tree in Figure 7.2 has a depth of 3). The *full* method generates full trees (i.e. all leaves are at the same depth). It does this by choosing at random from the available functions (known as the function set) until the maximum tree depth is reached. Then the tree is finished by adding randomly chosen leafs from the available terminals (known as the terminal set). Figure 7.3 shows a series of snapshots of the construction of a full



**Fig. 7.3** Creation of a full tree having maximum depth 2 using *full* initialization ( $t$  = time) [53].

tree of depth 2. The children of the  $*$  and  $/$  nodes must be leaves or otherwise the tree would be too deep. Thus, at steps  $t = 3$ ,  $t = 4$ ,  $t = 6$  and  $t = 7$  a terminal must be chosen. (In this example leafs  $x$ ,  $y$ , 1 and 0 were randomly chosen).

Although, the `full` method generates trees where all the leaves are at the same depth, this does not necessarily mean that all initial trees will have an identical number of nodes (often referred to as the *size* of a tree) or the same shape. This happens only if all the functions have the same arity (i.e. have the same number of inputs.) Nonetheless, even when mixed-arity primitive sets are used, the range of program sizes and shapes produced by the `full` method may be limited. The `grow` method creates trees of more varied sizes and shapes. Nodes are selected from the whole primitive set (i.e. functions and terminals) until the depth limit is reached. Once the depth limit is reached only terminals may be chosen (just as in the `full` method). Figure 7.4 illustrates growing a tree with depth limit of 2. In Figure 7.4 ( $t = 2$ ) the first argument of the  $+$  root node happens to be a terminal. This prevents that branch from growing any more. The other argument is a function  $(-)$ . It can grow one level before its arguments are forced to be terminals to ensure that the resulting tree does not exceed the depth limit. C++ code for a recursive implementation of both the `full` and the `grow` methods is given below.



**Fig. 7.4** Creation of a five node tree using the `grow` initialization method with a maximum depth of 2 ( $t$  = time). A terminal is chosen at  $t = 2$ , causing the left branch of the root to be closed at that point even though the maximum depth had not been reached [5].

Because neither the `grown` or the `full` method provides a very wide array of sizes or shapes on their own, Koza proposed a combination called *ramped half-and-half* [2]. Half the initial population is constructed using `full` and half is constructed using `grow`. This is done using a range of depth limits (hence the term “ramped”) to help ensure that we generate trees having a variety of sizes and shapes.

While these methods are easy to implement and use, the sizes and shapes of the trees generated are highly sensitive to the number of functions, the number of inputs they have and the number of terminals. This makes it difficult to control the sizes and shapes of the trees. For example, if there are many more terminals than

functions, the *grow* method will almost always generate very short trees regardless of the depth limit. Similarly, if the number of functions is considerably greater than the number of terminals, then the *grow* method will be like the *full* method.

```
//Choose desired depth uniformly at random between min and max depth.
//Choose either full or grow.
SubInit(rnd(max_depth-min_depth)+min_depth, rnd(2), min_depth);

void Individual::SubInit(int depth, BOOL isfull, int min_depth) {
if (depth <= 0)
    i=rand_terminal(); // terminal required
else if (isfull || min_depth>0)
    i=rand_function(); // function required
else {//grow: terminal allowed 50% of the time
    if (rnd(2)) // terminal required
        i=rand_terminal();
    else // node required
        i=rand_function();
}
SETNODE(code[ip],i); //store opcode in Individual
ip++;

for(int a=0;a<argnum(i);a++) {
    SubInit(depth-1,isfull, min_depth-1, tree);
}
}
```

C++ code fragment to create a random tree. For efficiency the tree is flattened and stored in array *code* (access is via macro *SETNODE*). *SubInit* recursively calls itself until it reaches a leaf of the tree. (Based upon Andy Singleton's GPquick.)

The initial population need not be entirely random. If something is known about likely properties of the desired solution, trees having these properties can be used to seed the initial population.

### 7.2.3 Selection

As with other evolutionary algorithms, in GP better individuals are more likely to have more child programs than inferior individuals. Tournament selection is most often used, followed by fitness-proportionate selection [4], but any standard evolutionary algorithm selection mechanism (e.g. stochastic universal sampling) can be used.

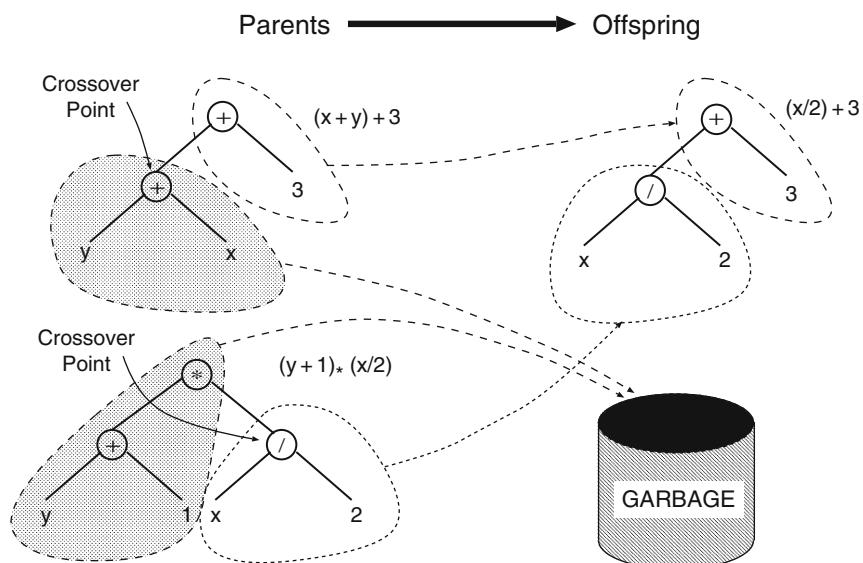
In *tournament selection* a number of individuals are chosen at random from the population. These are compared with each other and the best of them is chosen to be the parent. When doing crossover, two parents are needed and, so, two selection tournaments are made. Note that tournament selection only looks at which program is better than another. It does not need to know how much better. This effectively

automatically rescales fitness, so that the selection pressure is constant. Thus, a single extraordinarily good program cannot immediately swamp the next generation with its children. If it did, this would lead to a rapid loss of diversity with potentially disastrous consequences for a run. Conversely, tournament selection amplifies small differences in fitness to prefer the better program even if it is only marginally superior to the other individuals in a tournament.

Tournament selection, due to the random selection of programs to be included in the tournament, is inherently noisy. So, while preferring the best, tournament selection does ensure that even below average programs have some chance of having children. Since tournament selection is easy to implement and provides automatic fitness rescaling, it is commonly used in GP.

### 7.2.4 Recombination and Mutation

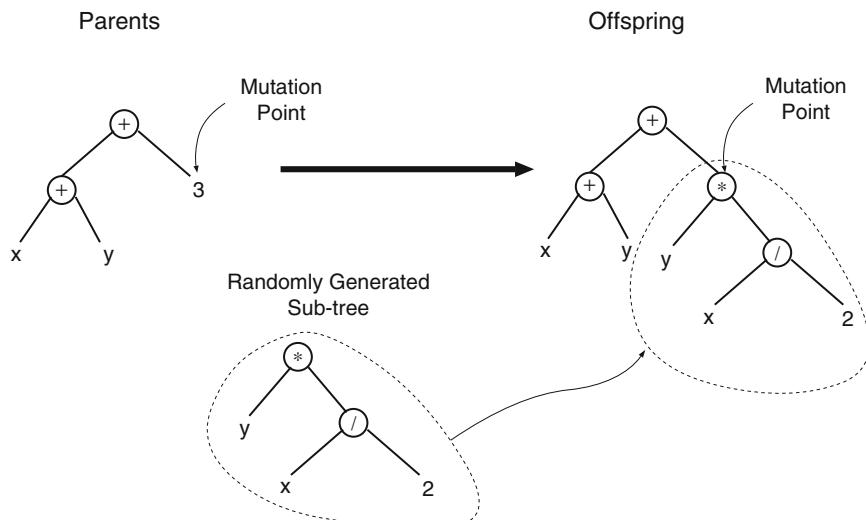
Crossover (recombination) and mutation in GP are very different from crossover and mutation in other evolutionary algorithms. The most commonly used form of crossover is *subtree crossover*. Given two parents, subtree crossover randomly (and independently) selects a *crossover point* (a node) in each parent tree. Then, it creates the offspring by replacing the subtree rooted at the crossover point in a copy of the first parent with a copy of the subtree rooted at the crossover point in the second parent [2], as illustrated in Figure 7.5.



**Fig. 7.5** Example of subtree crossover. Note that the trees on the left are actually *copies* of the parents. So, their genetic material can freely be used without altering the original individuals [5].

Typical GP primitive sets lead to trees with an average arity of at least two. This means most of the program will be leaves. So if crossover points were chosen uniformly, crossovers would frequently swap very small subtrees (even just leafs), i.e. exchange only very small amounts of genetic material. Whereas in nature (and many GAs) often both parents contribute more-or-less equally to their offspring's genetic code. To counter this, Koza suggested the widely used approach of choosing functions 90% of the time and leaves 10% of the time [2]. Many other types of crossover and mutation of GP trees are possible (see [5, pp 42–44]).

The most commonly used form of mutation in GP is *subtree mutation*. It randomly selects a mutation point in a tree and substitutes the subtree rooted there with a randomly generated subtree (cf. Figure 7.6 and [15]).



**Fig. 7.6** Example of subtree mutation [5].

Another common form of mutation is *point mutation*, which is GP's rough equivalent of the bit-flip mutation used in genetic algorithms [14]. In point mutation, a random node is selected and the primitive stored there is replaced with a different random primitive of the same arity taken from the primitive set. If no other primitives with that arity exist, nothing happens to that node (but other nodes may still be mutated). When subtree mutation is applied, it changes exactly one subtree. On the other hand, every node in the tree has a small probability of being mutated by point mutation. This means point mutation independently changes a random number of nodes.

In GP normally only one genetic operator is used to create each child. Which one is used is chosen at random. Typically, crossover is applied with the highest probability, the *crossover rate* often being 90% or higher. On the contrary, the *mutation rate* is much smaller, typically being in the region of 1%. If the sum of all

the probabilities comes to less than 100% the remaining offspring are created simply by copying better individuals from the current population. (This is known as *reproduction*.)

## 7.3 Getting Ready to Run Genetic Programming

### 7.3.1 Step 1: Terminal Set

GP is not typically used to evolve programs in the familiar languages people normally write programs in. Instead simpler programming languages are used. Indeed GP can usually be thought of as evolving executable expressions rather than fully fledged programs. The first two preparatory steps, the definition of the terminal and function sets, specify the language. Together they define the ingredients that are available to GP to create computer programs.

Typically the terminal set contains *the program's inputs* (e.g.  $x$ ,  $y$ , cf. Table 7.1). It may also contain *functions with no arguments*. They might be needed because they return different values each time they are used, such as a function which returns random numbers or returns the distance from a robot to an obstacle or because the function produces *side effects*. Functions with side effects may change some global data structures, draw on the screen, print to a file, control the motors of a robot, etc.

**Table 7.1** Examples of primitives in GP function and terminal sets.

Function set		Terminal set	
<i>Kind of primitive</i>	<i>Example</i>	<i>Kind of primitive</i>	<i>Example</i>
Arithmetic	$+, *, /$	Variables	$x, y$
Mathematical	$\sin, \cos, \exp$	Constant values	3, 0..45
Boolean	AND, OR, NOT	0-arity functions	rand, go_left
Conditional	IF-THEN-ELSE		
Looping	FOR, REPEAT		

Often an evolved program will need access to constants. We do not know in advance what their values will be, so GP chooses some randomly. In some implementations the number of constants is limited and it may be that new ones cannot be created during the GP run. Instead their values must be chosen as the population is initialized. Typically this is done by a special terminal that represents an *ephemeral random constant*. Every time it is chosen (either at the start or when a new subtree is created by mutation), a different random value is generated. This is used for that *particular* terminal and remain fixed for the rest of the run.

### 7.3.2 Step 2: Function Set

The function set typically contains only the arithmetic functions ( $+, -, *, /$ ). However, all sorts of other functions and constructs typically encountered in computer

programs can be used, see Table 7.1. Sometimes specialized functions or terminals, which are designed to solve particular problems are used. For example, if the goal is to evolve art, then the function set might include such actions as `select_from_pallet` and `paint`.

For GP to work effectively, most function sets are required to have an important property known as *closure* [2]. Closure can be broken down into *type consistency* and *evaluation safety*. Finally the primitive set must be able to (i.e. must be *sufficient* to) express solutions to the problem.

### 7.3.2.1 Type Consistency

Crossover (Section 7.2.4) can mix and join nodes arbitrarily. So it is important that *any* subtree can be used as any argument for every function in the function set. The simplest way to achieve this is to ensure all functions return values of the same type and that each of their arguments also have this type. For example, `+`, `-`, `*`, and `/` can be defined so that they each take two integer arguments and return an integer. The terminals would also be integers. (Automatic-type conversion, e.g., between Booleans and integers, default values and polymorphic functions, can also be used to ensure that crossover always produces syntactically correct and runnable programs.) Sections 7.4 and 7.5 will describe safe ways to extend GP.

### 7.3.2.2 Evaluation Safety

The purpose of evaluation safety is to ensure evolved programs can run and thereby be assigned fitness even when they run into errors. For example, an evolved expression might divide by 0 or call `MOVE_FORWARD` when facing a wall or precipice. It is common to use *protected* versions of numeric functions that can otherwise throw exceptions, such as division, logarithm, exponential and square root. The protected version of a function first tests for potential problems with its input(s) before executing the corresponding instruction. If a problem is spotted then some default value is returned. Protected division (often notated with `%`) checks to see if its second argument is 0. If so, `%` typically returns the value 1 (regardless of the value of the first argument). (The decision to return the value 1 provides the GP system with a simple way to generate the constant 1, via an expression of the form `(% x 0)`). This combined with a similar mechanism for generating 0 via `(- x x)` ensures that GP can easily construct these two important constants.) Similarly, in a robotic application a `MOVE_AHEAD` instruction can be modified to do nothing if a forward move is illegal or if moving the robot might damage it. (Braitenberg permitted his imaginary robots to make dangerous moves as a way of weeding the poor control program from the better [16].)

An alternative to protected functions is to trap run-time exceptions and strongly reduce the fitness of programs that generate such errors. However, if the likelihood of generating invalid expressions is very high, this can lead to too many individuals

in the population having nearly the same (very poor) fitness. This makes it hard for selection to choose which individuals might make good parents.

### 7.3.2.3 Sufficiency

By sufficiency we mean it is possible to express a solution to the problem using the elements of the primitive set. For example,  $\{\text{AND}, \text{OR}, \text{NOT}, x_1, x_2, \dots, x_N\}$  is a sufficient primitive set for logic problems, since it can produce all Boolean functions of the variables  $x_1, x_2, \dots, x_N$ . The primitive set  $\{+, -, *, /, x, 0, 1, 2\}$  is unable to represent transcendental functions, such as  $\sin(x)$ . When a primitive set is insufficient, GP can often develop programs that approximate the desired solution. Which may be good enough for the user's purpose. Adding a few unnecessary primitives in an attempt to ensure sufficiency tends not to slow down GP overmuch.

### 7.3.3 Step 3: Fitness Function

The task of the fitness measure is to choose which parents are to have offspring. That is, which parts of the search space we have just sampled (which is what the current population has done for us) are worth exploring further. The fitness function is our primary (and often sole) mechanism for giving a high-level statement of requirements to GP.

Fitness can be measured in many ways. For example, in terms of the amount of *error* between its output and the desired output; the amount of *time* (fuel, money, etc.) required; the *accuracy* of the program in recognizing patterns or classifying objects; the *payoff* a game-playing program produces.

Fitness evaluation normally requires executing all the programs in the population, typically multiple times. While one can compile the GP programs that make up the population, the overhead of building a compiler is usually substantial, so it is much more common to use an interpreter to evaluate the evolved programs. Interpreting a program tree means executing the nodes in the tree in an order that guarantees that nodes are not executed before the value of their arguments (if any) is known. This is usually done by traversing the tree recursively starting from the root node and postponing the evaluation of each node until the values of its children (arguments) are known. Other orders, such as going from the leaves to the root, are possible. If none of the primitives have side effects, the two orders are equivalent. Figure 7.7 contains C++ code fragments which implements top down recursive tree evaluation using a linear data structure for speed.

In some problems we are interested in the *output* produced by a program. In other problems we are interested in the actions performed by a program composed of functions with side effects. In either case the fitness of a program typically depends on the results produced by its execution on many different inputs or under a variety of different conditions. For example, the program might be tested on all possible combinations of inputs  $x_1, x_2, \dots, x_N$ . Alternatively, a robot control program might

```

//Flatted tree is stored in array of nodes.
//evalnode has three components. It trades space for speed.
//node (not shown) stores the same information in one byte. It
//is used to store the population. node is expanded to evalnode
//before fitness testing.
//typedef allows code to be compiled for several applications.

typedef float retval;
typedef retval (*EVALFUNC)(); // evaluation code with global pointer
typedef struct evalnode {
    EVALFUNC ef;
    evalnode* jump;
    retval value;
} evalnode; // node type

evalnode* IP;           //global pointer
evalnode* ExprGlobal; //Array holding flatten tree
//some old Sun compilers incorrectly optimise EVAL (workaround via FTP)
#define EVAL ((++IP)->ef)()
#define GETVAL IP->value
#define TRAVERSE() IP=(++IP)->jump

retval D0Eval() { return data[0]; } //data holds inputs D0-D9. Typically it
retval D9Eval() { return data[9]; } //is different for each training case
retval ConstEval() {return GETVAL;}
retval AddEval() {return EVAL + EVAL;}
retval SubEval() {return EVAL - EVAL;}
retval MulEval() {return EVAL * EVAL;}
retval DivEval() { // "Protected" division
    const retval numerator = EVAL;
    const retval denominator = EVAL;
    if (denominator !=0) return numerator/denominator;
    else return 1;
}
retval IflteEval() { //IfLTE(condition1<=condition2,dothis,dothat)
    retval rval=EVAL;
    if (rval<=EVAL) {
        rval=EVAL;
        TRAVERSE(); // Jump the third expression
    } else {
        TRAVERSE(); // Jump the second expression
        rval=EVAL;
    }
    return rval;
}

retval Individual::evalAll() { // eval the whole expression anew
    IP=ExprGlobal-1;           // start at begining of flatten tree
    return EVAL;
}

```

**Fig. 7.7** Example fast interpreter (based on Andy Singleton's GPquick). The tree is linearized and functions and terminals within it are replaced by pointers to C++ functions which implement them. On a typical modern computer the GP individual and the interpreter are held in fast cache. Since the tree is flatten into the traditional depth first order, the interpreter runs from top of the tree to the rightmost terminal in one forward pass. This avoids backtracking. Continuous forward motion suits typical cache architectures.

be tested with the robot in a number of starting locations. These different test cases typically contribute to the fitness value of a program incrementally and for this reason are called *fitness cases*.

Despite all this sophistication and the computational work it does, the fitness function ultimately boils down to just one bit of information: does this mutated program beget another child? We do not know the correct answer to this question. So we add noise (e.g. via tournament selection). Fortunately it is not necessary to get the answer right all the time, or even most of the time, just as long as we are right occasionally. We sometimes lose sight of this hard truth. Sometimes it may be better to accept a less accurate calculation of fitness. If by doing so we reduce the time taken to calculate a fitness value. Thus allowing us to take the life or death decision more times.

### 7.3.4 Step 4: GP Parameters

The most important control parameter is the *population size*. It is impossible to make general recommendations for setting *optimal* parameter values, as these depend too much on the details of the application. However, genetic programming is in practice robust and it is likely that many different parameter values will work. As a consequence, one need not typically spend a long time tuning GP for it to work adequately. Some possible parameter settings are given in the tableau in Table 7.2.

**Table 7.2** Typical parameters for example genetic programming run.

Objective:	Record your problem here
Function set:	For example, $+$ , $-$ , $\%$ (protected division) and $\times$ ; all operating on floats
Terminal set:	For example, $x$ and constants chosen randomly between $-5$ and $+5$
Fitness:	e.g. sum of absolute errors for a number of fitness cases. The number of fitness cases may be limited by the amount of training data available to evaluate the fitness of the evolved individuals. In other cases, e.g. 22-bit even parity [17], there can be too much training data. Then the fitness function may use a fraction of the training data. This does not necessarily have to be done manually as there are a number of algorithms that dynamically change the test set as the GP runs (see [5, Sect. 10.1]).
Selection:	Tournament size 7
Initial pop:	Ramped half-and-half (Section 7.2.2) depth range of 2–6
Parameters:	As a rule one prefers to have the largest population size that your system can handle gracefully. Normally the population size should be at least 500 and people often use much larger populations. (However, some prefer much smaller populations. Typically these rely on mutation rather than crossover and run many more generations.) Traditionally, 90% of children are created by subtree crossover. However, the use of a 50–50 mixture of crossover and a variety of mutations also appears to work well [5, Chapter 5]. Some implementations do not require arbitrary limits of tree size. Even so, because of bloat (the uncontrolled growth of program sizes during GP runs [5, Sect. 11.3]), it is common to impose either a size or a depth limit or both (see Section 7.7.6).
Termination:	10–50 (The most productive search is usually performed in those early generations.)

### 7.3.5 Step 5: When to Stop and How to Decide Who is the Solution

The last step, is choosing when to stop the GP and how to decide which of the thousands of programs that it has evolved to use. Typically we stop either when an acceptable solution has been found or a maximum number of generations has been reached. Typically, the single best-so-far individual is used. Although one might wish to study additional individuals, e.g. to look for particularly short or elegant solutions.

## 7.4 Guiding GP with A Priori Knowledge

As described so far, GP is essentially knowledge free: a powerful search mechanism (evolution) is set free to search the space of all expressions which can be formed from the function and terminal set. Contrast this with traditional methods, such as linear regression, in which a very basic search mechanism is used to search a very restricted set of expressions. Linear regression is often extended to more complex forms (polynomial regression, log regression, etc.), but this still leaves a vast gap between the complete search of GP and the very restricted parameter search of classical regression.

In many applications, the user will know a great deal about the form of acceptable solutions. It can be highly desirable to incorporate this knowledge into the search, since it can save the user time, e.g. by enabling the user to exclude solutions which will not be useful for some reason or to impose a preference ordering on solutions. Including the user's background knowledge can also increase data efficiency and so allow more complex models to be learnt than could possibly be justified solely by the available data. In some cases, such restriction may be essential, because it may not be possible to provide meaningful fitness values for all the solutions a GP system could evolve. Finally, GP search may be more efficient if the user's knowledge can be used to increase the concentration of solutions in the search space. This increase may be non-trivial (in example 3 below it is of many orders of magnitude) but it is nevertheless the least important reason.

In principle, a wide range of mechanisms could be used to restrict the search space; in practice, most available systems use some form of grammar, generally an extension of Context Free Grammars (CFG [18]). The constraints may range over a wide range of complexity, for example

1. Strongly typed systems (Section 7.3.2.1, [19]): in which only type-consistent expressions can be evolved.
2. Extended process models: in many domains, such as ecological modelling [20], there is a known sub-model of processes which are certainly occurring (zooplankton are eating phytoplankton, for example), but there may also be other unknown processes occurring which require adaptation of this process model to fit the data.

3. Dimensional consistency [21]: For example, in physics, equations must be consistent in time ( $t$ ), length ( $l$ ) and mass ( $m$ ) dimensions. For example, integrating Newton's second equation of motion gives  $s = ut + \frac{1}{2}at^2$ .  $s$  has dimensions of length.  $u$  is a velocity and hence has dimension  $l/t$ .  $\frac{1}{2}$  is a pure number and so has no dimensions.  $a$  is an acceleration and hence has dimension  $l/t^2$ . Putting these together, the right-hand side gives  $l/t \times t + lt^{-2} \times t^2 = l$ . Which is indeed the same as the dimensions of the left-hand side ( $l$ ). Dimensionally inconsistent formulae may fit the data well but they are nevertheless unacceptable.

### 7.4.1 Context-Free Grammars in GP

CFG-based GP systems are the most widely used, and the simplest to explain, so we take them as our base case. From the user's perspective, a CFG-GP system is very similar to a standard GP system. However, instead of just providing a list of function and terminal symbols<sup>2</sup>, the user must provide a grammar specifying the ways in which they may be used. That is the only change really required; the user does not have to do anything special with respect to the GP operators (selection, crossover, mutation); the system takes care of those. For some systems, there may be one further difference. In a grammar-based system, the fitness function can be defined in the same way as for standard GP. However, the grammar defines how to build up more complex expressions from simpler ones. In many cases, it is easier to define how to build up the meanings of the expressions (i.e. how to evaluate them) at the same time—we call this “providing a semantics for the grammar”. If this is done, the fitness function definition may reduce to just a few lines of code, defining how these values contribute to the fitness. We give a brief example in Section 7.4.1.1 below.

The grammar provides an additional way for the user to interact with the evolving population. When the CFG-GP system is first run, it may not produce the results the user desires, e.g. the evolved solution may not fit the data sufficiently well. Or it may not be acceptable to the user for some other reason. However, the CFG-GP runs may help the user see how the problem may be solved. Frequently, it is possible to incorporate this insight into the grammar so as to achieve more useful results in subsequent runs. This ability to interact with the solution space, through grammar definitions, is one of the primary practical benefits of grammar-based GP systems.

Of course, the implementation of CFG-GP is a little more complex than simple tree GP, though this is generally not visible to the user. In GP, the individuals of the evolutionary population are expression trees; in CFG-GP, the individuals are parse trees from the grammar, i.e. CFG-GP individuals are paths through the user-supplied

---

<sup>2</sup> A word of caution: GP and grammar terminology were both developed before grammar-based GP systems and use some of the same words. Unfortunately, when they came together in grammar-based GP, some inconsistencies arose. Thus, in a CFG-GP system, a (GP) function symbol is a terminal (in grammar terms), though it is not a member of the GP terminal set. Unfortunately there does not seem to be any reasonable way to resolve this inconsistency.

grammar, starting from the grammar's start symbol (the root of the parse tree). Eventually the path will reach terminals of the grammar (i.e. symbols which cannot be expanded further). The list of terminals (in the order they were encountered) is the output of the grammar. Typically, this list is an executable program (written in the language specified by the user's grammar). It is then run in order to find the fitness of the CFG-GP individual.

Initialization, which requires ensuring grammar consistency while guaranteeing to stay within the depth bound, is also a little complex; most systems use a variant of the grow-tree algorithm described in Section 7.2.2. This is combined with a counting mechanism, to ensure that it is always possible to complete a parse tree within the remaining depth. Crossover and mutation are defined in ways that preserve grammar consistency. Mutation replaces a subtree from the grammar with a random subtree. It creates the random subtree in the same way as the initial population is created, except that it starts from the location in the grammar occupied by the subtree it has just removed, rather than at the root. Crossover is essentially like normal GP crossover, except that crossover is only allowed between nodes with the same grammar non-terminal. This ensures, as with mutation, that the offspring is consistent with the grammar.

#### 7.4.1.1 Example of CFG-GP: Strong Typing in GP

We use Strongly Typed GP [19] as a simple example of grammar use. A GP problem requiring two types, arithmetic and Boolean, might use a grammar such as in Table 7.3. Thus the first “arithmetic” rule says that an arithmetic expression may consist of the sum of two arithmetic expressions, or (| means or) the difference of two arithmetic expressions, and so on. The second “interaction” rule says that a Boolean expression may be formed by comparing two arithmetic expressions, with any of the comparison operators <, = or >. Thus the grammar permits arbitrarily complex nesting of arithmetic and Boolean expressions, but guarantees that they are combined in meaningful ways.

In systems which also support semantic specification within the rules, a rule such as  $A \rightarrow A * A$  would be expanded to include variables. These variables represent the values generated by the rule (such as  $A(A_0) \rightarrow A(A_1) * A(A_2)$ ). Extra (semantic) rules then give the values of those variables (such as  $\text{val}(A_0) = \text{val}(A_1) * \text{val}(A_2)$ ). Of course, in simple cases like this, where the meaning of '\*' is already built into

**Table 7.3** An example grammar for Boolean and arithmetic types. The following six rules define how non-terminal symbols A (the start symbol, representing arithmetic expressions) and B (representing Boolean expressions) can be expanded into 15 (grammar) terminals + - \*/ x 0 if(, , ) < = > & ∨ ¬ true false.

Arithmetic rules	Interaction rules	Boolean rules
$A \rightarrow A + A   A - A   A * A   A / A$	$A \rightarrow \text{if}(B, A, A)$	$B \rightarrow B \& B   B \vee B - B$
$A \rightarrow x   0$	$B \rightarrow A < A   A = A   A > A$	$B \rightarrow \text{true}   \text{false}$

the language in which the GP system is written, the advantage is limited. In more complex domains or problems where other properties of the expression in addition to its value may be needed, semantic specification can greatly simplify coding the problem.

### 7.4.2 Variants of Grammar-Based GP

#### 7.4.2.1 More Powerful Grammars

Perhaps the most important issue is that the user's knowledge may not be expressible in context-free form. This has led to a wide range of extended-grammar systems. They fall into two main classes: Context Sensitive Grammars (CSG) [22] and attribute and other semantic grammars [23].

CSG permit more precise syntactic restrictions on the search space; for some problem domains, this greater expressiveness is important for encoding the problem.

Attribute grammars extend the semantic specification we described in Section 7.4.1.1. In some problems, the semantics may allow us to decide early in the evaluation process, that the individual will have low fitness. For example, in a constraint problem, we may know that if a constraint is breached early in evaluating an individual, the violation is only going to get worse as we continue with its evaluation. Thus semantic constraints may be used to short-circuit fitness evaluation. But even more intriguingly, they may be used to avoid creating poor individuals at all. For example, when we come to cross over individuals, the semantic values attached to the nodes in an individual might indicate that a crossover at a particular point would automatically breach a constraint. A system based on semantic grammars can then simply abort the crossover, never creating the potentially poor individual. In general, if it is difficult to express the user's knowledge about the search space in a CFG, consider using either a CSG or a semantic grammar.

#### 7.4.2.2 More Flexible Representations: GE and Tree Adjunct Grammars

The reduction in search space size provided by a CFG representation can be beneficial for search; but it comes at a cost. The CFG reduces not only the number of formulae that can be represented in the search space, but also the links between them. Paradoxically, in some cases this sparser search space might be *more* difficult for evolution to search than the original search space. Two approaches have been introduced to avoid this problem. In the first [24, 25], the CFG representation is linearized: instead of representing the individuals directly as grammar parse trees, a coding scheme represents them as linear strings. This approach has led to one of the most widely used GP systems, known as Grammatical Evolution (GE) [25]. The other [26] uses an alternative representation from natural language study, Tree Adjunct Grammars (TAGs [27]); unlike CFG trees, any rooted subtree of a TAG tree is

syntactically and semantically meaningful, so that there is much more flexibility in transforming one TAG tree to another.

In both cases, the syntactic flexibility provides an additional benefit: it is relatively easy to implement new operators (often analogous to biological processes that occur in DNA evolution) which may simplify search in particular domains. Practically, this means that where search with standard-GP and CFG-GP systems has stagnated, it may be worth investigating GE or TAGs. They may be able to solve problems which are beyond the reach of more classical GP systems.

#### 7.4.2.3 Grammar Learning

A number of more experimental grammar-based systems [28–30] refine the grammar describing the search space as search proceeds. (These systems are usually based on probabilistic grammars: each grammar rule option has a probability attached to it, indicating the probability that it will be used in generating an individual.) This has two consequences. It can make for faster search. But more importantly, it means that the grammar at the end of the search space may give an explicit representation of the space of solutions (rather than the implicit representation given by the best individuals in a final GP population).

This explicit representation may be of value in its own right, especially in applications such as scientific research, where the desired outcome is better understanding of the processes in the domain, rather than simply predictive models. The use of probabilistic grammars means that the understanding may be quite sensitive, going beyond just the content of the grammar rules. In some parts of the grammar, the probabilities may converge close to either 1.0 or 0.0, indicating that that aspect of the grammar is important in defining a solution to the problem; in others, the probabilities may be more widely spread, indicating that that aspect of the grammar is not particularly important to the problem solution.

## 7.5 Expanding the Search Space in Genetic Programming

In Section 7.4 we described some of the ways in which you can give the evolutionary process a helping hand. For example, by providing domain-specific data types or by constraining programs to conform to an appropriate grammar. But, in the context of a particular problem or a particular set of program representations, we do not necessarily know *how* to give evolution a helping hand. In which case it can be useful to expose more, rather than less, of the system’s decisions about data and control architecture to evolution. Doing so will often incur new costs, some from the added complexity of the system and some from the expansion of the space of programs which GP is searching [31]. However in many cases these costs can be justified by improvements to problem-solving power or scalability.

We will discuss some of the ways in which researchers have expanded the purview of the evolutionary process in GP. In Section 7.5.1, we first examine the evolution of data structures and the ways in which they can be accessed and manipulated by evolving programs. We then turn to program and control structure. Section 7.5.2 describes how GP can be used to evolve programs that use subroutines, macros, and more exotic techniques for controlling the flow of execution. The concept of “development” (here development means the evolved programs build *other* structures which then produce the desired behaviors) provides for even more evolutionary flexibility (Section 7.5.3). The last part of this section (Section 7.5.4) describes mechanisms by means of which the evolutionary processes themselves can be allowed to evolve.

### 7.5.1 Evolving Data Structures and Their Use

The earliest and simplest GP applications evolved programs that used single, simple data types. The use of multiple—but still simple—data types has been helped in a variety of ways, for example by the use of strong typing (see Section 7.4 and [19]). An important technique for evolving programs that use more complex data types is *indexed memory*, which was first presented by Teller [32]. An indexed memory is simply an array of variables of some simple type, accessed using integer indices. Teller showed that by including indexed memory `read` and `write` functions in the GP function one can evolve programs that use memory in relatively complex and useful ways. He also showed that the inclusion of indexed memory was useful in expanding the space of programs over which GP can search, e.g. it can be shown to include programs for all Turing computable functions.

Indexed memory can be used by evolving programs to implement a wide variety of more complex data structures, but modern software engineering practice suggests that it is even more useful for human programmers—and hence possibly also for GP—to have access to higher level data structures. Langdon has investigated the extent to which GP can evolve, and subsequently use, more abstract data structures including stacks, queues and lists [33]. He showed that GP can indeed evolve and subsequently solve problems using such data structures and that GP with abstract data types can outperform GP with indexed memory on several problems, including a context-free language recognition problem and the problem of implementing a simple four function calculator.

Alternative program representations provide additional opportunities for the evolution and use of data structures. For example, approaches based on polymorphic functional representations, initially developed by Yu using Haskell [34], have recently been extended by Binard and Felty, using a version of the  $\lambda$ -calculus to which they have added an operation of abstraction on types [35]. They showed how their system could evolve and use abstractions for Boolean and list data types which were not explicitly present in their initial environments.

To some extent, the ways in which data types and program syntax are interrelated determine the ways in which GP can discover and use complex data structures. Strongly typed GP and polymorphic GP provide two approaches but they do not exhaust the possibilities. For example, in the Push programming language all communication between instructions is accomplished via typed global data stacks. It is not specified by placing the instructions next to each other, as in most programming languages. This decouples an evolving program’s type structures from its control structures and thereby permits greater flexibility (for good or ill) in the expression of programs that manipulate multiple data types [36, 37].

### 7.5.2 *Evolving Program and Control Structure*

Most interesting programs that are written by humans involve the use of control structures not available in the simplest GP systems. These include mechanisms that support iteration, recursion, and the definition and use of reusable code modules. As with data structures, GP researchers have developed a range of techniques for evolving programs that evolve and use these powerful control abstractions.

Limited forms of iteration are relatively easy to handle through the use of primitive functions that simply repeat the execution of a subexpression some specified number of times. This was demonstrated in Koza’s first book using `do_until` structures [2]. A variety of more sophisticated techniques, such as the “restricted iteration creation” operations of Koza and Andre [38], have been developed to help GP systems incorporate iteration into evolving programs. Both iteration and recursion present challenges with respect to nontermination; this is generally handled either by imposing execution limits or by using primitives that are naturally self-limiting, such as the `foldr` function in Haskell [39]. While the search space of recursive programs appears to be rugged, and several early attempts to evolve recursive programs produced negative results (e.g. [40]), more recent research has been increasingly successful (e.g. [37, 39, 41–44]).

Modular structures can also be incorporated into evolving programs in several ways. A common approach, pioneered by Koza [45], is to simultaneously evolve a “main program” (sometimes called a “result producing branch”) and one or more “automatically defined function” (ADF) branches that can be called by the main program and possibly by each other (usually with restrictions to prevent nonterminating recursion). This approach has been shown to provide dramatic advantages in certain problem areas with exploitable regularities. In the original ADF framework the number of ADFs and the numbers of arguments that they take are specified manually, but the subsequent development of “architecture altering operations” brought these decisions, as well, under evolutionary control [46].

A variety of other approaches to the evolution and use of modules have also been developed. For example, in “evolutionary module acquisition” the code for modules is not evolved in separate branches but rather is extracted from the main programs of relatively successful individuals in the population [47–49]. “Automatically defined

macros” allow GP to evolve not only function modules but also control structure modules that execute code conditionally or repeatedly [50]. And several researchers have shown how GP can be used to evolve object-oriented programs in which functionality is modularized through the use of classes and objects [51, 52].

More radical forms of control structure evolution have also been explored. For example, the inclusion of combinators (higher order functions studied in the theory of functional programming languages) in the function set can allow GP to explore a large space of control architectures while imposing minimal constraints on program syntax [37, 53]. Perhaps the greatest flexibility—and therefore potentially the most intractable search space—is provided by the Push programming language, in which programs can contain arbitrary code-manipulation instructions and thereby transform their own code in arbitrary ways during execution [36, 37]. All of these innovations have been demonstrated to be useful in certain circumstances, but further study is required to determine exactly when.

### 7.5.3 Evolving Development

In nature an organism’s genes do not interact directly with its environment; rather, they direct the construction of proteins which form the organism’s body. It is that body—the phenotype—that interacts with the organism’s environment.

Several GP techniques have been inspired by the biological distinction between genotype and phenotype and by the process, called ontogeny, by which the genotype leads to the phenotype (e.g. [54–57]). In the most common approach, *developmental GP*, the programs produced by GP are structure-building programs, and it is the structures that are built by these programs, rather than the programs themselves, that are tested for fitness in the problem environment.

Typically one begins the developmental process with an “embryo” that consists of a minimal structure of the appropriate kind. The functions in the GP function set then, when executed, augment this embryo. For example, if the desired structure is a neural network then the embryo might consist of a single input node connected to a single output node and the functions in the GP function set might add additional nodes and connections [58]. Or if the desired structure is an electrical circuit then the embryo might consist of a voltage source connected to a load resistance and the functions in the GP function set might add components and wires [46].

The developmental approach has been successful in a wide range of application areas, ranging from the evolution of control systems [59] to the evolution of quantum circuits [60]. Part of its appeal comes from the way that it facilitates the application of GP to the evolution of structures that are not themselves best viewed as computer programs; developmental GP still evolves computer programs, but the programs build (develop) structures that might be quite different in nature from computer programs. Other attractions of developmental GP may derive from ways in which it affects the GP search process. For example, one might expect mutation to have a different range of effects when applied early in a developmental process than when applied to the fully developed phenotype (as is done in standard GP).

Whether this will be the case, and whether a developmental approach will therefore help or hinder, will depend on the specific problem and program representations. In biology, however, evolution often proceeds through adjustments to developmental programs and timing [61], and there is recent evidence that it can also lead to desirable properties such as robustness and self-repair in GP [62].

When the phenotype is not a computer program, developmental GP makes it easier to apply GP by making it easier to choose the function set. Because of the freedom that one has in designing a structure-building function set, developmental GP also allows you to experiment with different genotype-to-phenotype mappings, some of which may be more successful than others.

### 7.5.4 Evolving Evolutionary Mechanisms

The most radical expansions of the GP search space involve evolutionary control of the evolutionary process itself. There is a long history of research on self-adaptive mechanisms in evolutionary computation (e.g. see [63, 64]). In most areas outside of GP this means that numerical parameters of the evolutionary algorithm—for example mutation rates—are themselves encoded in the evolving genomes and are thereby subject to variation and selection. Similar strategies can also be applied to GP, but because GP involves the evolution of programs it is natural to ask whether a GP process can also usefully evolve its own utility programs—for example its utility program for performing mutation—and other aspects of the overall evolutionary algorithm along with the main problem-solving programs that are its primary targets.

Several approaches to self-adaptation in GP have been explored. These include several “meta-GP” approaches, in which programs implementing genetic operators (like mutation and crossover) co-evolve with problem-solving programs in separate populations [11, 65, 66]. In “autoconstructive evolution” these evolving auxiliary functions are encoded in the problem-solving programs themselves; much as in biology. Code for reproduction (mate selection, mutation, recombination, etc.) can be intermingled with, and can interact with, code for survival (problem-solving performance) in an individual’s genome [36].

The attractions of these techniques, which allow a GP system to evolve itself as it runs, stem from the possibility that the resulting systems will be adapted to their problem environments and therefore more effective than hand-designed systems. As with the other expansions to the GP search space discussed above, however, there are significant associated costs and many open research questions about how and when these costs can be overcome or justified.

## 7.6 Applications

There are more than 5000 recorded uses of GP. These include an enormous number of applications. It is impossible to list them all. However we shall start with a discussion of the general kinds of problems where GP has proved successful (Section 7.6.1) and the important area of symbolic regression (Section 7.6.2). Next

come sections which review the main application areas of GP: Image and Signal processing (Section 7.6.3) Finance (Section 7.6.4) Industrial Process Control (Section 7.6.5) Medicine and Bioinformatics (Section 7.6.6) Hyper-heuristics (Section 7.6.7) Entertainment and Computer Games (Section 7.6.8) and Art (Section 7.6.9). We conclude with a description of some of the human-competitive results automatically generated by GP (Section 7.6.10).

### 7.6.1 Where GP Has Done Well

If one or more of the following apply, GP may be suitable.

- The interrelationships among the relevant variables are unknown or poorly understood. GP can help discover which variables and operations are important; provide novel solutions to individual problems; unveil unexpected relationships among variables; and, sometimes GP can discover new concepts. These might then be taken and applied as in a conventional way.
- Finding the size and shape of the ultimate solution is a major part of the problem.
- Many training data are available in computer-readable form.
- There are good simulators to test the performance of tentative solutions to a problem, but poor methods to directly obtain good solutions.

In many areas there are tools to evaluate a completed design, (e.g. how far will this bridge bend under the forecast load.) Such tools solve the *direct problem* of working out the behaviour of a solution. However, the knowledge held within them cannot be easily used to solve the *inverse problem* of designing an artefact from its requirements. GP can exploit simulators and analysis tools and “data-mine” them to solve the *inverse problem* automatically.

- Conventional mathematical analysis cannot give analytic solutions.
- An approximate solution is acceptable.
- Small improvements are highly prized. Even in mature applications GP can sometimes discover small delta improvements, which may be very valuable.

Two examples are NASA’s work on satellite radio aerial design [67] and Spector’s evolution of new quantum computing algorithms that out-performed all previous approaches [68, 69]. Both of these domains are complex, do not have analytic solutions, but good simulators existed which were used to define the fitness of evolved solutions. In other words, people did not know how to solve the problems but they could (automatically) recognize a good solution when they saw one. In both cases GP discovered highly successful and unexpected designs. Also the key component of the evolved quantum algorithm was extracted and applied elsewhere [70].

### 7.6.2 Curve Fitting, Data Modelling and Symbolic Regression

There are many very good tools which will fit curves to data; however, typically they require you to specify the type of curve you want to fit, e.g. a straight line, an

exponential, a Gaussian distribution. Where GP can help is where the form of the curve or underlying model is unknown. In fact the main problem can be discovering the form of the solution or which data to use. This is generally known as *symbolic regression*.

By regression we mean finding the coefficients (e.g. slope and  $y$ -intercept) of a predefined function such that the function best fits some data. However, until a good fit is found the experimenter has to keep trying different functions by hand until a good model for the data is found. Sometimes, even expert users have strong biases when choosing functions to fit. For example, in many applications there is a tradition of using linear models, even when the data might be better fit by a more complex model. Since GP does not make this assumption, it is well suited to this sort of discovery task.

For instance, GP can evolve *soft sensors* [71]. The idea is to evolve a function which estimates what a real sensor would measure, based on data from other actual sensors in the system, (e.g. where placing an actual sensor would be expensive.) Experimental data (e.g. from industrial plant) typically come in large tables where numerous quantities are reported. Usually we know which variable we want to predict (e.g. the soft sensor value) and which other quantities we can use to make the prediction (e.g. the real sensor values). If this is not known, then experimenters must decide which are going to be their *dependent variables* before applying GP. Sometimes there are hundreds or even thousands of variables. (In bioinformatics the number of variables may approach a million.) It is well known that in these cases the efficiency and effectiveness of any machine learning or program induction method, including GP, can dramatically drop as most of the variables are typically redundant or irrelevant. This forces the system to waste considerable energy on isolating the key features. To avoid this, it is necessary to perform some form of *feature selection*, i.e. we need to decide which *independent variables* to keep and which to leave out. There are many techniques to do this, its even possible that GP itself can be used to do feature selection [72].

There are problems where more than one output (prediction) is required. For example, Table 7.4 contains data collected from a robot. The left-hand side gives four control variables, while the right-hand side contains six dependent variables measured after the robot obeyed the commands. The *Elvis* robot is shown in Figure 7.8 during the acquisition of a data sample. The roles of the independent and dependent variables are swapped when GP is given the task of controlling the arm given data from the robot's eyes.

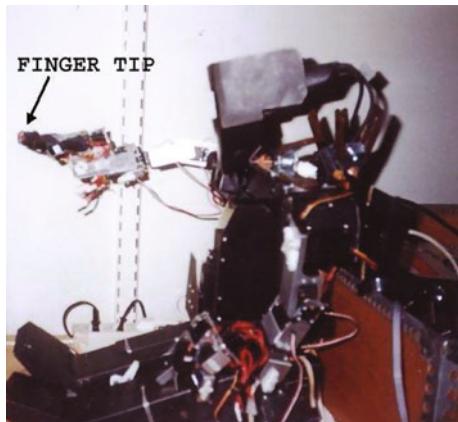
There are several GP techniques which might be used to deal with applications where multiple outputs are required, e.g. GP individuals made of multiple trees, linear GP with multiple output registers, graph-based GP with multiple output nodes, and a single GP tree with primitives operating on vectors.

After a suitable data set has been assembled, the GP terminal set (cf. Section 7.3.1) must be defined. Since the independent variables will become the evolved code's inputs, they must be included in the terminal set. Typically some constants are also included. Next is the function set (cf. Section 7.3.2). It is often sufficient to give GP the standard four arithmetical operations ( $+ - \times \%$ ) and an *if*. The

Arm actuator	Left eye			Right eye		
	x	y	size	x	y	size
-376 -626 1000 -360	44	10	29	-9	12	25
-372 -622 1000 -380	43	7	29	-9	12	29
-377 -627 899 -359	43	9	33	-20	14	26
-385 -635 799 -319	38	16	27	-17	22	30
-393 -643 699 -279	36	24	26	-21	25	20
-401 -651 599 -239	32	32	25	-26	28	18
-409 -659 500 -200	32	35	24	-27	31	19
-417 -667 399 -159	31	41	17	-28	36	13
-425 -675 299 -119	30	45	25	-27	39	8
:	:	:	:	:	:	:

*continues for a total of 691 lines*

**Table 7.4** Samples showing the size and location of Elvis's finger tip as apparent to his two eyes, given various right-arm actuator set points. GP inverts the mapping and evolves 4 functions which take data collected by both cameras (which show a target) and output instructions to the four arm motors so that his arm moves to the target.



**Fig. 7.8** Elvis robot sitting with his right hand outstretched. The apparent position and size of a bright red laser attached to his finger tip is recorded. The data are then used to train a GP to move the robot's arm to a spot in three dimensions using only his eyes.

terminal and function sets are the raw components from which GP tries to build its solutions.

In virtually all symbolic regression applications the fitness function (cf. Section 7.3.3) must measure how close the outputs produced by each program are to the values of the dependent variables, when the corresponding values of the independent ones are used as inputs for the program. So, symbolic regression fitness functions tend to include summing the errors measured for each record in the data set. Usually either the absolute difference or the square of the error is used.

### 7.6.3 Image and Signal Processing

Ford were among the first to consider using GP for industrial signal processing [66]. They evolved algorithms for pre-processing electronic motor vehicle signals for possible use in engine monitoring and control.

Several applications of GP for image processing have been for military uses. For example, QinetiQ evolved programs to pick out ships using SAR radar from space satellites and to locate ground vehicles from airborne photo reconnaissance. They also used GP to process surveillance data for civilian purposes, such as predicting motorway traffic jams from subsurface traffic speed measurements [67]. Satellite images can also be used for environmental studies and for prospecting for valuable minerals [24].

Zhang has been particularly active at evolving programs with GP to visually classify objects (such as human faces) [76].

To some extent, extracting text from images (OCR) can be done fairly reliably and the accuracy rate on well-formed letters and digits is close to 100%. However, many interesting cases remain [77] such as Arabic [78] and oriental languages, handwriting [79–81] (such as the MNIST examples of handwritten digits from IRS tax returns) and musical scores [82].

The scope for applications of GP to image and signal processing is almost unbounded. A promising area is medical imaging. GP image techniques can also be used with sonar signals [83]. Off-line work on images includes security and verification. For example, [84] have used GP to detect image watermarks which have been tampered with.

### 7.6.4 Financial Trading, Time Series Prediction and Economic Modelling

GP is very widely used in these areas. It is impossible to describe all its applications instead we will just hint at a few. Chen has written more than 60 papers on using GP in finance and economics. He has investigated modelling of agents in stock markets [85], game theory, evolving trading rules for the S&P 500 [86] and forecasting the Hong Kong Hang-Seng index.

The *efficient markets hypothesis* is a tenet of economics. It is founded on the idea that everyone in a market has “perfect information” and acts “rationally”. If the efficient markets hypothesis held, then everyone would see the same value for items in the market and so agree the same price. Without price differentials, there would be no money to be made from the market itself. Whether it is trading potatoes in northern France or dollars for yen, it is clear that traders are not all equal and considerable doubt has been cast on the efficient markets hypothesis. So, people continue to play the stock market. Game theory has been a standard tool used by economists to try to understand markets but is often supplemented by simulations

with both human and computerized agents. GP is increasingly being used as part of these simulations of social systems.

The US Federal Reserve Bank used GP to study intra-day technical trading on the foreign exchange markets to suggest the market is “efficient” and found no evidence of excess returns [87]. This negative result was criticized in [88]. Later work by Neely et al. suggested that data after 1995 are consistent with Lo’s *adaptive markets hypothesis* rather than the *efficient markets hypothesis* [89]. GP and computer tools are being used in a novel data-driven approach to try and resolve issues which were previously a matter of dogma.

From a more pragmatic viewpoint, Kaboudan shows GP can forecast international currency exchange rates [90], stocks and stock returns, house prices and consumption of natural gas. Tsang and his co-workers continue to apply GP to a variety of financial arenas, including betting [91], forecasting stock prices, studying markets, approximating Nash equilibrium in game theory and arbitrage. Dempster and HSBC also use GP in foreign exchange trading [92]. Pillay has used GP in social studies and teaching aids in education, e.g. [93].

### 7.6.5 Industrial Process Control

Kordon and his coworkers in Dow Chemical have been very active in applying GP to industrial process control. In [94] Kordon describes where industrial GP stands now and how it will progress. Another active collaboration is that of Kovacic and Balic, who used GP in the computer numerical control of industrial milling and cutting machinery [95]. The partnership of Deschaine and Francone is most famous for their use of Discipulus for detecting bomb fragments and unexploded ordinance [96]. Genetic programming has also been used in the food processing industry. For example, Barriere et al. modelled the ripening of camembert [97].

Lewin, Dassau and Grosman applied GP to the control of an integrated circuit fabrication plant [98]. GP has also been used to identify the state of a plant to be controlled (in order to decide which of various alternative control laws to apply). For example, Fleming’s group in Sheffield used multi-objective GP [99] to reduce the cost of running aircraft jet engines.

### 7.6.6 Medicine, Biology and Bioinformatics

Kell and his colleagues in Aberystwyth have had great success in applying GP widely in bioinformatics [100]. Another very active medical research group is that of Moore and his colleagues at Vanderbilt [101]. Many medical data sets are very wide. Some have many thousands of inputs, but relatively few cases. (For example, a typical GeneChip data set will have tens of thousands of measurements per patient but may cover less than a hundred people [72]). Such wide data sets tend to

be avoided by traditional statistical techniques, where often the first reaction is to try and remove as many attributes as possible. Discarding whole columns of training data is often called “feature selection”. However, as has been repeatedly shown, e.g. by the Aberystwyth and Vanderbilt groups, GP can sometimes be successfully applied directly to very wide data sets.

Computational chemistry is widely used in the drug industry. Some properties of simple molecules can be calculated. However, the interactions between chemicals which might be used as drugs and medicinal targets within the body are beyond exact calculation. Therefore, there is great interest in the pharmaceutical industry in approximate in silico models which attempt to predict either favourable or adverse interactions between proto-drugs and biochemical molecules. Since these are computational models, they can be applied very cheaply in advance of the manufacturing of chemicals, to decide which of the myriad of chemicals might be worth for further study. Potentially, such models can make a huge impact both in terms of money and time without being anywhere near 100% correct. Machine learning and GP have both been tried. GP approaches include [102, 103].

### ***7.6.7 GP to Create Searchers and Solvers—Hyper-heuristics***

A heuristic can be considered to be a rule-of-thumb or “educated guess” that reduces the search required to find a solution. A meta-heuristic (such as a genetic algorithm) is a non-problem specific heuristic, i.e. a rule-of-thumb which can be tried on a range of problems. A hyper-heuristic is a heuristic to choose other heuristics. The difference between meta-heuristics and hyper-heuristics is that the meta-heuristic operates directly on the problem search space with the goal of finding optimal or near-optimal solutions. Hyper-heuristic operate on the heuristics search space (which consists of the heuristics used to solve the target problem). Their aim is to find good heuristics for a problem, for a certain class of instances of a problem or even for a particular instance of the problem.

GP has been very successfully used as a hyperheuristic. For example, GP has evolved competitive SAT solvers [104], state-of-the-art bin packing algorithms, particle swarm optimizers, evolutionary algorithms and travelling salesman problem solvers [105].

### ***7.6.8 Entertainment and Computer Games***

Today, a major usage of computers is interactive games. There has been some work on incorporating artificial intelligence into mainstream commercial games. Naturally the software owners are not keen on explaining exactly how much AI the games contain or giving away sensitive information on how they use AI. However, published work on GP and games includes Othello, Poker, Backgammon [106],

robotics, including robotic football, Corewares, Ms Pac-Man, radio-controlled model car racing, Draughts, and Chess. Funes [107] reports experiments which attracted thousands of people via the Internet who were entertained by evolved Tron players.

### 7.6.9 The Arts

Computers have long been used to create purely aesthetic artefacts. Much of today's computer art tends to ape traditional drawing and painting, producing static pictures on a computer monitor. However, the immediate advantage of the computer screen — movement — can also be exploited. In both cases evolutionary computation can, and has been, exploited. Indeed, with evolution's capacity for unlimited variation, evolutionary computation offers the artist the scope to produce ever changing works. The use of GP in computer art can be traced back at least to the work of Karl Sims and William Latham. Christian Jacob's work provides many examples. Many recent techniques are described in [108].

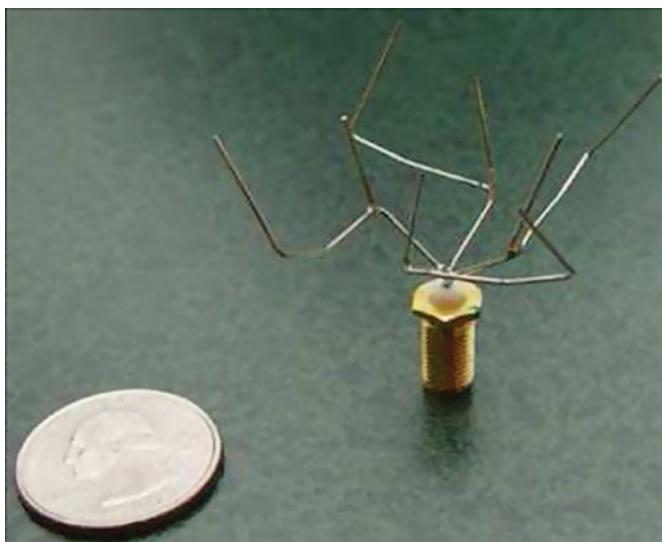
Evolutionary music has been dominated by Jazz [109], which is not to everyone's taste. Most approaches to evolving music have made at least some use of interactive evolution [110] in which the fitness of programs is provided by users, often via the Internet. The limitation is almost always finding enough people willing to participate [111]. It is surprising given their monetary value that so far little use has been made of GP to generate novel cell phone ring tones.

One of the sorrows of AI is that as soon as it works it stops being AI and becomes computer engineering. For example, the use of computer-generated images has recently become cost-effective and is widely used in Hollywood. One of the standard state-of-the-art techniques is the use of Reynold's swarming "boids" [112] to create animations of large numbers of rapidly moving animals. This was first used in *Cliffhanger* (1993) to animate a cloud of bats. Its use is now commonplace (herds of wildebeest, schooling fish, and even large crowds of people). In 1997 Craig was awarded an Oscar.

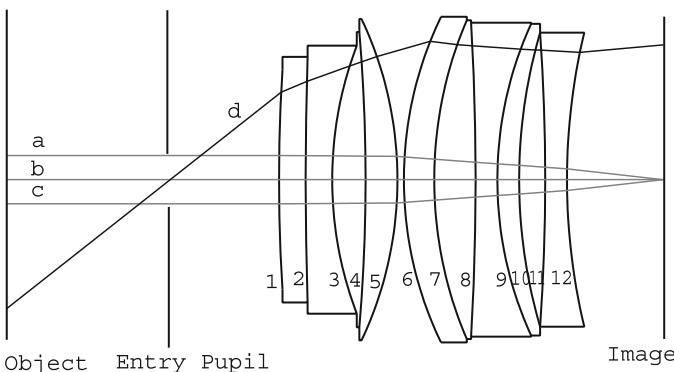
### 7.6.10 Human Competitive Results: The Humies

A particularly informative measure of the power of a problem-solving technology is its track record in solving problems that could only be solved previously by means of human intelligence and ingenuity. In order to highlight such achievements by genetic and evolutionary computation an annual competition has been held since 2004 at the Genetic and Evolutionary Computation Conference (GECCO), organized by the Association for Computing Machinery's Special Interest Group on Genetic and Evolutionary Computation (ACM SIGEVO). This competition, known as the "Humies", awards substantial cash prizes to results deemed "human competitive" as assessed by objective criteria such as patents and publications [113].

25 gold, silver and bronze “medals” with cash prizes have been awarded in the Humies competition, totaling \$45,700. Of these, 13 of the medals (4 gold, 7 silver, 2 bronze) have been awarded to teams using GP (as opposed to other genetic and evolutionary computation methods), in application areas including antenna design, quantum circuit design, mechanical engineering, optical system design, game strategy design, computer vision, and pure mathematics. Figure 7.9 shows a gold medal winning result from 2004, an antenna that was designed using GP for NASA’s Space Technology 5 mission [67]. Figure 7.10 shows a silver medal winning result from 2005, a lens system that duplicates the functionality of the patented Nagler lens system but with a novel topology [114].



**Fig. 7.9** Award winning human-competitive antenna design produced by GP.



**Fig. 7.10** Award winning human-competitive lens designed by GP [114, Figure 12].

## 7.7 Trouble-Shooting GP

The evolutionary dynamics are often very complex so it often difficult to troubleshoot evolutionary computation system. On the other hand, they can be very resilient and even GP systems with horrendous bugs can evolve valid solutions. Nonetheless, we suggest some general issues to keep in mind. To a large extent the advice in [2, 115] and [33, Chapter 9] also remains sound.

### 7.7.1 *Can You Trust Your Results?*

Here we have an interesting divergence between universities and life. Are your results about your GP system? (e.g. is it better than a GP without mutation?) Or about your application? (e.g. have you evolved a better image filter?) [5] describes how to overcome the stochastic nature of evolution and what it means to be better.

In complex applications with powerful techniques, like GP, the danger of learning the training data and so creating a solution which fits it too faithfully is ever present [116]. The classic example is where a neural network was asked to find tanks. It was presented with pictures of fields containing tanks and the same fields without tanks. After prolonged training, it could differentiate between the two. However, the final system failed to find tanks. Eventually the problem was traced to the training images. Since tanks are heavy all the pictures with tanks in them were taken on one day, the tanks were moved, and some time later another set of pictures were taken. The ANN had cheated, it had learnt (using brightness) to distinguish pictures taken early in the day from those taken later and totally ignored the presence or absence of the tanks. While it humorous and is easy to see after the fact, you must ensure the machine learning does not play this joke on you.

### 7.7.2 *Study Your Populations*

If you are not getting your desired results, take the time to dig around in the populations and see what is actually being evolved. For example, if you included a particular input or function, is it included in the better individuals? Are they using it in sensible ways? (Sometimes the tree may include an input but it has no or little impact on the program's behaviour, e.g., because it is multiplied by zero.) Is the primitive being multiply used? Similarly, if you are using grammatical evolution, are your evolved individuals using your grammar as you expected? Or is the grammar biasing the system in an undesirable or an unexpected way?

Remember GP is doing genetic search. GP increases the numbers of genes (i.e. terminals and functions) which appear in above average fitness individuals. You might keep a count of the number of times important primitives occur in the population. Mostly gene numbers vary randomly according to how lucky they are. However, look out for primitives that become extinct or (if using mutation) reduce to low

background levels. If this happens, it suggests that the fitness function is driving the current GP population in an unintended direction.

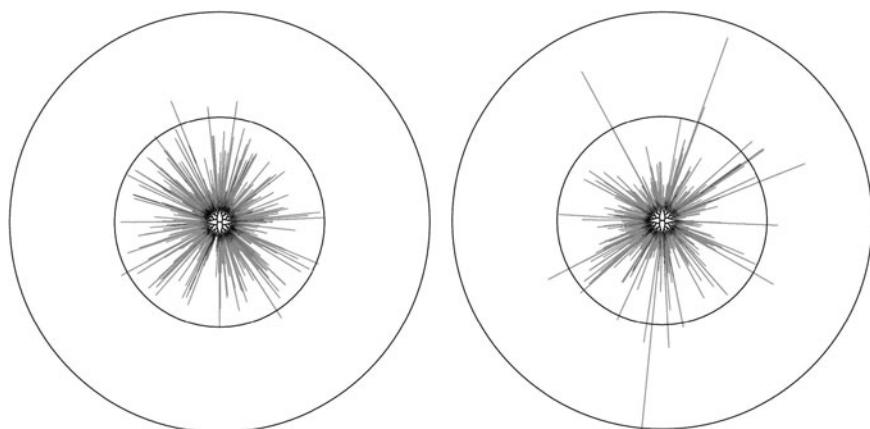
Is the distribution of fitness values of members of the population (particularly that of the better programs) dominated by a few values with large gaps between them? This suggests jumping these gaps may be hard. It also suggests that the next improvement may also be separated from the current best. So finding the next improved solution will require jumping a large gap and so be difficult. Perhaps changing your fitness function to be more continuous will improve performance considerably.

### 7.7.3 Studying Your Programs

A major advantage of GP is you create *visible* programs. You can see how they work. You can explain how they work to your customers. When presenting GP results, include a slide of the evolved program. The `dot` package (<http://www.graphviz.org/>) is good a starting point for nicely presented graphs. GP trees can be automatically converted to dot (<http://www.cs.ucl.ac.uk/staff/W.Langdon/lisp2dot.html>).

There are methods to automatically simplify expressions (e.g. in Mathematica and Emacs). However, simply things like removing excess significant digits and combining constant terms can make your solution more intelligible. After cleaning up the answer, make sure it still works.

In some cases the details of the trees (e.g. the particular nodes) are less important than the general size and shape. In [117], Daida describes a way to visualize the size and shape of either individual trees or an entire population, cf. Figure 7.11. (A Mathematica implementation is available via <http://library.wolfram.com/infocenter/MathSource/5163/>.)



**Fig. 7.11** The size and shape of 1,000 individuals in the final generation of runs using a depth limit of 50 (on the left) and a size limit of 600 (on the right). The inner circle is at depth 50, and the outer circle is at depth 100. These plots are from [119].

### 7.7.4 Encourage Diversity

If GP is to benefit from using a population: the population must be diverse. (Otherwise it might be much more efficient to use a hill climbing or other single point search like simulated annealing.) Sections 7.7.2 and 7.7.3 have described measuring its diversity in terms of its genes (i.e. functions and terminals) and the size and shape of the programs. You can also consider the variation in the programs' behaviour (cf., for Boolean problems, [118]). Studying behaviour, as opposed to genetic makeup, avoids the problem that GP populations often bloat [5, Sect. 11.3]. In bloated populations syntactically (i.e. genetically) evolved programs appear different but the difference is in unused code. However, studying the programs' behaviours will show if the population's phenotypes have converged excessively.

Since mutation and crossover often produce diverse but low fitness individuals you could restrict studies of population diversity to just the subset of the population which is selected to have children.

If you suspect the population has converged excessively you could

- Not use the reproduction operator.
- Add one or more mutation operators.
- Use a weaker selection mechanism, e.g., to reduce the tournament size.
- If you are using the “steady state” approach, i.e. you add new programs to the population immediately, rather than waiting until a whole new population has been created. You could choose who to overwrite at random. (Often people kill the worse member of the population and replace him with the new child. This is fine but tends to increase the convergence of the population.) You might want to protect the best member of the population to ensure he is not deleted.
- Use a generational population model instead of a steady-state model.
- The standard population is panmictic. This means there are no restrictions on which individual mates with and favourable genetic innovations rapidly spread through and may take over the whole population. In contrast a large population may be split into semi-isolated demes [5, Sect. 10.5] which keeps diversity high by slowing the spread of improvements [33, 120].

Demes are often used in conjunction with parallel hardware. The speed at which innovations spread is controlled both by the number of emigrants and how the demes are interconnected. Both all-to-all and toroidal topologies are common. They have very short paths between demes. Arranging demes in a ring gives a longer convergence time. Typically many individuals (say 2%) are transferred between demes for each generation. Over biological timescales, such a high immigration rate is sufficient to prevent a converged population diverging into separate species. However, GP is typically not run for so many generations and, from an engineering standpoint, one has to trade off rapid take up of good solutions versus searching different locations.

- Use fitness sharing or even multi-objective approaches to encourage the formation of many fitness niches.

### 7.7.5 Approximate Solutions Are Better than No Solution

When GP starts, typically, it starts from nowhere, i.e. well behind the state of the art. For the initial population to evolve, it must contain some programs with an “edge”. With some advantage, even if slight, over the rest of the population. You must design your selection and fitness function to amplify this. A typical fitness function gives a continuous measure of how far a program is from your requirements. The final program will be a descendent of those you select at the beginning so the fitness function must not only prefer better than random but also reward approximate solution to the whole problem which may be refined into a complete solution. Or rather, an approximate solution to the whole problem.

This is true throughout the run. At every generation the fitness function must seek out approximate solutions from which better ones can be evolved. Even standing still may not be enough. Nature tends to the easy thing. In GP this often corresponds to finding new programs which have the same fitness as their parents. Unfortunately within a few generations, they can evolve to become very resistant to change and further progress to your goal (as opposed to theirs) becomes very hard.

Consider (just for illustrative purposes) a problem with just five test cases, four of which are fairly easy and consequently less important, with the fifth being crucial and quite difficult. So the population may contain individuals that can do the four easier tasks, but are unable to make the jump to the fifth. There are several things you could try: (1) weight the hard task more heavily or (2) use a multi-objective approach. However, a more fundamental and probably more successful approach is to redesign how you sort the programs, e.g. (3) divide the task up in some way into sub-tasks, (4) provide more tasks, or (5) change it from being a binary condition (meaning that an individual does or does not succeed on the fifth task) to a continuous condition, so that an individual GP program can partially succeed on the fifth task to a greater or less extent. The idea is to create a smoother gradient for the evolutionary process to follow.

### 7.7.6 Control Bloat

If you are running out of memory or your execution times seem inordinately long, look the size of your evolved expressions. Often they will be growing over time. It is usually necessary to provide some form of bloat control, cf. [5, Sect. 11.3]. Controlling bloat is also important if one’s goal is to find a comprehensible model, since in practice these must be small. A large model will not only be difficult to understand but also may over-fit the training data [121].

### 7.7.7 Convince Your Customers

For your work to make an impact it must be presented in a form that can convince others of the validity of its results and conclusions. This might include a pitch within

a corporation seeking continued financial support for a project, the submission of a research paper to a journal or the presentation of a GP-based product to potential customers. [5] contains suggestions on improving written and verbal presentation of artificial evolution experiments. While [122, e.g., Chapter 14] has many suggestions about getting your work accepted (and paid for) by your customers.

The burden of proof is on the users of GP. It is important to use the customer’s language. If the fact that GP discovered a particular chemical is important in a reaction or drug design, you should make this stand out during the presentation. A great advantage of GP over many AI techniques in that its results are often simple equations. Ensure these are intelligible to your customer, e.g., by simplification. Also make an effort to present your results using your customer’s terminology. Your GP system may produce answers as trees, but if the customers use spreadsheets, consider translating the tree into a spreadsheet formula.

Also, one should try to discover how the customers intend to validate GP’s answer. Do not let them invent some totally new data which have nothing to do with the data they supplied for training (“just to see how well it does...”). Avoid customers with contrived data. GP is not omnipotent, it knows nothing about things it has not seen. At the same time you should be scrupulous about your own use of holdout data. GP is a very powerful machine learning technique. With this comes the ever present danger of over-fitting. One should never allow performance on data reserved for validation to be used to choose which answer to present to the customer.

## 7.8 Conclusions

We have seen how genetic programming works and how to use it. We have hinted at just two of the many exciting research areas. The first widens the application of GP by using formal grammars to capture user knowledge and so guide GP. The second (so far) complementary approach extends GP by doing the opposite! That is, sometimes GP can benefit from having additional freedom, including the freedom to evolve parts of itself. We have skimmed through a few of GP’s many applications, including cases where GP has evolved human competitive solutions. Finally we have tried to distill some practical “how to” knowledge into a few pages.

To conclude perhaps the best introduction to genetic programming is to create your own (or borrow someone else’s) and evolve things.

## References

1. Turing, A.M.: Intelligent machinery. Report for National Physical Laboratory. Reprinted in Ince, D.C. (ed.) (1992) Mechanical Intelligence: Collected Works of A.M. Turing. (1948)
2. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press (1992)
3. Darwin, C.: The Origin of Species, 1859 edn. John Murray, Penguin classics (1859)

4. Banzhaf, W., Nordin, P., Keller, R.E., Francone, F.D.: *Genetic Programming – An Introduction; On the Automatic Evolution of Computer Programs and its Applications*. Morgan Kaufmann, San Francisco, CA, USA (1998)
5. Poli, R., Langdon, W.B., McPhee, N.F.: 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 Koza, J.R.)
6. Darwin, C.: *Voyage of the Beagle*. Henry Colburn, London, penguin classics, 1989 edition, (1839)
7. Nordin, P., Banzhaf, W., Francone, F.D.: Efficient evolution of machine code for CISC architectures using instruction blocks and homologous crossover. In: Spector, L., Langdon, W.B., O'Reilly, U.-M., Angeline, P.J. (eds.) *Advances in Genetic Programming 3*, chapter 12, pp. 275–299. MIT Press, Cambridge, MA, USA (1999)
8. Foster, J.A.: Review: Discipulus: a commercial genetic programming system. *GP&EM*, **2**(2), 201–203 (2001)
9. Brameier, M., Banzhaf, W.: A comparison of linear genetic programming and neural networks in medical data mining. *IEEE Trans. EC*, **5**(1), 17–26 (2001)
10. Poli, R.: Discovery of symbolic, neuro-symbolic and neural networks with parallel distributed genetic programming. Technical Report CSRP-96-14, University of Birmingham, School of Computer Science, August 1996. Presented at 3rd International Conference on Artificial Neural Networks and Genetic Algorithms, ICANNGA'97
11. Teller, A.: Evolving programmers: The co-evolution of intelligent recombination operators. In: Angeline, P.J., Kinnear, Jr., K.E. (eds.) *Advances in Genetic Programming 2*, chapter 3, pp. 45–68. MIT Press, Cambridge, MA, USA (1996)
12. Miller, J.F., Smith, S.L.: Redundancy and computational efficiency in cartesian genetic programming. *IEEE Trans. EC*, **10**(2), 167–174 (2006)
13. Langdon, W.B.: Size fair and homologous tree genetic programming crossovers. *GP&EM*, **1**(1/2), 95–119 (2000)
14. Goldberg, D.E.: *Genetic Algorithms in Search Optimization and Machine Learning*. Addison-Wesley, (1989)
15. Angeline, P.J.: Subtree crossover: building block engine or macromutation? In: Koza et al., J.R. (eds.) GP 1997, pp. 9–17. Morgan Kaufmann, Stanford, 13–16 Jul.
16. Braitenberg, V.: *Vehicles*. MIT Press, (1984)
17. Poli, R., Page, J., Langdon, W.B.: Smooth uniform crossover, sub-machine code GP and demes: A recipe for solving high-order boolean parity problems. In: Banzhaf, W., Daida, J., Eiben, A.E., Garzon, M.H., Honavar, V., Jakielo, M., Smith, R.E. (eds.) *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pp. 1162–1169. Morgan Kaufmann, Orlando, Florida, USA (1999)
18. Chomsky, N.: Three models for the description of language. *IEEE Trans. Inf. Theory*, **2**(3), 113–124 (1956)
19. Montana, D.J.: Strongly typed genetic programming. *Evol. Comput.*, **3**(2), 199–230 (1995)
20. Recknagel et al., F.: Comparative application of artificial neural networks and genetic algorithms for multivariate time series modelling of algal blooms in freshwater lakes. *HydroInformatic*, **4**(2), 125–134 (2002)
21. Ratle, A., Sebag, M.: Genetic programming and domain knowledge: Beyond the limitations of grammar-guided machine discovery. In: Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., JulianMerelo, J., Schwefel, H.-P. (eds.) *Parallel Problem Solving from Nature – PPSN VI 6th International Conference*, volume 1917 of *LNCS*, pp. 211–220. Springer Verlag, Paris, France (2000)
22. Wong, M.L., Leung, K.S.: Inductive logic programming using genetic algorithms. In: Brahman, J.W., Lasker, G.E. (eds.) *Advances in Artificial Intelligence - Theory and Application II*, pp. 119–124. I.I.A.S., Ontario, Canada (1994)
23. Hussain, T.S., Browse, R.A.: Attribute grammars for genetic representations of neural networks and syntactic constraints of genetic programming. In *Workshop on Evolutionary Computation. Held at the 12 Canadian Conference on Artificial Intelligence*, Vancouver, Canada (1998)

24. Paterson, N.R., Livesey, M.: Distinguishing genotype and phenotype in genetic programming. In: John, R. Koza, (ed.) *Late Breaking Papers at the Genetic Programming 1996 Conference Stanford University July 28–31, 1996*, pp. 141–150. Stanford Bookstore, Stanford University, CA, USA 28–31 (1996)
25. Ryan et al., C.: Grammatical evolution: evolving programs for an arbitrary language. In: Banzhaf et al., W. (eds.) *EuroGP*, LNCS 1391, pp. 83–95. Paris, 14–15 Apr (1998)
26. Hoai et al., N.X.: Representation and structural difficulty in genetic programming. *IEEE Trans. EC*, **10**(2), 157–166 (2006)
27. Joshi, A.K., Levy, L.S., Takahashi, M.: Tree adjunct grammars. *J. Comput. Syst. Sci.* **10**, 136–163 (1975)
28. Ratle, A., Sebag, M.: Avoiding the bloat with probabilistic grammar-guided genetic programming. In: Collet et al., P. (eds), *EA 2001*, LNCS 2310, pp. 255–266
29. Shan, Y., McKay, R.I., Baxter, R., Abbass, H., Essam, D., Hoai, N.X.: Grammar model-based program evolution. In: *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pp. 478–485. IEEE Press, Portland, Oregon (2004)
30. Bosman, P.A.N., de Jong, E.D.: Grammar transformations in an EDA for genetic programming. In: Poli, R., Cagnoni, S., Keijzer, M., Costa, E., Pereira, F., Raidl, G., Upton, S.C., Goldberg, D., Lipson, H., de Jong, E., Koza, J., Suzuki, H., Sawai, H., Parmee, I., Pelikan, M., Sastry, K., Thierens, D., Stolzmann, W., Lanzi, P.L., Wilson, S.W., O'Neill, M., Ryan, C., Yu, T., Miller, J.F., Garibay, I., Holifield, G., Wu, A.S., Riopka, T., Meynsburg, M.M., Wright, A.W., Richter, N., Moore, J.H., Ritchie, M.D., Davis, L., Roy, R., Jakielia, M., (eds.) *GECCO 2004 Workshop Proceedings*, Seattle, Washington, USA 26–30 (2004)
31. Langdon, W.B., Poli, R.: Mapping non-conventional extensions of genetic programming. *Natural Comput.*, **7**, 21–43 (2008)
32. Teller, A.: The evolution of mental models. In: Kinnear, Jr., K.E. (ed.) *Advances in GP*, ch 9, pp. 199–219. MIT Press (1994)
33. Langdon, W.B.: *Genetic Programming and Data Structures: Genetic Programming +Data Structures = Automatic Programming!*, volume 1 of *Genetic Programming*. Kluwer, Boston, (1998)
34. Yu, T., Clack, C.: PolyGP: A polymorphic genetic programming system in haskell. In: Koza, J.R., Banzhaf, W., Chellapilla, K., Deb, K., Dorigo, M., Fogel, D.B., Garzon, M.H., Goldberg, D.E., Iba, H., Riolo, R. (eds.) *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pp. 416–421. University of Wisconsin, Madison, Wisconsin, USA, Morgan Kaufmann. (1998)
35. Binard, F., Felty, A.: Genetic programming with polymorphic types and higher order functions. In: Keijzer, M., Antoniol, G., Congdon, C.B., Deb, K., Doerr, B., Hansen, N., Holmes, J.H., Hornby, G.S., Howard, D., Kennedy, J., Kumar, S., Lobo, F.G., Miller, J.F., Moore, J., Neumann, F., Pelikan, M., Pollack, J., Sastry, K., Stanley, K., Stoica, A., Talbi, El-G., Wegener, I. (eds.) *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pp. 1187–1194. ACM, Atlanta, GA, USA (2008)
36. Spector, L., Robinson, A.: Genetic programming and autoconstructive evolution with the push programming language. *GP&EM*, **3**(1), 7–40 (2002)
37. Spector, L., Klein, J., Keijzer, M.: The push3 execution stack and the evolution of control. In: Beyer, H.-G., O'Reilly, U.-M., Arnold, D.V., Banzhaf, W., Blum, C., Bonabeau, E.W., Cantu-Paz, E., Dasgupta, D., Deb, K., Foster, J.A., de Jong, E.D., Lipson, H., Llora, X., Mancoridis, S., Pelikan, M., Raidl, G.R., Soule, T., Tyrrell, A.M., Watson, J.-P., Zitzler, E. (eds.) *GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation*, volume 2, pp. 1689–1696. ACM Press, Washington DC, USA (2005)
38. Koza, J.R., Andre, D.: Evolution of iteration in genetic programming. In: Fogel, L.J., Angelina, P.J., Baeck, T. (eds.) *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming*, San Diego, MIT Press, (1996)
39. Yu, T.: Hierachical processing for evolving recursive and modular programs using higher order functions and lambda abstractions. *GP&EM*, **2**(4), 345–380 (2001)

40. Whigham, P.A., McKay, R.I.: Genetic approaches to learning recursive relations. In: Yao, X. (ed.) *Progress in Evolutionary Computation*, LNAI 956, pp. 17–27. Springer (1995)
41. Brave, S.: Evolving recursive programs for tree search. In: Angeline, P.J., Kinnear, Jr., K.E. (eds.) *Advances in Genetic Programming 2*, chapter 10, pp. 203–220. MIT Press, Cambridge, MA, USA (1996)
42. Wong, M.L., Leung, K.S.: Evolving recursive functions for the even-parity problem using genetic programming. In: Angeline, P.J., Kinnear, Jr., K.E. (eds.) *Advances in GP 2*, ch 11, pp. 221–240. MIT Press (1996)
43. Yu, T.: A higher-order function approach to evolve recursive programs. In: Yu et al., T. (eds.) *GPTP III*, ch 7, pp. 93–108. Springer, Ann Arbor, 12–14 May (2005)
44. Agapitos, A., Lucas, S.M.: Learning recursive functions with object oriented genetic programming. In: Collet, P., Tomassini, M., Ebner, M., Gustafson, S., Ekárt, A. (eds.) *Proceedings of the 9th European Conference on Genetic Programming*, volume 3905 of *Lecture Notes in Computer Science*, pp. 166–177. Budapest, Hungary, Springer (2006)
45. Koza, J.R.: *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge Massachusetts, (1994)
46. Koza et al., J.R.: *Genetic Programming 3: Darwinian Invention and Problem Solving*. Morgan Kaufman (1999)
47. Angeline, P.J., Pollack, J.: Evolutionary module acquisition. In: Fogel, D., Atmar, W. (eds.) *Proceedings of the Second Annual Conference on Evolutionary Programming*, pp. 154–163. La Jolla, CA, USA (1993)
48. Kinnear, K.E. Jr.: Alternatives in automatic function definition: A comparison of performance. In: Kinnear, K.E. Jr., (ed.) *Advances in Genetic Programming*, chapter 6, pp. 119–141. MIT Press, (1994)
49. Walker, J.A., Miller, J.F.: The automatic acquisition, evolution and reuse of modules in cartesian genetic programming. *IEEE Trans. EC* **12**(4), 397–417 (2008)
50. Spector, L.: Simultaneous evolution of programs and their control structures. In: Angeline, P.J., Kinnear, K.E. Jr., (eds.) *Advances in Genetic Programming 2*, chapter 7, pp. 137–154. MIT Press, Cambridge, MA, USA (1996)
51. Bruce, W.S.: Automatic generation of object-oriented programs using genetic programming. In: Koza et al., J.R. (eds.) *GP 1996*, pp. 267–272. MIT Press, Stanford, 28–31 Jul 1996
52. Lucas, S.: Exploiting reflection in object oriented genetic programming. In: Keijzer et al., M. (eds.) *EuroGP*, LNCS 3003, pp. 369–378. Springer 5–7 Apr (2004)
53. Briggs, F., O'Neill, M.: Functional genetic programming with combinators. In: Pham et al., T.L. (eds.) *Proceedings of the Third Asian-Pacific workshop on Genetic Programming*, ASPGP, pp. 110–127, Military Technical Academy, Hanoi, Vietnam (2006)
54. Banzhaf, W.: Genotype-phenotype-mapping and neutral variation – A case study in genetic programming. In: Davidor, Y., Schwefel, H.-P., Männer, R. (eds.) *Parallel Problem Solving from Nature III*, volume 866 of *LNCS*, pp. 322–332, Springer-Verlag, Jerusalem (1994)
55. Jacob, C.: Genetic L-system programming. In: Davidor, Y., Schwefel, H.-P., Männer, R. (eds.) *Parallel Problem Solving from Nature III*, volume 866 of *LNCS*, pp. 334–343. Springer-Verlag, Jerusalem (1994)
56. Spector, L., Stoffel, K.: Ontogenetic programming. In: Koza et al., J.R. (eds.) *GP-96*, pp. 394–399. Stanford, MIT Press 28–31 Jul (1996)
57. Harding, S.L., Miller, J.F., Banzhaf, W.: Self-modifying cartesian genetic programming. In: Thierens, D., Beyer, H.-G., Bongard, J., Branke, J., Clark, J.A., Cliff, D., Congdon, C.B., Deb, K., Doerr, B., Kovacs, T., Kumar, S., Miller, J.F., Moore, J., Neumann, F., Pelikan, M., Poli, R., Sastry, K., Stanley, K.O., Stutzle, T., Watson, R.A., Wegener, I. (eds.) *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, volume 1, pp. 1021–1028. ACM Press, London (2007)
58. Gruau, F.: Genetic micro programming of neural networks. In: Kinnear, Jr., Kenneth, E. (ed.) *Advances in Genetic Programming*, chapter 24, pp. 495–518. MIT Press, (1994)
59. Koza, J.R., Keane, M.A., Streeter, M.J., Mydlowec, W., Yu, J., Lanza, G.: *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers, (2003)

60. Gottlieb, G.: *Individual development and evolution: The genesis of novel behavior*. Oxford University Press, New York (1992)
61. Gottlieb, G.: Individual development and evolution: The genesis of novel behavior. Oxford University Press, New York (1992)
62. Miller, J.F.: Evolving developmental programs for adaptation, morphogenesis, and self-repair. In: Banzhaf et al., W. (eds.), ECAI, LNAI 2801, pp. 256–265. Springer 14–17 Sep (2003)
63. Angeline, P.J.: Adaptive and self-adaptive evolutionary computations. In: Palaniswami, M., Attikiouzel, Y. (eds.) *Computational Intelligence: A Dynamic Systems Perspective*, pp. 152–163. IEEE Press, (1995)
64. Meyer-Nieberg, S., Beyer, H.-G.: Self-adaptation in evolutionary algorithms. Parameter Setting in Evolutionary Algorithm. Springer (2006)
65. Schmidhuber, J.: Evolutionary principles in self-referential learning. on learning now to learn: the meta-meta-meta...hook. Diploma thesis, Technische Universitat Munchen, Germany, 14 May (1987)
66. Edmonds, B.: Meta-genetic programming: Co-evolving the operators of variation. CPM Report 98-32, Centre for Policy Modelling, Manchester Metropolitan University, UK, AytonSt., Manchester, M1 3GH. UK, (1998)
67. Lohn et al., J.: Evolutionary antenna design for a NASA spacecraft. In: O'Reilly et al., U.-M. eds. GPTP II, ch 18, pp. 301–315. Springer, Ann Arbor, 13–15 May 2004
68. Langdon, W.B., Buxton, B.F.: Genetic programming for mining DNA chip data from cancer patients. Genet. Prog. Evol. Mach., **5**(3), 251–257, (2004)
69. Spector, L., Barnum, H., Bernstein, H.J., Swamy, N.: Finding a betterthan-classical quantumAND/OR algorithm using genetic programming. In: Angeline, P.J., Michalewicz, Z., Schoenauer, M., Yao, X., Zalzala, A. (eds.) *Proceedings of the Congress on Evolutionary Computation*, volume
70. Spector, L., Bernstein, H.J.: Communication capacities of some quantum gates, discovered in part through genetic programming. In: Shapiro, Jeffrey H., Hirota, O. (eds) *Proceedings of the Sixth International Conference on Quantum Communication, Measurement, and Computing (QCMC)*, pp. 500–503. Rinton Press, (2003)
71. Jordaan, E., Kordon, A., Chiang, L., Smits, G.: Robust inferential sensors based on ensemble of predictors generated by genetic programming. In: Yao, X., Burke, E., Lozano, J.A., Smith, J., Merelo-Guervós, J.J., Bullinaria, J.A., Rowe, J., Tiño Ata Kabán, P., Schwefel, H.-P. (eds.) *Parallel Problem Solving from Nature – PPSN VIII*, volume 3242 of LNCS, pp. 522–531. Springer-Verlag Birmingham, UK (2004)
72. Langdon, W.B., Buxton, B.F.: Genetic programming for mining DNA chip data from cancer patients. GP&EM, **5**(3), 251–257 (2004)
73. Hampo, R.J., Marko, K.A.: Application of genetic programming to control of vehicle systems. In: *Proceedings of the Intelligent Vehicles '92 Symposium*, pp. 191–195. IEEE, Detroit, Mi, USA (1992)
74. D. H. and S.C. Roberts. Incident detection on highways. In U.-M. O'Reilly et al., eds., *GPTP II*, ch 16, pp 263–282. Springer, Ann Arbor, 13–15 May (2004)
75. Ross et al., B.J.: Hyperspectral image analysis using genetic programming. Appl. Soft Comput. **5**(2), 147–156 (2005)
76. Zhang, M., Smart, W.: Using gaussian distribution to construct fitness functions in genetic programming for multiclass object classification. Pattern. Recognit. Lett., **27**(11), 1266–1274, Evolutionary Computer Vision and Image Understanding. (2006)
77. Cilibrasi, R., Vitanyi, P.M.B.: Clustering by compression. IEEE Trans. Inf. Theory, **51**(4), 1523–1545 (2005)
78. Klassen, T.J., Heywood, M.I.: Towards the on-line recognition of arabic characters. In *Proceedings of the 2002 International Joint Conference on Neural Networks IJCNN'02*, pp. 1900–1905. IEEE Press, Hilton Hawaiian Village Hotel, Honolulu, Hawaii, (2002)
79. De Stefano et al., C.: Character preclassification based on genetic programming. Pattern Recognit. Lett. **23**(12), 1439–1448 (2002)

80. Teredesai, A., Govindaraju, V.: GP-based secondary classifiers. *Pattern Recognit.* **38**(4), 505–512 (2005)
81. Parkins, A.D., Nandi, A.K.: Genetic programming techniques for hand written digit recognition. *Signal Process.*, **84**(12), 2345–2365 Dec (2004)
82. Quintana et al., M.I.: Morphological algorithm design for binary images using genetic programming. *GP&EM*, **7**(1), 81–102 (2006)
83. Martin, M.C. Evolving visual sonar: Depth from monocular images. *Pattern Recognition Letters*, **27**(11), 1174–1180 Evolutionary Computer Vision and Image Understanding. (2006)
84. Usman et al., I.: Image authenticity and perceptual optimization via genetic algorithm and a dependence neighborhood. *IJAMCS*, **4**(1), 615–620 (2007)
85. Chen, S.-H., Liao, C.-C.: Agent-based computational modeling of the stock price-volume relation. *Information Sciences*, **170**(1), 75–100 (2005)
86. Yu, T., Chen, S.-H.: Using genetic programming with lambda abstraction to find technical trading rules. In: *Computing in Economics and Finance*, University of Amsterdam, (2004)
87. Neely et al., C.J.: Is technical analysis in the foreign exchange market profitable? A GP approach. *J. Finan. Quant. Anal.* **32**(4), 405–426 (1997)
88. Marney, J.P., Miller, D., Fyfe, C., Tarbert, H.F.E.: Risk adjusted returns to technical trading rules: a genetic programming approach. In: *7th International Conference of Society of Computational Economics*, Yale, 28–29 (2001)
89. Neely et al., C.J.: The adaptive markets hypothesis: evidence from the foreign exchange market. Working Paper 2006-046B, Federal Reserve Bank of St. Louis Rev (2007)
90. Kaboudan, M.: Extended daily exchange rates forecasts using wavelet temporal resolutions. *New Math. Nat. Comput.* **1**, 79–107 (2005)
91. Tsang et al., E.P.K.: EDDIE beats the bookies. *Softw. Pract. Exper.*, **28**(10), 1033–1043 (1998)
92. Austin et al., M.P.: Adaptive systems for foreign exchange trading. *Quant. Finan.*, **4**(4), 37–45 (2004)
93. Pillay, N.: Evolving solutions to ASCII graphics programming problems in intelligent programming tutors. In: Akerkar, R. ed. International Conference on Applied Artificial Intelligence (ICAAI'2003), pp. 236–243. Fort Panhala, Kolhapur, India TMRF. 15–16 Dec (2003)
94. Kordon, A.: Evolutionary computation in the chemical industry. In: Yu et al., T. (eds.) *Evolutionary Computation in Practice*, ch 11, pp. 245–262. Springer (2008)
95. Kovacic, M., Balic, J.: Evolutionary programming of a CNC cutting machine. *Int. J. Advan. Manufact. Technol.*, **22**(1–2), 118–124 (2003)
96. Deschaine, L.: Using information fusion, machine learning, and global optimisation to increase the accuracy of finding and understanding items interest in the subsurface. *GeoDrill. Int.* **122**, 30–32 May (2006)
97. Barriere, O., Lutton, E., Baudrit, C., Sicard, M., Pinaud, B., Perrot, N.: Modeling human expertise on a cheese ripening industrial process using GP. In: Rudolph, G. (ed.) *PPSN X*, volume 5199 of *LNCS*, Springer, Dortmund (2008)
98. Dassau et al., E.: Modeling and temperature control of rapid thermal processing. *Comput. Chem. Eng.*, **30**(4), 686–697 (2006)
99. Rodriguez-Vazquez et al., K.: Identifying the structure of nonlinear dynamic systems using multiobjective genetic programming. *IEEE Trans. Syst., Man Cyberne., Part A*, **34**(4), 531–545 (2004)
100. Kell, D.: Defence against the flood. *Bioinformatics World*, pp. 16–18. (2002)
101. Moore et al., J.H.: Symbolic discriminant analysis of microarray data in autoimmune disease. *Genet. Epidemiol.*, **23**, 57–69 (2002)
102. Barrett, S.J., Langdon, W.B.: Advances in the application of machine learning techniques in drug discovery, design and development. In: Tiwari, A., Knowles, J., Avineri, E., Dahal, K., Roy, R. (eds.) *Applications of Soft Computing:Recent Trends*, Advances in Soft Computing, pp. 99–110. On the World Wide Web, Springer, (2006)
103. Bains et al., W.: Evolutionary computational methods to predict oral bioavailability QSPRs. *Curr. Opin. Drug Discov. Develop.*, **5**(1), 44–51 (2002)

104. Fukunaga, A.: Automated discovery of composite SAT variable selection heuristics. In: AAAI, pp. 641–648. (2002)
105. Keller, R.E., Poli, R.: Cost-benefit investigation of a genetic-programming hyperheuristic. In *Evolution Artificielle, 8th International Conference*, Lecture Notes in Computer Science, Tours, Springer, France (2007)
106. Azaria, Y., Sipper, M.: GP-gammon: genetically programming backgammon players. *Gene. Program. Evolvable Mach.*, **6**(3), 283–300 (2005)
107. Funes, P., Sklar, E., Juille, H., Pollack, J.: Animal-animat coevolution: using the animal population as fitness function. In: Pfeifer et al., R. (eds.) SAB, pp. 525–533. Zurich, Aug 17–21 (1998). MIT Press
108. Machado, P., Romero, J. (eds.): *The Art of Artificial Evolution*. Springer (2008)
109. Spector, L., Alpern, A.: Criticism, culture, and the automatic generation of artworks. In: AAAI-94, pp. 3–8. Seattle, Washington, USA AAAI Press/MIT Press (1994)
110. Takagi, H.: Interactive evolutionary computation: Fusion of the capabilities of EC optimization and human evaluation. *Proc. IEEE*, **89**(9), 1275–1296 (2001)
111. Langdon, W.B.: Global distributed evolution of L-systems fractals. In: Keijzer et al., M. (eds.) EuroGP'2004, LNCS 3003, pp. 349–358. Coimbra, Portugal, Springer 5–7 Apr (2004)
112. Reynolds, C.W.: Flocks, herds, and schools: a distributed behavioral model. *SIGGRAPH Comput. Graph.*, **21**(4), 25–34 (1987)
113. Koza et al., J.R.: Automatic creation of human-competitive programs and controllers by means of genetic programming. *GP&EM*, **1**(1/2), 121–164 (2000)
114. Koza, J.R., Al-Sakran, S.H., Jones, L.W.: Automated re-invention of six patented optical lens systems using genetic programming. In: Beyer, H.-G., O'Reilly, U.-M., Arnold, D.V., Banzhaf, W., Blum, C., Bonabeau, E.W., Cantu-Paz, E., Dasgupta, D., Deb, K., Foster, J.A., de Jong, E.D., Lipson, H., Llora, X., Mancoridis, S., Pelikan, M., Raidl, G.R., Soule, T., Tyrrell, A.M., Watson, J.-P., Zitzler, E. (eds.) *GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation*, volume 2, pp. 1953–1960. ACM Press, Washington DC, USA (2005)
115. Kinnear, K.E., Jr.: A perspective on the work in this book. In: Kinnear, Jr., K.E. (ed.) *Advances in Genetic Programming*, chapter 1, pp. 3–19. MIT Press, (1994)
116. Corney, D.P.A.: Intelligent analysis of small data sets for food design. Ph.D. thesis, University College, London (2002)
117. Daida, J.M., Hilss, A.M., Ward, D.J., Stephen, L.: Long. Visualizing tree structures in genetic programming. *Genet. Prog. Evol. Mach.*, **6**(1), 79–110 (2005)
118. McPhee, N.F., Ohs, B., Hutchison, T., Semantic building blocks in genetic programming. In: O'Neill, M., Vanneschi, L., Gustafson, S., Alcazar, A.I.E., De Falco, I., Cioppa, A.D., Tarantino, E. (eds.) *Proceedings of the 11th European Conference on Genetic Programming, EuroGP 2008*, volume 4971 of *Lecture Notes in Computer Science*, pp. 134–145. Springer, Naples (2008)
119. Crane, E.F., McPhee, N.F.: The effects of size and depth limits on tree based genetic programming. In: Yu et al., T. (eds.) *Genetic Programming Theory and Practice III*, ch 15, pp. 223–240. Springer, Ann Arbor, 12–14 May (2005)
120. Spector, L., Klein, J.: Trivial geography in genetic programming. In: Yu et al., T. (eds.) *GPTP III*, ch 8, pp. 109–123. Springer, Ann Arbor, 12–14 May (2005)
121. Gelly, S., Teytaud, O., Bredeche, N., Schoenauer, M.: Universal consistency and bloat in GP. Issue on New Methods in Machine Learning. *Theory Appl. Rev. d'Intelligence Artif.*, **20**(6), 805–827 (2006)
122. Yu, T., Davis, D., Baydar, C., Roy, R. (eds.) *Evolutionary Computation in Practice*, volume 88 of *Studies in Computational Intelligence*. Springer, (2008)



# Chapter 8

## Ant Colony Optimization: Overview and Recent Advances

Marco Dorigo and Thomas Stützle

**Abstract** Ant Colony Optimization (ACO) is a metaheuristic that is inspired by the pheromone trail laying and following behavior of some ant species. Artificial ants in ACO are stochastic solution construction procedures that build candidate solutions for the problem instance under concern by exploiting (artificial) pheromone information that is adapted based on the ants' search experience and possibly available heuristic information. Since the proposal of the Ant System, the first ACO algorithm, many significant research results have been obtained. These contributions focused on the development of high-performing algorithmic variants, the development of a generic algorithmic framework for ACO algorithms, successful applications of ACO algorithms to a wide range of computationally hard problems, and the theoretical understanding of properties of ACO algorithms. This chapter reviews these developments and gives an overview of recent research trends in ACO.

### 8.1 Introduction

Ant Colony Optimization (ACO) [57, 59, 66] is a metaheuristic for solving hard combinatorial optimization problems. The inspiring source of ACO is the pheromone trail laying and following behavior of real ants, which use pheromones as a communication medium. In analogy to the biological example, ACO is based on indirect communication within a colony of simple agents, called (artificial) ants, mediated by (artificial) pheromone trails. The pheromone trails in ACO serve as a distributed, numerical information, which the ants use to probabilistically construct solutions to the problem being solved and which the ants adapt during the algorithm's execution to reflect their search experience.

---

Marco Dorigo and Thomas Stützle  
IRIDIA, Université Libre de Bruxelles (ULB), Brussels, Belgium  
e-mail: {mdorigo, stuetzle}@ulb.ac.be

The first example of such an algorithm is Ant System (AS) [55, 63–65], which was proposed using as example application the well-known traveling salesman problem (TSP) [6, 99, 128]. Despite encouraging initial results, AS could not compete with state-of-the-art algorithms for the TSP. Nevertheless, it had the important role of stimulating further research both on algorithmic variants, which obtain much better computational performance, and on applications to a large variety of different problems. In fact, there exist now a considerable number of applications of such algorithms where world class performance is obtained. Examples are applications of ACO algorithms to problems such as sequential ordering [76], scheduling [18], assembly line balancing [19], probabilistic TSP [7], 2D-HP protein folding [132], DNA sequencing [25], protein–ligand docking [98], and packet-switched routing in Internet-like networks [47]. The ACO metaheuristic provides a common framework for the existing applications and algorithmic variants [57, 59]. Algorithms which follow the ACO metaheuristic are called ACO algorithms.

The (artificial) ants in ACO implement a randomized construction heuristic which makes probabilistic decisions as a function of artificial pheromone trails and possibly available heuristic information based on the input data of the problem to be solved. As such, ACO can be interpreted as an extension of traditional construction heuristics, which are readily available for many combinatorial optimization problems. Yet, an important difference with construction heuristics is the adaptation of the pheromone trails during algorithm execution to take into account the cumulated search experience.

The rest of this chapter is organized as follows. In Section 8.2, we briefly overview construction heuristics and local search algorithms. In Section 8.3, we present a specific version of the ACO metaheuristic that focuses on applications to  $\mathcal{NP}$ -hard problems. Section 8.4 outlines the inspiring biological analogy and describes the historical developments leading to ACO. In Section 8.5, we illustrate how the ACO metaheuristic can be applied to different types of problems and we give an overview of its successful applications. Section 8.6 gives an overview of recent developments in ACO and Section 8.7 concludes the chapter.

## 8.2 Approximate Approaches

Many important combinatorial optimization problems are hard to solve. The notion of problem hardness is captured by the theory of computational complexity [79, 123] and for many important problems it is well known that they are  $\mathcal{NP}$ -hard, that is, the time needed to solve an instance in the worst case grows exponentially with instance size. Often, approximate algorithms are the only feasible way to obtain near-optimal solutions at relatively low computational cost.

Most approximate algorithms are either *construction* algorithms or *local search* algorithms.<sup>1</sup> These two types of methods are significantly different, because

---

<sup>1</sup> Other approximate methods are also conceivable. For example, when stopping exact methods, like Branch & Bound, before completion [10, 95] (for example, after some given time bound,

construction algorithms work on partial solutions trying to extend these in the best possible way to complete problem solutions, while local search methods move in the search space of complete solutions.

### 8.2.1 Construction Algorithms

Construction algorithms build solutions to a problem under consideration in an incremental way starting with an empty initial solution and iteratively adding appropriate solution components without backtracking until a complete solution is obtained. In the simplest case, solution components are added in random order. Often better results are obtained if a heuristic estimate of the myopic benefit of adding solution components is taken into account. *Greedy construction heuristics* add at each step a solution component that achieves the maximal myopic benefit as measured by some heuristic information. An algorithmic outline of a greedy construction heuristic is given in Figure 8.1. The function `GreedyComponent` returns the solution component  $e$  with the best heuristic estimate as a function of the current partial solution  $s_p$ . Solutions returned by greedy algorithms are typically of (much) better quality than randomly generated solutions. Yet, a disadvantage of greedy construction heuristics is that they typically generate only a limited number of different solutions. Additionally, greedy decisions in early stages of the construction process constrain the available choices at later stages, often causing very poor moves in the final phases of the solution construction.

```

procedure Greedy Construction Heuristic
     $s_p = \text{empty solution}$ 
    while  $s_p$  not_a_complete_solution do
         $e = \text{GreedyComponent}(s_p)$ 
         $s_p = s_p \otimes e$ 
    end
    return  $s_p$ 
end Greedy Construction Heuristic

```

**Fig. 8.1** Algorithmic skeleton of a greedy construction heuristic. The addition of component  $e$  to a partial solution  $s_p$  is denoted by the operator  $\otimes$ .

As an example, consider a greedy construction heuristic for the TSP. In the TSP we are given a complete weighted graph  $G = (V, E)$  with  $V$  being the set of vertices, representing the cities, and  $E$  the set of edges fully connecting the vertices. Each edge is assigned a value  $d_{ij}$ , which is the length of edge  $(i, j) \in E$ . The TSP is the problem of finding a minimal length Hamiltonian circuit of the graph, where an Hamiltonian circuit is a closed tour visiting exactly once each of the  $n = |V|$  vertices of  $G$ . For symmetric TSPs, the distances between the cities are independent of the

---

or when some guarantee on the solution quality is obtained through the use of lower and upper bounds), we can convert exact algorithms into approximate ones.

direction of traversing the edges, that is,  $d_{ij} = d_{ji}$  for every pair of vertices. In the more general asymmetric TSP (ATSP) at least for one pair of vertices  $i, j$  we have  $d_{ij} \neq d_{ji}$ .

A simple rule of thumb to build a tour is to start from some initial city and to always choose to go to the closest still unvisited city before returning to the start city. This algorithm is known as the *nearest neighbor* tour construction heuristic.

Construction algorithms are typically the fastest approximate methods, but the solutions they generate often are not of a very high quality and they are not guaranteed to be optimal with respect to small changes; the results produced by constructive heuristics can therefore often be improved by local search algorithms.

### 8.2.2 Local Search Algorithms

Local search algorithms start from a complete initial solution and try to find a better solution in an appropriately defined *neighborhood* of the current solution. In its most basic version, known as *iterative improvement*, the algorithm searches the neighborhood for an improving solution. If such a solution is found, it replaces the current solution and the local search continues. These steps are repeated until no improving neighbor solution can be found and the algorithm ends in a *local optimum*. An outline of an iterative improvement algorithm is given in Figure 8.2. The procedure `Improve` returns a better neighbor solution if one exists, otherwise it returns the current solution, in which case the algorithm stops.

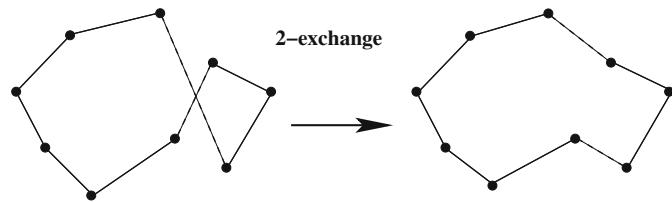
```

procedure IterativeImprovement ( $s \in S$ )
     $s' = \text{Improve}(s)$ 
    while  $s' \neq s$  do
         $s = s'$ 
         $s' = \text{Improve}(s)$ 
    end
    return  $s$ 
end IterativeImprovement

```

**Fig. 8.2** Algorithmic skeleton of iterative improvement. As input is given a complete solution  $s$  in search space  $S$ .

The choice of an appropriate neighborhood structure is crucial for the performance of local search algorithms and has to be done in a problem-specific way. The neighborhood structure defines the set of solutions that can be reached from  $s$  in one single step of the algorithm. An example neighborhood for the TSP is the *k-exchange* neighborhood in which neighbor solutions differ by at most  $k$  edges. Figure 8.3 shows an example of a 2-exchange neighborhood. The 2-exchange algorithm systematically tests whether the current tour can be improved by replacing two edges. To fully specify a local search algorithm, it is necessary to designate



**Fig. 8.3** Schematic illustration of a 2-exchange move. The proposed move reduces the total tour length if we consider the Euclidean distance between the points.

a particular neighborhood examination scheme that defines how the neighborhood is searched and which neighbor solution replaces the current one. In the case of iterative improvement algorithms, this rule is called the *pivoting rule* [157] and examples are the *best-improvement* rule, which chooses the neighbor solution giving the largest improvement of the objective function, and the *first-improvement* rule, which uses the first improved solution found when scanning the neighborhood to replace the current one. A common problem with local search algorithms is that they easily get trapped in local minima and that the result strongly depends on the initial solution.

### 8.3 The ACO Metaheuristic

Artificial ants used in ACO are stochastic solution construction procedures that probabilistically build a solution by iteratively adding solution components to partial solutions by taking into account (i) heuristic information about the problem instance being solved, if available, and (ii) (artificial) pheromone trails which change dynamically at runtime to reflect the agents' acquired search experience.

A stochastic component in ACO allows the ants to build a wide variety of different solutions and hence to explore a much larger number of solutions than greedy heuristics. At the same time, the use of heuristic information, which is readily available for many problems, can guide the ants toward the most promising solutions. More important, the ants' search experience can be used to influence, in a way reminiscent of reinforcement learning [149], the solution construction in future iterations of the algorithm. Additionally, the use of a colony of ants can give the algorithm increased robustness and in many ACO applications the collective interaction of a population of agents is needed to efficiently solve a problem.

The domain of application of ACO algorithms is vast. In principle, ACO can be applied to any discrete optimization problem for which some solution construction mechanism can be conceived. In Section 8.3.1, we first define a generic problem representation that the ants in ACO may exploit to construct solutions, and then we define the ACO metaheuristic.

### 8.3.1 Problem representation

Let us consider minimization problems<sup>2</sup> and define a general model of a combinatorial optimization problem.

**Definition 8.1** A model  $P = (S, \Omega, f)$  of a combinatorial optimization problem consists of

- a search space  $S$  that is defined by a finite set of decision variables, each with a finite domain, and a set  $\Omega$  of constraints among the variables;
- an objective function  $f : S \rightarrow \mathbb{R}_0^+$  to be minimized.

The search space is defined by a finite set of variables  $X_i, i = 1, \dots, n$ , each having an associated domain  $D_i$  of values that can be assigned to it. An instantiation of a variable consists of an assignment of a value  $v_i^j \in D_i$  to variable  $X_i$  and it is denoted by  $X_i = v_i^j$ . A feasible solution  $s \in S$  is an assignment to each variable of a value in its domain such that all the problem constraints in  $\Omega$  are satisfied. If  $\Omega$  is empty, then the problem is unconstrained and each decision variable can take any value from its domain, independent of the other variables. In this case,  $P$  is an *unconstrained* problem model; otherwise it is called *constrained*. A feasible solution  $s^* \in S$  is called a global minimum of  $P$  if and only if we have that  $f(s^*) \leq f(s) \forall s \in S$ . We denote by  $S^* \subseteq S$  the set of all global minima.  $\square$

This model of a combinatorial optimization problem can be directly used to derive a generic pheromone model that is exploited by ACO algorithms. To see how, let us call the instantiation of a variable  $X_i$  with a particular value  $v_i^j$  of its domain a solution component, which is denoted by  $c_i^j$ . Ants then need to appropriately combine solution components to form high-quality, feasible solutions. To do so, each solution component  $c_i^j$  will have an associated pheromone variable  $T_{ij}$ . We denote the set of all solution components by  $C$  and the set of all pheromone variables by  $T$ . Each pheromone variable  $T_{ij}$  has a pheromone value  $\tau_{ij}$ ; this value indicates the desirability of choosing solution component  $c_i^j$ . Note that, as said before, the pheromone values are time varying and therefore they are a function of the algorithm iteration  $t$ . In what follows we will, however, omit the reference to the iteration counter and write simply  $\tau_{ij}$  instead of  $\tau_{ij}(t)$ .

As an example of this formalization, consider the TSP. In this case, the solution components are the moves from one city to another one. This can be formalized by associating one variable to each city. Each variable  $X_i$  has then associated  $n - 1$  values,  $j = 1, \dots, n, j \neq i$ . As a result, a pheromone value  $\tau_{ij}$  is associated to each edge connecting two cities. An instantiation of the decision variables corresponds to a feasible solution, if and only if the set of edges corresponding to the values of the decision variables forms a Hamiltonian cycle. (Note that for the TSP it is easy to guarantee that ants generate feasible solutions.) The objective function  $f(\cdot)$  computes for each feasible solution the sum of the edge lengths, that is, the length of the Hamiltonian cycle.

---

<sup>2</sup> The adaptation to maximization problems is straightforward.

### 8.3.2 The Metaheuristic

A general outline of the ACO metaheuristic for applications to static combinatorial optimization problems<sup>3</sup> is given in Figure 8.4. After initializing parameters and pheromone trails, the main loop consists of three main steps. First,  $m$  ants construct solutions to the problem instance under consideration, biased by the pheromone information and possibly by the available heuristic information. Once the ants have completed their solutions, these may be improved in an optional local search phase. Finally, before the start of the next iteration, the pheromone trails are adapted to reflect the search experience of the ants. The main steps of the ACO metaheuristic are explained in more detail in the following.

```

procedure ACO algorithm for combinatorial optimization problems
    Initialization
    while (termination condition not met) do
        ConstructAntSolutions
        ApplyLocalSearch      % optional
        UpdatePheromones
    end
end ACO algorithm for combinatorial optimization problems

```

**Fig. 8.4** Algorithmic skeleton for ACO algorithms applied to combinatorial optimization problems. The application of a local search algorithm is a typical example of a possible daemon action in ACO algorithms.

**Initialization.** At the start of the algorithm, parameters are set and all pheromone variables are initialized to a value  $\tau_0$ , which is a parameter of the algorithm.

**ConstructAntSolutions.** A set of  $m$  ants constructs solutions to the problem instance being tackled. To do so, each ant starts with an initially empty solution  $s_p = \emptyset$ . At each construction step, an ant extends its current partial solution  $s_p$  by choosing one feasible solution component  $c_i^j \in N(s_p) \subseteq C$  and adding it to its current partial solution.  $N(s_p)$  is the set of solution components that may be added while maintaining feasibility and it is defined implicitly by a solution construction process that the ants implement. If a partial solution cannot be extended maintaining feasibility, it depends on the particular construction mechanism whether the solution construction is abandoned or an infeasible, complete solution is constructed. In the latter case, infeasible solutions may be penalized in dependence of the degree to which they violate problem constraints.

---

<sup>3</sup> Static problems are those whose topology and costs do not change while they are being solved. This is the case, for example, for the classic TSP, in which city locations and intercity distances do not change during the algorithm's runtime. In contrast, in dynamic problems the topology and costs can change while solutions are built. An example of such a problem is routing in telecommunications networks [47], in which traffic patterns change all the time.

The choice of the solution component to add is done probabilistically at each construction step. Various ways for defining the probability distributions have been considered. The most widely used rule is that of Ant System (AS) [65]:

$$p(c_i^j | s_p) = \frac{\tau_{ij}^\alpha \cdot [\eta(c_i^j)]^\beta}{\sum_{c_l^l \in N(s_p)} \tau_{il}^\alpha \cdot [\eta(c_l^l)]^\beta}, \quad \forall c_i^j \in N(s_p), \quad (8.1)$$

where  $\eta(\cdot)$  is a function that assigns to each feasible solution component  $c_i^j \in N(s_p)$  a heuristic value, which is usually called the heuristic information. Parameters  $\alpha$  and  $\beta$  determine the relative influence of the pheromone trails and the heuristic information and have the following influence on the algorithm behavior. If  $\alpha = 0$ , the selection probabilities are proportional to  $[\eta_{ij}]^\beta$  and a solution component with a high heuristic value will more likely be selected: this case corresponds to a stochastic greedy algorithm. If  $\beta = 0$ , only pheromone amplification is at work.

`ApplyLocalSearch`. Once complete candidate solutions are obtained, these may further be improved by applying local search algorithms. In fact, for a wide range of combinatorial optimization problems, ACO algorithms reach best performance when coupled with local search algorithms [66]. More generally, local search is one example of what have been called *daemon actions* [57, 59]. These are used to implement problem specific or centralized actions that cannot be performed by individual ants.

`UpdatePheromones`. The pheromone update is intended to make solution components belonging to good solutions more desirable for ants operating in the following iterations. There are essentially two mechanisms that are used to achieve this goal. The first is pheromone deposit, which increases the level of the pheromone of solution components that are associated with a chosen set  $S_{\text{upd}}$  of good solutions. The second is pheromone trail evaporation, which is the mechanism that decreases over time the pheromone deposited by previous ants. From a practical point of view, pheromone evaporation is needed to avoid a too rapid convergence of the algorithm toward a suboptimal region. It implements a useful form of *forgetting*, favoring the exploration of new areas of the search space. The pheromone update is commonly implemented as:

$$\tau_{ij} = (1 - \rho)\tau_{ij} + \sum_{s \in S_{\text{upd}} | c_i^j \in s} g(s), \quad (8.2)$$

where  $S_{\text{upd}}$  is the set of solutions that are used to deposit pheromone,  $\rho \in (0, 1]$  is a parameter called evaporation rate,  $g(\cdot) : S \mapsto \mathbb{R}^+$  is a function such that  $f(s) < f(s') \Rightarrow g(s) \geq g(s')$ . It determines the quality of a solution and is commonly called *evaluation function*.

ACO algorithms typically differ in the way pheromone update is implemented: different specifications of how to determine  $S_{\text{upd}}$  result in different instantiations of update rule 8.2. Typically,  $S_{\text{upd}}$  is a subset of  $S_{\text{iter}} \cup \{s_{\text{gb}}\}$ , where  $S_{\text{iter}}$  is the set of all solutions constructed in the current iteration of the main loop and  $s_{\text{gb}}$  is the best solution found since the start of the algorithm (gb stands for global-best).

## 8.4 History

The first ACO algorithm to be proposed was Ant System (AS). AS was applied to some rather small TSP instances with up to 75 cities. It was able to reach the performance of other general-purpose heuristics like evolutionary computation [55, 65]. Despite these initial encouraging results, AS did not prove to be competitive with state-of-the-art algorithms specifically designed for the TSP. Therefore, a substantial amount of research in ACO has focused on extending/modifying AS to improve its performance. In Section 8.4.1 we first briefly introduce the biological metaphor by which AS and ACO are inspired, and then we present a brief history of the early developments that have led from the original AS to more performing ACO algorithms.

### 8.4.1 Biological Analogy

In many ant species, individual ants may deposit a pheromone (a chemical that ants can smell) on the ground while walking [43, 80]. By depositing pheromone, ants create a trail that is used, for example, to mark the path from the nest to food sources and back. Foragers can sense the pheromone trails and follow the path to food discovered by other ants. Several ant species are capable of exploiting pheromone trails to determine the shortest among the available paths leading to the food.

Deneubourg and colleagues [43, 80] used a double bridge connecting a nest of ants and a food source to study pheromone trail laying and following behavior in controlled experimental conditions.<sup>4</sup> They ran a number of experiments in which they varied the ratio between the length of the two branches of the bridge. The most interesting, for our purposes, of these experiments is the one in which one branch was longer than the other. In this experiment, at the start the ants were left free to move between the nest and the food source and the percentage of ants that chose one or the other of the two branches was observed over time. The outcome was that, although in the initial phase random oscillations could occur, in most experiments all the ants ended up using the shorter branch.

This result can be explained as follows. When a trial starts there is no pheromone on the two branches. Hence, the ants do not have a preference and they select with the same probability either of the two branches. It can be expected that, on average, half of the ants choose the short branch and the other half the long branch, although stochastic oscillations may occasionally favor one branch over the other. However, because one branch is shorter than the other, the ants choosing the short branch are the first to reach the food and to start their travel back to the nest.<sup>5</sup> But then, when they must make a decision between the short and the long branch, the higher

---

<sup>4</sup> The experiment described was originally executed using a laboratory colony of Argentine ants (*Iridomyrmex humilis*). It is known that these ants deposit pheromone both when leaving and when returning to the nest [80].

<sup>5</sup> In the ACO literature, this is often called *differential path length effect*.

level of pheromone on the short branch biases their decision in its favor.<sup>6</sup> Therefore, pheromone starts to accumulate faster on the short branch, which will eventually be used by the great majority of the ants.

It should be clear by now how real ants have inspired AS and later algorithms: the double bridge was substituted by a graph, and pheromone trails by artificial pheromone trails. Also, because we wanted artificial ants to solve problems more complicated than those solved by real ants, we gave artificial ants some extra capacities, like a memory (used to implement constraints and to allow the ants to retrace their solutions without errors) and the capacity for depositing a quantity of pheromone proportional to the quality of the solution produced (a similar behavior is observed also in some real ant species in which the quantity of pheromone deposited while returning to the nest from a food source is proportional to the quality of the food source [9]).

In Section 8.4.2 we will see how, starting from AS, new algorithms have been proposed that, although retaining some of the original biological inspiration, are less and less biologically inspired and more and more motivated by the need of making ACO algorithms competitive with other state-of-the-art algorithms. Nevertheless, many aspects of the original Ant System remain: the need for a colony, the role of autocatalysis, the cooperative behavior mediated by artificial pheromone trails, the probabilistic construction of solutions biased by artificial pheromone trails and local heuristic information, the pheromone updating guided by solution quality, and the evaporation of pheromone trails are present in all ACO algorithms.

#### 8.4.2 Historical Development

As said, AS was the first ACO algorithm to be proposed in the literature. In fact, AS was originally a set of three algorithms called *ant-cycle*, *ant-density*, and *ant-quantity*. These three algorithms were proposed in Dorigo’s doctoral dissertation [55] and first appeared in a technical report [63, 64] that was published a few years later in the IEEE Transactions on Systems, Man, and Cybernetics [65]. Other early publications are [34, 35].

While in *ant-density* and *ant-quantity* the ants updated the pheromone directly after a move from a city to an adjacent one, in *ant-cycle* the pheromone update was only done after all the ants had constructed the tours and the amount of pheromone deposited by each ant was set to be a function of the tour quality. Because *ant-cycle* performed better than the other two variants, it was later called simply Ant System (and in fact, it is the algorithm that we will present in Section 8.4.2.1), while the other two algorithms were no longer studied.

The major merit of AS, whose computational results were promising but not competitive with other more established approaches, was to stimulate a number of

---

<sup>6</sup> A process like this, in which a decision taken at time  $t$  increases the probability of making the same decision at time  $T > t$  is said to be an *autocatalytic* process. Autocatalytic processes exploit *positive feedback*.

researchers to develop extensions and improvements of its basic ideas so as to produce better performing, and often state-of-the-art, algorithms.

### 8.4.2.1 The First ACO Algorithm: Ant System and the TSP

The TSP is a paradigmatic  $\mathcal{NP}$ -hard combinatorial optimization problem that has attracted an enormous amount of research effort [6, 94, 99]. The TSP is a very important problem also in the context of Ant Colony Optimization because it is the problem to which the original AS was first applied [55, 63–65], and it has later often been used as a benchmark to test new ideas and algorithmic variants.

In AS each ant is initially put on a randomly chosen city and has a memory, in which the partial solution it has constructed so far is stored (initially the memory contains only the start city). Starting from its start city, an ant iteratively moves from city to city, which corresponds to adding iteratively solution components as explained in Section 8.3.2. When being at a city  $i$ , an ant  $k$  chooses to go to an as yet unvisited city  $j$  with a probability given by Equation (8.1). The heuristic information is given by  $\eta_{ij} = 1/d_{ij}$  and  $N(s_p)$  is the set of cities that ant  $k$  has not yet visited.

The solution construction ends after each ant has completed a tour, that is, after each ant has constructed a sequence of length  $n$ , corresponding to a permutation of the city indices. Next, the pheromone trails are updated. In AS this is done by using Equation (8.2), where we have

$$S_{\text{upd}} = S_{\text{iter}} \quad (8.3)$$

and

$$g(s) = 1/f(s), \quad (8.4)$$

where  $f(s)$  is the length of the tour  $s$ . Hence, the shorter the ant's tour is, the more pheromone is received by edges (solution components) belonging to the tour.<sup>7</sup> In general, edges which are used by many ants and which are contained in shorter tours will receive more pheromone and therefore are also more likely to be chosen in future iterations of the algorithm.

### 8.4.2.2 Ant System and Its Extensions

As previously stated, AS was not competitive with state-of-the-art algorithms for the TSP. Researchers then started to extend it to try to improve its performance.

A first improvement, called the *elitist strategy*, was introduced in [55, 65]. It consists in giving the best tour since the start of the algorithm (called  $s_{\text{gb}}$ ) a strong additional weight. In practice, each time the pheromone trails are updated

---

<sup>7</sup> Note that when applied to symmetric TSPs the edges are considered to be bidirectional and edges  $(i, j)$  and  $(j, i)$  are both updated. This is different for the ATSP, where edges are directed; in this case an ant crossing edge  $(i, j)$  will update only this edge and not edge  $(j, i)$ .

by Equation (8.2), we have that  $S_{\text{upd}} = S_{\text{iter}} \cup \{s_{\text{gb}}\}$  and that  $g(s), s \neq s_{\text{gb}}$ , is given by Equation (8.4). For  $s_{\text{gb}}$  we have that  $g(s_{\text{gb}}) = e/f(s_{\text{gb}})$ , where  $e$  is a positive integer. Note that this type of pheromone update is a first example of daemon action as described in Section 8.3.2.

Other improvements were rank-based Ant System (AS<sub>rank</sub>), MAX-MIN Ant System (MMAS), and Ant Colony System (ACS). AS<sub>rank</sub> [30] is in a sense an extension of the elitist strategy: it sorts the ants according to the lengths of the tours they generated and, after each tour construction phase, only the  $(w - 1)$  best ants and the global-best ant are allowed to deposit pheromone. The  $r$ th best ant of the colony contributes to the pheromone update with a weight given by  $\max\{0, w - r\}$  while the global-best tour reinforces the pheromone trails with weight  $w$ . This can easily be implemented by an appropriate choice of  $S_{\text{upd}}$  and  $g(s)$  in Equation (8.2).

MMAS [144, 147, 148] introduces upper and lower bounds to the values of the pheromone trails, as well as a different initialization of their values. In practice, in MMAS the allowed range of the pheromone trail strength is limited to the interval  $[\tau_{\min}, \tau_{\max}]$ , that is,  $\tau_{\min} \leq \tau_{ij} \leq \tau_{\max} \forall \tau_{ij}$ , and the pheromone trails are initialized to the upper trail limit, which causes a higher exploration at the start of the algorithm. In [144, 148] it is discussed how to set the upper and lower pheromone trail limits in a principled way. Pheromone updates are performed using a strong elitist strategy: only the best solution generated is allowed to update pheromone trails. This can be the *iteration-best* solution, that is, the best in the current iteration, or the *global-best* solution. The amount of pheromone deposited is then given by  $g(s_b) = 1/f(s_b)$ , where  $s_b$  is either  $s_{\text{ib}}$ , the iteration-best solution, or  $s_{\text{gb}}$ . In fact, in MMAS the iteration-best ant and the global-best ant can be used alternately in the pheromone update. Computational results have shown that best results are obtained when pheromone updates are performed using the global-best solution with increasing frequency during the algorithm execution [144, 148]. As an additional means for increasing the explorative behavior of MMAS, occasional pheromone trail reinitialization is used. MMAS has been improved also by the addition of local search routines that take the solution generated by ants to their local optimum just before the pheromone update.

ACS [60, 61, 75] improves over AS by increasing the importance of exploitation of information collected by previous ants with respect to exploration of the search space.<sup>8</sup> This is achieved via two mechanisms. First, a strong elitist strategy is used to update pheromone trails. Second, ants choose a solution component (that is, the next city in the TSP case) using the so-called pseudo-random proportional rule [61]: with probability  $q_0$ ,  $0 \leq q_0 < 1$ , they move to the city  $j$  for which the product between pheromone trail and heuristic information is maximum, that is,  $j = \arg \max_{c_i^j \in N(s_p)} \left\{ \tau_{ij} \cdot \eta_{ij}^\beta \right\}$ , while with probability  $1 - q_0$  they operate a biased exploration in which the probability  $p_{ij}(t)$  is the same as in AS (see Equation (8.1)).

---

<sup>8</sup> ACS was an offspring of Ant-Q [74], an algorithm intended to create a link between reinforcement learning [149] and Ant Colony Optimization. Computational experiments have shown that some aspects of Ant-Q, in particular the pheromone update rule, could be strongly simplified without affecting performance. It is for this reason that Ant-Q was abandoned in favor of the simpler and equally good ACS.

The value  $q_0$  is a parameter: when it is set to a value close to 1, as it is the case in most ACS applications, exploitation is favored over exploration. Obviously, when  $q_0 = 0$  the probabilistic decision rule becomes the same as in AS.

Also, as in *MMAS*, in ACS only the best ant (the global-best or the iteration-best ant) is allowed to add pheromone after each algorithm iteration; the former is the most common choice in applications of ACS. The amount of pheromone deposited is then given by  $g(s_b) = \rho / f(s_{gb})$ , where  $\rho$  is the pheromone evaporation.

Finally, ACS also differs from most ACO algorithms because ants update the pheromone trails while building solutions (as it was the case in *ant-quantity* and in *ant-density*, see [55, 63, 64]). In practice, ACS ants “eat” some of the pheromone trail on the edges they visit. This has the effect of decreasing the probability that the same path is used by all ants (that is, it favors exploration, counterbalancing this way the other two above-mentioned modifications that strongly favor exploitation of the collected knowledge about the problem). Similar to *MMAS*, ACS also usually exploits local search to improve its performance.

One could continue by enumerating the modifications that have been proposed in various other ACO algorithms that have been reported in the literature. Instead, we refer the interested reader to [66] and here we just give a brief overview of the various developments on ACO algorithms for  $\mathcal{NP}$ -hard problems. In Table 8.1 we give for each of the main ACO variants that have been proposed, the main references to these algorithms, the year in which they have been proposed, and whether they have been tested on the TSP. In fact (published) tests of most ACO variants have been done on the TSP, which again confirms the central role of this problem in ACO research.

**Table 8.1** Overview of the main ACO algorithms for  $\mathcal{NP}$ -hard problems that have been proposed in the literature. Given are the ACO algorithm name, the main references where these algorithms are described, the year they first have been published, and whether the corresponding algorithms have been tested on the TSP.

ACO algorithm	Main references	Year	TSP
Ant System	[55, 63, 65]	1991	Yes
Elitist AS	[55, 63, 65]	1992	Yes
Ant-Q	[74]	1995	Yes
Ant Colony System	[60, 61, 75]	1996	Yes
MMAS	[146–148]	1996	Yes
Rank-based AS	[29, 30]	1997	Yes
ANTS	[104, 105]	1998	No
Best-worst AS	[36, 37]	2000	Yes
Population-based ACO	[82]	2002	Yes
Beam-ACO	[17, 18]	2004	No

#### 8.4.2.3 Applications to Dynamic Network Routing Problems

The application of ACO algorithms to dynamic problems, that is, problems whose characteristics change while being solved, is among the main success stories in ACO. The first such application [131] concerned routing in circuit-switched

networks (e.g., classical telephone networks). The proposed algorithm, called ABC, was demonstrated on a simulated version of the British Telecom network. The main merit of ABC was to stimulate the interest of ACO researchers in dynamic problems. In fact, only rather limited comparisons were made between ABC and state-of-the-art algorithms, so that it is not possible to judge on the quality of the results obtained.

A very successful application of ACO to dynamic problems is the AntNet algorithm, proposed by Di Caro and Dorigo [45–48] and discussed in Section 8.5.3. AntNet was applied to routing in packet-switched networks. It contains a number of innovations with respect to AS and it has been shown experimentally to outperform a whole set of state-of-the-art algorithms on numerous benchmark problems. Later, AntNet has also been extended to routing problems in mobile ad hoc networks, obtaining again excellent performance [68].

#### 8.4.2.4 Toward the ACO Metaheuristic

Given the initial success of ACO algorithms in the applications to  $\mathcal{NP}$ -hard problems as well as to dynamic routing problems in networks, Dorigo and Di Caro [57] made the synthesis effort that led to the definition of a first version of the ACO metaheuristic (see also [57, 59, 66]). In other words, the ACO metaheuristic was defined a posteriori with the goal of providing a common characterization of a new class of algorithms and a reference framework for the design of new instances of ACO algorithms.

The first version of the ACO metaheuristic was aimed at giving a comprehensive framework for ACO algorithm to “classical”  $\mathcal{NP}$ -hard problems *and* for applications to highly dynamic problems in network routing applications. As such, this early version of the ACO metaheuristic left a lot of freedom to the algorithm designer in the definition of solution components, of the construction mechanism, of the pheromone update, and of the ants’ behavior. This more comprehensive variant of the ACO metaheuristic is presented in many publications on this topic [57, 59, 66]. Differently, the version of the ACO metaheuristic described in Section 8.3 is targeted toward the application of ACO algorithms to  $\mathcal{NP}$ -hard problems and therefore it is also more precise with respect to the definition of solution components and of the solution construction procedure. It follows mainly the versions presented in chapter 3 of [66] or in [21, 22].

## 8.5 Applications

The versatility and the practical use of the ACO metaheuristic for the solution of combinatorial optimization problems is best illustrated via example applications to a number of different problems.

The ACO application to the TSP has already been illustrated in Section 8.4.2.4. Here, we additionally discuss applications to two  $\mathcal{NP}$ -hard optimization problems:

the single machine total weighted tardiness problem (SMTWTP) and the set covering problem (SCP). We have chosen these problems since they are in several aspects different from the TSP. Although the SMTWTP is also a permutation problem, it differs from the TSP in the interpretation of the permutations. In the SCP a solution is represented as a subset of the available solution entities.

Applications of ACO to dynamic problems focus mainly on routing in data networks. To give a flavor of these applications, as a third example, we present the AntNet algorithm [47].

### 8.5.1 Example 1: The Single Machine Total Weighted Tardiness Scheduling Problem (SMTWTP)

In the SMTWTP  $n$  jobs have to be processed sequentially without interruption on a single machine. Each job has an associated processing time  $p_j$ , a weight  $w_j$ , and a due date  $d_j$  and all jobs are available for processing at time 0. The tardiness of job  $j$  is defined as  $T_j = \max\{0, C_j - d_j\}$ , where  $C_j$  is its completion time in the current job sequence. The goal in the SMTWTP is to find a job sequence that minimizes the sum of the weighted tardiness given by  $\sum_{i=1}^n w_i \cdot T_i$ .

For the ACO application to the SMTWTP, we can have for each position  $i$  in the sequence one variable  $X_i$  and each variable has  $n$  associated values  $j = 1, \dots, n$ . The solution components model the assignment of a job  $j$  to position  $i$  in the sequence.

The SMTWTP was tackled in [42] using ACS (ACS-SMTWTP). In ACS-SMTWTP, the positions of the sequence are filled in their canonical order, that is, first position one, next position two, and so on, until position  $n$ . At each construction step, an ant assigns a job to the current position using the *pseudo-random-proportional* action choice rule, where the feasible neighborhood of an ant is the list of yet unscheduled jobs. Pheromone trails are therefore defined as follows:  $\tau_{ij}$  refers to the desirability of scheduling job  $j$  at position  $i$ . This definition of the pheromone trails is, in fact, used in many ACO applications to scheduling problems [8, 42, 110, 142]. Concerning the heuristic information, the use of three priority rules allowed to define three different types of heuristic information for the SMTWTP [42]. The investigated priority rules were (i) the earliest due date rule, that puts the jobs in non-decreasing order of the due dates  $d_j$ ; (ii) the modified due date rule, that puts the jobs in non-decreasing order of the modified due dates given by  $mdd_j = \max\{C + p_j, d_j\}$  [8], where  $C$  is the sum of the processing times of the already sequenced jobs; and (iii) the apparent urgency rule, that puts the jobs in non-decreasing order of the apparent urgency [118], given by  $au_j = (w_j/p_j) \cdot \exp(-(\max\{d_j - C_j, 0\})/k\bar{p})$ , where  $k$  is a parameter of the priority rule. In each case, the heuristic information was defined as  $\eta_{ij} = 1/h_j$ , where  $h_j$  is either  $d_j$ ,  $mdd_j$ , or  $au_j$ , depending on the priority rule used.

The global and the local pheromone updates are carried out as in the standard ACS described in Section 8.4.2, where in the global pheromone update  $g(s_{gb})$  is the total weighted tardiness of the global-best solution.

In [42], ACS-SMTWTP was combined with a powerful local search algorithm. The final ACS algorithm was tested on a benchmark set available from ORLIB at <http://people.brunel.ac.uk/~mastjeb/jeb/info.html>. Within the computation time limits given,<sup>9</sup> ACS reached a very good performance and could find in each single run the optimal or best-known solutions on all instances of the benchmark set [42].

### 8.5.2 Example 2: The Set Covering Problem (SCP)

In the set covering problem (SCP) we are given a finite set  $A = \{a_1, \dots, a_n\}$  of elements and a set  $B = \{B_1, \dots, B_l\}$  of subsets,  $B_i \subseteq A$ , that covers  $A$ , that is, we have  $\bigcup_{i=1}^l B_i = A$ . We say that a set  $B_i$  covers an element  $a_j$ , if  $a_j \in B_i$ . Each set  $B_i$  has an associated cost  $c_i$ . The goal in the SCP is to choose a subset  $C$  of the sets in  $B$  such that (i) every element of  $A$  is covered and that (ii)  $C$  has minimum total cost, that is, the sum of the costs of the subsets in  $C$  is minimal.

ACO can be applied in a very straightforward way to the SCP. A binary variable  $X_i$  is associated with every set  $B_i$  and a solution component  $c_i^1$  indicates that  $B_i$  is selected for set  $C$  (that is,  $X_i = 1$ ), while a solution component  $c_i^0$  indicates it is not selected (that is,  $X_i = 0$ ). Each solution component  $c_i^1$  is associated with a pheromone trail  $\tau_i$  and a heuristic information  $\eta_i$  that indicates the learned and the heuristic desirability of choosing subset  $B_i$ . (Note that no pheromone trails are associated with solution components  $c_i^0$ .) Solutions can be constructed as follows. Each ant starts with an empty solution and then adds at each step one subset until a cover is completed. A solution component  $c_i^1$  is chosen with probability

$$p_i(s_p) = \frac{\tau_i^\alpha \cdot [\eta_i(s_p)]^\beta}{\sum_{l \in N(s_p)} \tau_l^\alpha \cdot [\eta_l(s_p)]^\beta}, \quad \forall c_i^1 \in N(s_p), \quad (8.5)$$

where  $N(s_p)$  consists of all subsets that cover at least one still uncovered element of  $A$ . The heuristic information  $\eta_i(s_p)$  can be chosen in several different ways. For example, a simple static information could be used, taking into account only the subset cost:  $\eta_i = 1/c_i$ . A more sophisticated approach would be to consider the total number of elements  $d_i$  covered by a set  $B_i$  and to set  $\eta_i = d_i/c_i$ . These two ways of defining the heuristic information do not depend on the partial solution. Typically, more accurate heuristics can be developed taking into account the partial solution of an ant. In this case, it can be defined as  $\eta_i(s_p) = e_i(s_p)/c_i$ , where  $e_i(s_p)$  is the so-called cover value, that is, the number of additional elements covered when adding subset  $B_i$  to the current partial solution  $s_p$ . In other words, the heuristic information measures the unit cost of covering one additional element.

An ant ends the solution construction when all the elements of  $A$  are covered. In a post-optimization step, an ant can remove redundant subsets—subsets that only cover elements that are already covered by other subsets in the final solution—or

---

<sup>9</sup> The maximum time for the largest instances was 20 min on a 450 MHz Pentium III PC with 256 MB RAM. Programs were written in C++ and the PC was run under Red Hat Linux 6.1.

apply some additional local search to improve solutions. The pheromone update can be carried out in a standard way as described in earlier sections.

When applying ACO to the SCP, one difference with the previously presented applications is that the number of solution components in the ant's solutions may differ among the ants and, hence, the solution construction only ends when all the ants have terminated their corresponding walks.

There have been a few applications of ACO algorithms to the SCP [4, 90, 101]. The best results of these ACO algorithms are obtained by the variants that are tested by Lessing et al. [101]. In their article, they compared the performance of a number of ACO algorithms with and without the usage of a local search algorithm based on 3-flip neighborhoods [156]. The best performance results were obtained, as expected, when including local search and for a large number of instances the computational results were competitive with state-of-the-art algorithms for the SCP.

### 8.5.3 Example 3: AntNet for Network Routing Applications

Given a graph representing a telecommunications network, the problem solved by AntNet is to find the minimum cost path between each pair of vertices of the graph. It is important to note that, although finding a minimum cost path on a graph is an easy problem (it can be efficiently solved by algorithms having polynomial worst case complexity [12]), it becomes extremely difficult when the costs on the edges are time-varying stochastic variables. This is the case of routing in packet-switched networks, the target application for AntNet.

Here we briefly describe a simplified version of AntNet (the interested reader should refer to [47], where the AntNet approach to routing is explained and evaluated in detail). As stated earlier, in AntNet each ant searches for a minimum cost path between a given pair of vertices of the network. To this end, ants are launched from each network vertex toward randomly selected destination vertices. Each ant has a source vertex  $s$  and a destination vertex  $d$  and moves from  $s$  to  $d$  hopping from one vertex to the next until vertex  $d$  is reached. When ant  $k$  is in vertex  $i$ , it chooses the next vertex  $j$  to move to according to a probabilistic decision rule which is a function of the ant's memory and of local pheromone and heuristic information (very much like AS, for example).

Unlike AS, where pheromone trails are associated with edges, in AntNet pheromone trails are associated with edge-destination pairs. That is, each directed edge  $(i, j)$  has  $n - 1$  associated trail values  $\tau_{ijd} \in [0, 1]$ , where  $n$  is the number of vertices in the graph associated with the routing problem. In other words, there is one trail value  $\tau_{ijd}$  for each possible destination vertex  $d$  an ant located in vertex  $i$  can have. In general, it will hold that  $\tau_{ijd} \neq \tau_{jid}$ . Each edge also has an associated heuristic value  $\eta_{ij} \in [0, 1]$  independent of the final destination. The heuristic values can be set, for example, to the values  $\eta_{ij} = 1 - q_{ij} / \sum_{l \in N_i} q_{il}$ , where  $q_{ij}$  is the length (in bits waiting to be sent) of the queue of the link connecting vertex  $i$  with its neighbor  $j$ : links with a shorter queue have a higher heuristic value  $N_i$  is the set of neighbors to vertex  $i$ .

Ants choose their way probabilistically, using as probability a functional composition of the local pheromone trails  $\tau_{ijd}$  and heuristic values  $\eta_{ij}$ . While building the path to their destinations, ants move using the same link queues as data and experience the same delays as data packets. Therefore, the time  $T_{sd}$  elapsed while moving from the source vertex  $s$  to the destination vertex  $d$  can be used as a measure of the quality of the path they built. The overall quality of a path is evaluated by a heuristic function of the trip time  $T_{sd}$  and of a local adaptive statistical model maintained in each vertex. In fact, paths need to be evaluated relative to the network status because a trip time  $T$  judged of low quality under low congestion conditions could be an excellent one under high traffic load. Once the generic ant  $k$  has completed a path, it deposits on the visited vertices an amount of pheromone  $\Delta\tau^k(t)$  proportional to the quality of the path. To deposit pheromone, after reaching its destination vertex, the ant moves back to its source vertex along the same path but backward and using high priority queues, to allow a fast propagation of the collected information. The pheromone trail intensity of each edge  $l_{ij}$  used by the ant while it was moving from  $s$  to  $d$  is increased as follows:  $\tau_{ijd}(t) \leftarrow \tau_{ijd}(t) + \Delta\tau^k(t)$ . After the pheromone trail on the visited edges has been updated, the pheromone value of all the outgoing connections of the same vertex  $i$ , relative to destination  $d$ , evaporates in such a way that the pheromone values are normalized and can continue to be used as probabilities:  $\tau_{ijd}(t) \leftarrow \tau_{ijd}(t)/(1 + \Delta\tau^k(t))$ ,  $\forall j \in N_i$ .

AntNet was compared with many state-of-the-art algorithms on a large set of benchmark problems under a variety of traffic conditions. It always compared favorably with competing approaches and it was shown to be very robust with respect to varying traffic conditions and parameter settings. More details on the experimental results can be found in [47].

### 8.5.4 Applications of the ACO Metaheuristic

ACO has raised a lot of interest in the scientific community. There are now hundreds of successful implementations of the ACO metaheuristic applied to a wide range of different combinatorial optimization problems. The vast majority of these applications concern  $\mathcal{NP}$ -hard combinatorial optimization problems.

Many successful ACO applications to  $\mathcal{NP}$ -hard problems use local search algorithms to improve the ants' solutions. Another common feature of most successful ACO applications is that they use one of the advanced ACO algorithms such as ACS and MMAS. In fact, AS has been abandoned by now in favor of more performing variants. Finally, for problems for which ACO algorithms reach very high performance, the available ACO algorithms are fine-tuned to the problem under consideration. Apart from fine-tuning parameter settings, this typically involves the exploitation of problem knowledge, for example, through the use of appropriate heuristic information, informed choices for the construction mechanism, or the use of fine-tuned local search algorithms. An overview of some successful applications of ACO algorithms to challenging “static” optimization problems is given in Table 8.2. In fact, this overview is very incomplete since the currently available ACO applications

**Table 8.2** Some current applications of ACO algorithms. Applications are listed according to problem types.

<i>Problem type</i>	<i>Problem name</i>	<i>Authors</i>	<i>Year</i>	<i>References</i>
Routing	Traveling salesman	Dorigo et al. Dorigo and Gambardella Stützle and Hoos	1991, 1996 1997 1997, 2000	[64, 65] [61] [147, 148]
	TSP with time windows	López Ibáñez et al.	2009	[102]
	Sequential ordering	Gambardella and Dorigo	2000	[76]
	Vehicle routing	Gambardella et al. Reimann et al. Favoretto et al. Fuellerer et al.	1999 2004 2007 2009	[77] [127] [72] [73]
	Multicasting	Hernández and Blum	2009	[91]
	Quadratic assignment	Maniezzo	1999	[105]
	Frequency assignment	Stützle and Hoos Maniezzo and Carbonaro	2000 2000	[148] [106]
	Course timetabling	Socha et al.	2002, 2003	[138, 139]
	Graph coloring	Costa and Hertz	1997	[39]
	Project scheduling	Merkle et al.	2002	[111]
Scheduling	Weighted tardiness	den Besten et al.	2000	[42]
	Flow shop	Merkle and Middendorf Stützle	2000 1997	[109] [142]
	Open shop	Rajendran, Ziegler	2004	[125]
	Car sequencing	Blum	2005	[18]
		Solnon	2008	[140]
Subset	Set covering	Lessing et al.	2004	[101]
	<i>l</i> -cardinality trees	Blum and Blesa	2005	[20]
	Multiple knapsack	Leguizamón and Michalewicz	1999	[100]
	Maximum clique	Solnon and Fenet	2006	[141]
Machine learning	Classification rules	Parpinelli et al.	2002	[124]
		Martens et al.	2006	[107]
		Otero et al.	2008	[121]
	Bayesian networks	Campos, Fernández-Luna	2002	[40, 41]
Bioinformatics	Neural networks	Socha, Blum	2007	[135]
	Protein folding	Shmygelska and Hoos	2005	[132]
	Docking	Korb et al.	2006	[97, 98]
	DNA sequencing	Blum et al.	2008	[25]
	Haplotype inference	Benedettini et al.	2008	[11]

go into the hundreds. For a more complete overview (covering, however, only applications until the year 2004), we refer to [66].

Another large class of applications of ACO algorithms is routing problems where some system properties such as the availability of links or the cost of traversing links are time varying. This is a common case in telecommunications networks. As said before, the first ACO applications have been to telephone-like networks [131], which are circuit-switched, and to packet-switched networks such as the Internet [47].

Ant-based algorithms have given rise to several other routing algorithms, enhancing performance in a variety of wired network scenarios, see [44, 133] for a survey.

More recent applications of these strategies involved the more challenging class of mobile ad hoc networks (MANETs). Unfortunately, the straightforward application of the ACO algorithms developed for wired networks has proven unsuccessful due to the specific characteristics of MANETs (very fast dynamics, link asymmetry) [159]. An ACO algorithm that is competitive with state-of-the-art routing algorithms for MANETs, while at the same time offering better scalability, has been proposed by Ducatelle et al. [49, 68]. For an exhaustive list of references on ACO applications for dynamic network routing problems we refer to [69, 71].

### **8.5.5 Main Application Principles**

ACO algorithms have been applied to a large number of different combinatorial optimization problems. Based on this experience, one can identify some basic issues that need to be addressed when attacking a new problem. These issues are discussed in the following.

#### **8.5.5.1 Definition of Solution Components and Pheromone Trails**

Of crucial importance in ACO applications is the definition of the solution components and of the pheromone model. Consider, for example, the differences in the definition of solution components in the TSP and the SMTWTP. Although both problems represent solutions as permutations, the definition of solution components (and, hence, the interpretation of the pheromone trails), is very different. In the TSP case, a solution component refers to the direct successor relationship between elements, while in the SMTWTP it refers to the allocation of a job to a specific position in the permutation. This is intuitively due to the different role that permutations have in the two problems. In the TSP, only the relative order of the solution components is important and a permutation  $\pi = (1, 2, \dots, n)$  has the same tour length as the permutation  $\pi' = (n, 1, 2, \dots, n - 1)$ —it represents the same tour. On the contrary, in the SMTWTP (as well as in many other scheduling problems),  $\pi$  and  $\pi'$  would represent two different solutions with most probably very different costs; in this case the position information is very important.

In some applications, the role of the pheromone trail definition has been investigated in more depth. Blum and Sampels compare different ways of defining the pheromone model for shop scheduling problems [23]. In [22], Blum and Dorigo show that the choice of an inappropriate pheromone model can result in an undesirable performance degradation over time. Fortunately, in many applications the solution components used in high-performing constructive algorithms, together with the correct choice of the pheromone model, typically result in high-performing algorithms. However, finding the best pheromone model is not always a straightforward task. Examples of some more complex or unusual choices are the ACO application to the shortest common supersequence problem [114] or the application of ACO to protein–ligand docking [98].

### 8.5.5.2 Balancing Exploration and Exploitation

Any effective metaheuristic algorithm has to achieve an appropriate balance between the exploitation of the search experience gathered so far and the exploration of unvisited or relatively unexplored search space regions. In ACO, several ways exist for achieving such a balance, typically through the management of the pheromone trails. In fact, the pheromone trails induce a probability distribution over the search space and determine which parts of the search space are effectively sampled, that is, in which part of the search space the constructed solutions are located with higher frequency.

The best performing ACO algorithms typically use an *elitist strategy* in which the best solutions found during the search contribute strongly to pheromone trail updating. A stronger exploitation of the “learned” pheromone trails can be achieved during solution construction by applying the pseudo-random proportional rule of ACS, as explained in Section 8.4.2.2. These exploitation features are then typically combined with some means to ensure enough search space exploration trying to avoid convergence of the ants to a single path, corresponding to a situation of search stagnation. There are several ways to try to avoid such stagnation situations. For example, in ACS the ants use a local pheromone update rule during solution construction to make the path they have taken less desirable for subsequent ants and, thus, to diversify the search. MMAS introduces an explicit lower limit on the pheromone trail value so that a minimal level of exploration is always guaranteed. MMAS also uses a reinitialization of the pheromone trails, which is a way of enforcing search space exploration. Finally, an important role in the balance of exploration and exploitation is played by the parameters  $\alpha$  and  $\beta$  in Equation (8.1). Consider, for example, the influence of parameter  $\alpha$ . (Parameter  $\beta$  has an analogous influence on the exploitation of the heuristic information.) For  $\alpha > 0$ , the larger the value of  $\alpha$  the stronger the exploitation of the search experience; for  $\alpha = 0$  the pheromone trails are not taken into account at all; and for  $\alpha < 0$  the most probable choices taken by the ants are those that are less desirable from the point of view of pheromone trails. Hence, varying  $\alpha$  could be used to shift from exploration to exploitation and conversely.

### 8.5.5.3 ACO and Local Search

In many applications to  $\mathcal{NP}$ -hard combinatorial optimization problems, ACO algorithms perform best when coupled with local search algorithms. Local search algorithms locally optimize the ants’ solutions and these locally optimized solutions are used in the pheromone update.

The use of local search in ACO algorithms can be very interesting as the two approaches are complementary. In fact, ACO algorithms perform a rather coarse-grained search, and the solutions they produce can then be locally fine-tuned by an adequate local search algorithm. On the other side, generating appropriate

initial solutions for local search algorithms is not an easy task. In practice, ants probabilistically combine solution components which are part of the best locally optimal solutions found so far and generate new, promising initial solutions for the local search. Experimentally, it has been found that such a combination of a probabilistic, adaptive construction heuristic with local search can yield excellent results [26, 61, 147].

Despite the fact that the use of local search algorithms has been shown to be crucial for achieving state-of-the-art performance in many ACO applications, it should be noted that ACO algorithms also show very good performance when local search algorithms cannot be applied easily [47, 114].

#### 8.5.5.4 Heuristic Information

The possibility of using heuristic information to direct the ants' probabilistic solution construction is important because it gives the possibility of exploiting problem-specific knowledge. This knowledge can be available *a priori* (this is the most frequent situation in  $\mathcal{NP}$ -hard problems) or at runtime (this is the typical situation in dynamic problems).

For most  $\mathcal{NP}$ -hard problems, the heuristic information  $\eta$  can be computed at initialization time and then it remains the same throughout the whole algorithm's run. An example is the use, in the TSP applications, of the length  $d_{ij}$  of the edge connecting cities  $i$  and  $j$  to define the heuristic information  $\eta_{ij} = 1/d_{ij}$ . However, the heuristic information may also depend on the partial solution constructed so far and therefore be computed at each step of an ant's solution construction. This determines a higher computational cost that may be compensated by the higher accuracy of the computed heuristic values. For example, in the ACO applications to the SMTWTP and the SCP the use of such "adaptive" heuristic information was found to be crucial for reaching very high performance.

Finally, it should be noted that while the use of heuristic information is rather important for a generic ACO algorithm, its importance is strongly reduced if local search is used to improve solutions. This is due to the fact that local search takes into account information about the cost to improve solutions in a more direct way.

## 8.6 Developments

In this section, we review recent research trends in ACO. These include (i) the application of ACO algorithms to non-standard problems; (ii) the development of ACO algorithms that are hybridized with other metaheuristics or techniques from mathematical programming; (iii) the parallel implementation of ACO algorithms; and (iv) theoretical results on ACO algorithms.

### 8.6.1 Non-standard Applications of ACO

We review here applications of ACO to problems that involve complicating factors such as multiple objective functions, time-varying data, and stochastic information

about objective values or constraints. In addition, we review some recent applications of ACO to continuous optimization problems.

### 8.6.1.1 Multiobjective Optimization

Frequently, in real-world applications, various solutions are evaluated as a function of multiple, often conflicting objectives. In simple cases, objectives can be ordered with respect to their importance or they can be combined into a single-objective by using a weighted sum approach. An example of the former approach is the application of a two-colony ACS algorithm for the vehicle routing problem with time windows [77]; an example of the latter is given by Doerner et al. [51] for a bi-objective transportation problem.

If a priori preferences or weights are not available, the usual option is to approximate the set of Pareto-optimal solutions—a solution  $s$  is Pareto optimal if no other solution has a better value than  $s$  for at least one objective and is not worse than  $s$  for the remaining objectives. The first general ACO approach targeted to such problems is due to Iredi et al. [93], who discussed various alternatives to apply ACO to multiobjective problems and presented results with a few variants for a bi-objective scheduling problem. Since then, several algorithmic studies have tested various alternative approaches. These possible approaches differ in whether they use one or several pheromone matrices (one for each objective), one or several heuristic information, how solutions are chosen for pheromone deposit, and whether one or several colonies of ants are used. Several combinations of these possibilities have been studied, for example, in [3, 92]. For a detailed overview of the available multiobjective ACO algorithms we refer to the review articles by García-Martínez [78], that also contains an experimental evaluation of some proposed ACO approaches, and by Angus and Woodward [5].

### 8.6.1.2 Dynamic Versions of $\mathcal{NP}$ -Hard Problems

As said earlier, ACO algorithms have been applied with significant success to dynamic problems in the area of network routing [47, 49]. ACO algorithms have also been applied to dynamic versions of classical  $\mathcal{NP}$ -hard problems. Examples are the applications to dynamic versions of the TSP, where the distances between cities may change or where cities may appear or disappear [70, 81, 82]. More recently, applications of ACS to dynamic vehicle routing problems are reported in [54, 117], showing good results on academic instances of the problem as well as on real-world instances.

### 8.6.1.3 Stochastic Optimization Problems

In many optimization problems data are not known exactly before generating a solution. Rather, what is available is stochastic information on the objective function value(s), on the decision variable values, or on the constraint boundaries due to uncertainty, noise, approximation or other factors. ACO algorithms have been applied

to a few stochastic optimization problems. The first stochastic problem to which ACO was applied is the probabilistic TSP (PTSP), where for each city the probability that it requires a visit is known and the goal is to find an a priori tour of minimal expected length over all the cities. The first to apply ACO to the PTSP were Bianchi et al. [14], who used an adaptation of ACS. This algorithm was improved by Bräne and Guntzsch and, very recently, by Balaprakash et al. [7], resulting in a state-of-the-art algorithm for the PTSP. Other applications of ACO include vehicle routing problems with uncertain demands [13] and the selection of optimal screening policies for diabetic retinopathy [28]; the latter approach builds on the S-ACO algorithm proposed earlier by Gutjahr [85].

#### 8.6.1.4 Continuous Optimization

Although ACO was proposed for combinatorial problems, researchers started to adapt it to continuous optimization problems.<sup>10</sup> The simplest approach for applying ACO to continuous problems would be to discretize the real-valued domain of the variables. This approach has been successfully followed when applying ACO to the protein–ligand docking problem [98], where it was combined with a local search that was, however, working on the continuous domain of the variables. Recently, ACO algorithms that handle continuous parameters natively have been proposed. An example is the work of Socha and Dorigo [137], where the probability density functions that are implicitly built by the pheromone model are explicitly represented by Gaussian kernel functions. Their approach has also been extended to mixed-variable—continuous and discrete—problems [136]. Other references on this subject are [134, 151, 153].

### 8.6.2 Algorithmic Developments

In the early years of ACO research, the focus was in developing ACO variants that modify the pheromone update or the solution generation mechanism to improve the algorithmic performance. More recently, researchers have started to explore combinations of ACO with other algorithmic techniques. Here, we review some of the most noteworthy of these recent developments.

#### 8.6.2.1 Hybridizations of ACO with Other Metaheuristics

The most straightforward hybridization of ACO is with local improvement heuristics, which are used to fine-tune the solutions constructed by the ants. Often simple iterative improvement algorithms are used. However, in various researches other

---

<sup>10</sup> There have been several proposals of ant-inspired algorithms for continuous optimization [15, 67, 116]. However, these differ strongly from the underlying ideas of ACO (for example, they use direct communication among ants) and therefore cannot be considered as algorithms falling into the framework of the ACO metaheuristic.

metaheuristic algorithms have been used as improvement methods. One example is the use of tabu search to improve ants' solutions for the quadratic assignment problem [148, 150]. Interestingly, also more sophisticated hybridizations have been proposed. A first one is to let the ants start the solution construction not from scratch but from partial solutions that are obtained either by removing solution components from an ants' complete solution [155, 158] or by taking partial solutions from other complete solutions [1, 2, 152]. Two important advantages of starting the solution construction from partial solutions are that (i) the solution construction process is much faster and (ii) good parts of solutions may be exploited directly. Probably the most straightforward of these proposals is the *iterated ants* [155], which uses ideas from the iterated greedy (IG) metaheuristic [130]. Once some initial solution has been generated, IG iterates over construction heuristics by first removing solution components of a complete solution  $s$ , resulting in a partial solution  $s_p$ . From  $s_p$  a complete solution is then reconstructed using some construction mechanism. In the iterated ants algorithm, this mechanism is simply the standard solution construction of the underlying ACO algorithm. Computational results suggest that this idea is particularly useful if no effective local search is available.

### 8.6.2.2 Hybridizations of ACO with Branch-and-Bound Techniques

The integration of tree search techniques into constructive algorithms is an appealing possibility of hybridization since the probabilistic solution construction of ants can be seen as the stochastic exploration of a search tree. Particularly attractive are combinations of ACO with tree search techniques from mathematical programming such as branch-and-bound. A first algorithm is the approximate nondeterministic tree search (ANTS) algorithm by Maniezzo [105]. The most important innovation of ANTS is the use of lower bound estimates as the heuristic information for rating the attractiveness of adding specific solution components. Additionally, lower bound computations allow the method to prune feasible extensions of partial solutions if the estimated solution cost is larger than that of the best solution found so far. An additional innovation of ANTS consists in the use of an initial lower bound computation to influence the order in which solution components are considered in the solution construction. Computational results obtained with ANTS for the quadratic assignment and the frequency assignment problems are very promising [105, 106].

BeamACO, the combination of ACO algorithms with beam-search was proposed by Blum [18]. Beam-search is a derivative of branch-and-bound algorithms that keeps at each iteration a set of at most  $fw$  nodes in a search tree and expands each of them in at most  $bw$  directions according to a selection based on lower bounds [122]; this results in at most  $fw \cdot bw$  partial candidate solutions. After each step of extending partial solutions, the best  $fw$  partial solutions are kept (where best is rated with respect to a lower bound). BeamACO takes from beam-search the parallel exploration of the search tree and replaces beam-search's deterministic solution extension mechanism with that of ACO. The results with BeamACO have been very good so far.

For example, it is a state-of-the-art algorithm for open shop scheduling [18] and for some variants of assembly line balancing [19].

### 8.6.2.3 Combinations of ACO with Constraint Programming Techniques

For problems that are highly constrained and for which it is difficult to find feasible solutions, an attractive possibility is to integrate constraint programming techniques into ACO. A first proposal in this direction has been made by Meyer and Ernst [113]. In particular, they integrate a constraint propagation mechanism into the solution construction of the ants to identify earlier in the construction process whether specific solutions extensions would lead to infeasible solutions. Computational tests on a highly constrained scheduling problem showed the high potential of this approach. More recently, Khichane et al. [96] have examined the integration of an ACO algorithm into a constraint solver.

### 8.6.3 Parallel Implementations

The very nature of ACO algorithms lends them to be parallelized in the data or population domains. In particular, many parallel models used in other population-based algorithms can be easily adapted to ACO. Most parallelization strategies can be classified into *fine-grained* and *coarse-grained* strategies. Characteristics of fine-grained parallelization are that very few individuals are assigned to one single processor and that frequent information exchange among the processors takes place. On the contrary, in coarse-grained approaches larger subpopulations or even full populations are assigned to single processors and information exchange is rather rare. We refer, for example, to [32] for an overview.

Fine-grained parallelization schemes have been investigated with parallel versions of AS for the TSP on the Connection Machine CM-2 by attributing a single processing unit to each ant [27]. Experimental results showed that communication overhead can be a major problem with this approach on fine-grained parallel machines, since ants end up spending most of their time communicating to other ants the modifications they have made to pheromone trails. Similar negative results have also been reported in [31, 126].

As shown by several researches [27, 31, 50, 103, 115, 143], coarse-grained parallelization schemes are much more promising for ACO. When applied to ACO, coarse-grained schemes run  $p$  subcolonies in parallel, where  $p$  is the number of available processors. Subcolonies exchange information according to some policy that defines the kind of information to be exchanged, how migrants between the subcolonies are selected, to which colonies the information is sent, when information is sent, and what is to be done with the received information. Middendorf et al. [115] investigate different ways of exchanging solutions among ant colonies. They consider an exchange of the global best solutions among all colonies and local exchanges based on a virtual neighborhood among subcolonies

which corresponds to a directed ring. Their main observation was that the best solutions, with respect to computing time and solution quality, were obtained by limiting the information exchange to a local neighborhood of the colonies. Twomey et al. [154] study communication strategies for multi-colony ACO algorithms. They show that the cooperation of multiple homogenous colonies becomes less effective for increasing search effort and stronger local search algorithms. Doerner et al. [50] study different frequencies of information exchange using a fixed-frequency policy. Chen and Zhang [33] propose the use of variable frequency schedules for the migration of ants. In the extreme case, there is no communication among the subcolonies, resulting in parallel independent runs. This is the easiest way to parallelize randomized algorithms and can be very effective as shown in the computational results of Stützle [143] and of Manfrin et al. [103].

#### 8.6.4 Theoretical Results

The initial, experimentally driven research on ACO has established it as an interesting algorithmic technique. After this initial phase, researchers have started to obtain insights into fundamental properties of ACO algorithms.

The first question was whether an ACO algorithm, if given enough time, will eventually find an optimal solution. This is an interesting question, because the pheromone update could prevent ACO algorithms from ever reaching an optimum. The first convergence proofs were presented by Gutjahr in [83]. He proved convergence with probability  $1 - \varepsilon$  to the optimal solution of Graph-Based Ant System (GBAS), an ACO algorithm whose empirical performance is unknown. Later, he proved convergence to any optimal solution [84] with probability one for two extended versions of GBAS. Interestingly, convergence proofs for two of the top performing ACO algorithms in practice, ACS and MMAS, could also be obtained [66, 145].

Unfortunately, these convergence proofs do not say anything about the speed with which the algorithms converge to the optimal solution. Recently, a number of results on the expected runtime when applying ACO algorithms to specific problems have been obtained. The first results in this direction were obtained by Gutjahr [86] and since then a number of additional results have been established [53, 88, 89, 119, 120]. For a recent review of this research direction, we refer to [87].

Other research in ACO theory has focused on establishing formal links between ACO and other techniques for learning and optimization. Birattari et al. [16] relate ACO to the fields of optimal control and reinforcement learning; Meuleau and Dorigo [112] examine the connections between ACO algorithms and probabilistic learning algorithms such as the stochastic gradient ascent and the cross-entropy method; finally, Zlochin et al. [160] have proposed a unifying framework for so-called model-based search algorithms that allows a better understanding of what are the important parts of an ACO algorithm and that will help to a better cross-fertilization among algorithms.

While convergence proofs give insight into some mathematically relevant properties of algorithms, they usually do not provide guidance to practitioners for the implementation of efficient algorithms. More relevant for practical applications are research efforts that aim at a better understanding of the behavior of ACO algorithms. Blum and Dorigo [22] have shown that ACO algorithms in general suffer from *first-order deception* in the same way as genetic algorithms suffer from deception. They further introduced the concept of *second-order deception*, which occurs, for example, in situations where some solution components receive updates from more solutions on average than others they compete with [24]. The first to study the behavior of ACO algorithms by analyzing the dynamics of the pheromone model were Merkle and Middendorf [108]. For idealized permutation problems, they showed that the bias introduced on decisions in the construction process (due to constraints on the feasibility of solutions) leads to a bias which they call *selection bias*.

A review paper on recent advances in ACO theory is [56].

## 8.7 Conclusions

Since the proposal of the first ACO algorithms in 1991, the field of ACO has attracted a large number of researchers and nowadays a large number of research results of both experimental and theoretical nature exist. By now ACO is a well-established metaheuristic. The importance of ACO is exemplified by (i) the biannual conference ANTS (International conference on Ant Colony Optimization and Swarm Intelligence; <http://iridia.ulb.ac.be/~ants/>), where researchers meet to discuss the properties of ACO and other ant algorithms, both theoretically and experimentally; (ii) the IEEE Swarm Intelligence Symposium series; (iii) various conferences on metaheuristics and evolutionary algorithms, where ACO is a central topic; and (iv) a number of journal special issues [38, 52, 58, 62]. More information on ACO can also be found on the Ant Colony Optimization web page: [www.aco-metaheuristic.org](http://www.aco-metaheuristic.org). Additionally, a moderated mailing list dedicated to the exchange of information related to ACO is accessible at: [www.aco-metaheuristic.org/mailing-list.html](http://www.aco-metaheuristic.org/mailing-list.html).

The majority of the currently published articles on ACO are clearly on its application to computationally challenging problems. While most researches here are on academic applications, it is noteworthy that companies have started to use ACO algorithms for real-world applications [129]. For example, the company AntOptima ([www.antoptima.com](http://www.antoptima.com)) plays an important role in promoting the real-world application of ACO. In real-world applications, features such as time-varying data, multiple objectives, or the availability of stochastic information about events or data are rather common. Interestingly, applications of ACO to problems that show such characteristics are receiving increasing attention. In fact, we believe that ACO algorithms will show their greatest advantage when they will be systematically applied to such “ill-structured” problems for which it is not clear how to apply local search, or to highly dynamic domains where only local information is available.

**Acknowledgments** This work was supported by the META-X project, an *Action de Recherche Concertée* funded by the Scientific Research Directorate of the French Community of Belgium. Marco Dorigo and Thomas Stützle acknowledge support from the Belgian F.R.S.-FNRS, of which they are a Research Director and a Research Associate, respectively.

## References

1. Acan, A.: An external memory implementation in ant colony optimization. In: Dorigo, M., Birattari, M., Blum, C., Gambardella, L.M., Mondada, F., Stützle, T. (eds.) *Ant Colony Optimization and Swarm Intelligence: 4th International Workshop, ANTS 2004*. Lecture Notes in Computer Science, vol. 3172, pp. 73–84. Springer, Berlin (2004)
2. Acan, A.: An external partial permutations memory for ant colony optimization. In: Raidl, G., Gottlieb, J. (eds.) *Evolutionary Computation in Combinatorial Optimization*. Lecture Notes in Computer Science, vol. 3448, pp. 1–11. Springer, Berlin (2005)
3. Alaya, I., Solnon, C., Ghédira, K.: Ant colony optimization for multi-objective optimization problems. In: *19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007)*, vol. 1, pp. 450–457. IEEE Computer Society, Los Alamitos, CA (2007)
4. Alexandrov, D.A., Kochetov, Y.A.: The behavior of the ant colony algorithm for the set covering problem. In: Inderfurth, K., Schwödauer, G., Domschke, W., Juhnke, F., Kleinschmidt, P., Wäscher, G. (eds.) *Operations Research Proceedings 1999*, pp. 255–260. Springer, Berlin (2000)
5. Angus, D., Woodward, C.: Multiple objective ant colony optimization. *Swarm Intell.* **3**(1), 69–85 (2009)
6. Applegate, D., Bixby, R.E., Chvátal, V., Cook, W.J.: *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, Princeton, NJ (2006)
7. Balaprakash, P., Birattari, M., Stützle, T., Yuan, Z., Dorigo, M.: Estimation-based ant colony optimization algorithms for the probabilistic travelling salesman problem. *Swarm Intell.*, **3**(3), 223–242 (2009)
8. Bauer, A., Bullnheimer, B., Hartl, R.F., Strauss, C.: An ant colony optimization approach for the single machine total tardiness problem. In: *Proceedings of the 1999 Congress on Evolutionary Computation (CEC'99)*, pp. 1445–1450. IEEE Press, Piscataway, NJ (1999)
9. Beckers, R., Deneubourg, J.-L., Goss, S.: Modulation of trail laying in the ant *Lasius niger* (hymenoptera: Formicidae) and its role in the collective selection of a food source. *J. Insect Behav.* **6**(6), 751–759 (1993)
10. Bellman, R., Esogbue, A.O., Nabeshima, I.: *Mathematical Aspects of Scheduling and Applications*. Pergamon Press, New York, NY (1982)
11. Benedettini, S., Roli, A., Di Gaspero, L.: Two-level ACO for haplotype inference under pure parsimony. In: Dorigo, M., Birattari, M., Blum, C., Clerc, M., Stützle, T., Winfield, A. F. T. (eds.) *Ant Colony Optimization and Swarm Intelligence, 6th International Workshop, ANTS 2008*. Lecture Notes in Computer Science, vol. 5217, pp. 179–190. Springer, Berlin (2008)
12. Bertsekas, D.: *Network Optimization: Continuous and Discrete Models*. Athena Scientific, Belmont, MA (1998)
13. Bianchi, L., Birattari, M., Manfrin, M., Mastrolilli M., Paquete, L., Rossi-Doria, O., Schiavinotto, T.: Hybrid metaheuristics for the vehicle routing problem with stochastic demands. *J. Math. Model. Algorithms* **5**(1), 91–110 (2006)
14. Bianchi, L., Gambardella, L.M., Dorigo, M.: An ant colony optimization approach to the probabilistic traveling salesman problem. In: Merelo Guervós, J.J., Adamidis, P., Beyer, H.-G., Fernández-Villacanas, J.-L., Schwefel, H.-P. (eds.) *Parallel Problem Solving from Nature – PPSN VII: 7th International Conference*, Lecture Notes in Computer Science, vol. 2439, pp. 883–892. Springer, Berlin (2002)

15. Bilchev, G., Parmee, I.C.: The ant colony metaphor for searching continuous design spaces. In: Fogarty, T.C. (ed.) *Evolutionary Computing, AISB Workshop, Lecture Notes in Computer Science*, vol. 993, pp. 25–39. Springer, Berlin (1995)
16. Birattari, M., Di Caro, G., Dorigo, M.: Toward the formal foundation of ant programming. In: Dorigo, M., Di Caro, G., Sampels, M. (eds.) *Ant Algorithms: Third International Workshop, ANTS 2002, Lecture Notes in Computer Science*, vol. 2463, pp. 188–201. Springer, Berlin (2002)
17. Blum, C.: *Theoretical and Practical Aspects of Ant Colony Optimization*. PhD Thesis, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, 2004
18. Blum, C.: Beam-ACO—Hybridizing ant colony optimization with beam search: An application to open shop scheduling. *Comput. Oper. Res.* **32**(6), 1565–1591 (2005)
19. Blum, C.: Beam-ACO for simple assembly line balancing. *INFORMS J. Comput.* **20**(4), 618–627 (2008)
20. Blum, C., Blesa, M. J.: New metaheuristic approaches for the edge-weighted k-cardinality tree problem. *Comput. Oper. Res.* **32**(6), 1355–1377 (2005)
21. Blum, C., Dorigo, M.: The hyper-cube framework for ant colony optimization. *IEEE Trans. Syst. Man Cybern. – Part B* **34**(2), 1161–1172 (2004)
22. Blum, C., Dorigo, M.: Search bias in ant colony optimization: on the role of competition-balanced systems. *IEEE Trans. Evol. Comput.* **9**(2), 159–174 (2005)
23. Blum, C., Sampels, M.: Ant colony optimization for FOP shop scheduling: a case study on different pheromone representations. In: *Proceedings of the 2002 Congress on Evolutionary Computation (CEC'02)*, pp. 1558–1563. IEEE Press, Piscataway, NJ, 2002
24. Blum, C., Sampels, M., Zlochin, M.: On a particularity in model-based search. In: Langdon, W.B. et al. (eds.) *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002)*, pp. 35–42. Morgan Kaufmann, San Francisco, CA (2002)
25. Blum, C., Yabar, M., Blesa, M.J.: An ant colony optimization algorithm for DNA sequencing by hybridization. *Comput. Oper. Res.* **35**(11), 3620–3635 (2008)
26. Boese, K.D., Kahng, A.B., Muddu, S.: A new adaptive multi-start technique for combinatorial global optimization. *Oper. Res. Lett.* **16**, 101–113 (1994)
27. Bolondi, M., Bondanza, M.: Parallelizzazione di un algoritmo per la risoluzione del problema del commesso viaggiatore. Master's thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1993
28. Brailsford, S.C., Gutjahr, W.J., Rauner, M.S., Zeppelzauer, W.: Combined discrete-event simulation and ant colony optimisation approach for selecting optimal screening policies for diabetic retinopathy. *Comput. Manage. Sci.* **4**(1), 59–83 (2006)
29. Bullnheimer, B., Hartl, R.F., Strauss, C.: A new rank based version of the ant system—a computational study. Technical Report, Institute of Management Science, University of Vienna, 1997
30. Bullnheimer, B., Hartl, R.F., Strauss, C.: A new rank-based version of the ant system: A computational study. *Cent. Eur. J. Oper. Res. Econ.* **7**(1), 25–38 (1999)
31. Bullnheimer, B., Kotsis, G., Strauss, C.: Parallelization strategies for the ant system. In: De Leone, R., Murli, A., Pardalos, P., Toraldo, G. (eds.) *High Performance Algorithms and Software in Nonlinear Optimization*. Kluwer Series of Applied Optimization, vol. 24 pp. 87–100. Kluwer, The Netherlands (1998)
32. Cantú-Paz, E.: *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer, Boston, MA (2000)
33. Chen, L., Zhang, C.: Adaptive parallel ant colony algorithm. In: *Advances in Natural Computation, First International Conference, ICNC 2005*. Lecture Notes in Computer Science, vol. 3611, pp. 1239–1249. Springer, Berlin (2005)
34. Colomi, A., Dorigo, M., Maniezzo, V.: Distributed optimization by ant colonies. In: Varela, F.J., Bourgine, P. (eds.) *Proceedings of the First European Conference on Artificial Life*, pp. 134–142. MIT Press, Cambridge, MA (1992)

35. Colorni, A., Dorigo, M., Maniezzo, V.: An investigation of some properties of an ant algorithm. In: Männer, R., Manderick, B. (eds.) Parallel Problem Solving from Nature – PPSN II, pp. 509–520. North-Holland, Amsterdam, The Netherlands (1992)
36. Cordón, O., Fernández de Viana, I., Herrera, F.: Analysis of the best-worst Ant System and its variants on the TSP. *Math. Soft Comput.* **9**(2–3), 177–192 (2002)
37. Cordón, O., Fernández de Viana, I., Herrera, F., Moreno, L.: A new ACO model integrating evolutionary computation concepts: The best-worst Ant System. In: Dorigo, M., Middendorf, M., Stützle, T. (eds.) Abstract proceedings of ANTS 2000 – From Ant Colonies to Artificial Ants: Second International Workshop on Ant Algorithms, pp. 22–29. IRIDIA, Université Libre de Bruxelles, Brussels, Belgium (2000)
38. Cordón, O., Herrera, F., Stützle, T.: Special issue on ant colony optimization: models and applications. *Mathw. Soft Comput.* **9**(2–3), 137–268 (2003)
39. Costa, D., Hertz, A.: Ants can colour graphs. *J. Oper. Res. Soc.* **48**, 295–305 (1997)
40. de Campos, L.M., Fernández-Luna, J.M., Gámez, J.A., Puerta, J.M.: Ant colony optimization for learning Bayesian networks. *Int. J. Approx. Reasoning* **31**(3), 291–311 (2002)
41. de Campos, L.M., Gamez, J.A., Puerta, J.M.: Learning Bayesian networks by ant colony optimisation: searching in the space of orderings. *Mathw. Soft Comput.* **9**(2–3), 251–268 (2002)
42. den Besten, M.L., Stützle, T., Dorigo, M.: Ant colony optimization for the total weighted tardiness problem. In: Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Merelo, J.J., Schwefel, H.-P. (eds.) Proceedings of PPSN-VI, Sixth International Conference on Parallel Problem Solving from Nature. Lecture Notes in Computer Science, vol. 1917, pp. 611–620. Springer, Berlin (2000)
43. Deneubourg, J.-L., Aron, S., Goss, S., Pasteels, J.-M.: The self-organizing exploratory pattern of the Argentine ant. *J. Insect Behav.* **3**, 159–168 (1990)
44. Di Caro, G.: Ant Colony Optimization and its application to adaptive routing in telecommunication networks. PhD thesis, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, 2004
45. Di Caro, G., Dorigo, M.: AntNet: a mobile agents approach to adaptive routing. Technical Report IRIDIA/97-12, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, 1997
46. Di Caro, G., Dorigo, M.: Ant colonies for adaptive routing in packet-switched communications networks. In: Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.-P. (eds.) Proceedings of PPSN-V, Fifth International Conference on Parallel Problem Solving from Nature. Lecture Notes in Computer Science, vol. 1498, pp. 673–682. Springer, Berlin (1998)
47. Di Caro, G., Dorigo, M.: AntNet: distributed stigmergetic control for communications networks. *J. Artif. Intell. Res.* **9**, 317–365 (1998)
48. Di Caro, G., Dorigo, M.: Mobile agents for adaptive routing. In: El-Rewini, H. (ed.) Proceedings of the 31st International Conference on System Sciences (HICSS-31), pp. 74–83. IEEE Computer Society Press, Los Alamitos, CA (1998)
49. Di Caro, G., Ducatelle, F., Gambardella, L.M.: AntHocNet: an adaptive nature-inspired algorithm for routing in mobile ad hoc networks. *Eur. Trans. Telecomm.* **16**(5), 443–455 (2005)
50. Doerner, K.F., Hartl, R.F., Benkner, S., Lucka, M.: Parallel cooperative saving based ant colony optimization - multiple search and decomposition approaches. *Parallel Process. Lett.* **16**(3), 351–369 (2006)
51. Doerner, K.F., Hartl, R.F., Reimann, M.: Are CompetAnts more competent for problem solving? The case of a multiple objective transportation problem. *Cent. Eur. J. Oper. Res. Econ.* **11**(2), 115–141 (2003)
52. Doerner, K.F., Merkle, D., Stützle, T.: Special issue on ant colony optimization. *Swarm Intell.* **3**(1), 1–85 (2009)
53. Doerr, B., Neumann, F., Sudholt, D., Witt, C.: On the runtime analysis of the 1-ANT ACO algorithm. In: Genetic and Evolutionary Computation Conference, GECCO 2007, Proceedings, pp. 33–40. ACM press, New York, NY (2007)

54. Donati, A.V., Montemanni, R., Casagrande, N., Rizzoli, A.E., Gambardella, L.M.: Time dependent vehicle routing problem with a multi ant colony system. *Euro. J. Oper. Res.* **185**(3), 1174–1191 (2008)
55. Dorigo, M.: Optimization, learning and natural algorithms (in Italian). PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1992
56. Dorigo, M., Blum, C.: Ant colony optimization theory: a survey. *Theor. Comput. Sci.* **344**(2–3), 243–278 (2005)
57. Dorigo, M., Di Caro, G.: The ant colony optimization meta-heuristic. In: Corne, D., Dorigo, M., Glover, F. (eds.) *New Ideas in Optimization*, pp. 11–32. McGraw Hill, London, UK (1999)
58. Dorigo, M., Di Caro, G., Stützle T. (eds.): Special issue on “Ant Algorithms”. *Future Gen. Comput. Syst.* **16**(8), 851–946 (2000)
59. Dorigo, M., Di Caro, G., Gambardella, L. M. Ant algorithms for discrete optimization. *Artif. Life* **5**(2), 137–172 (1999)
60. Dorigo, M., Gambardella, L.M.: Ant colonies for the traveling salesman problem. *BioSystems* **43**, 73–81 (1997)
61. Dorigo, M., Gambardella, L.M.: Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Trans. Evol. Comput.* **1**(1), 53–66 (1997)
62. Dorigo, M., Gambardella, L.M., Middendorf, M., Stützle, T. (eds.): Special section on “Ant Colony Optimization”. *IEEE Trans. Evol. Comput.* **6**(4), 317–365 (2002)
63. Dorigo, M., Maniezzo, V., Colorni, A.: The ant system: an autocatalytic optimizing process. Technical Report 91-016 Revised, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1991
64. Dorigo, M., Maniezzo, V., Colorni, A.: Positive feedback as a search strategy. Technical Report 91-016, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1991
65. Dorigo, M., Maniezzo, V., Colorni, A.: Ant system: optimization by a colony of cooperating agents. *IEEE Trans. Syst. Man Cybern. – Part B* **26**(1), 29–41 (1996)
66. Dorigo, M., Stützle, T.: *Ant Colony Optimization*. MIT Press, Cambridge, MA (2004)
67. Dréo, J., Siarry, P.: Continuous interacting ant colony algorithm based on dense heterarchy. *Future Gen. Comput. Syst.* **20**(5), 841–856 (2004)
68. Ducatelle, F., Di Caro, G., Gambardella, L.M.: Using ant agents to combine reactive and proactive strategies for routing in mobile ad hoc networks. *Int. J. Comput. Intell. Appl.* **5**(2), 169–184 (2005)
69. Ducatelle, F., Di Caro, G., Gambardella, L.M.: Principles and applications of swarm intelligence for adaptive routing in telecommunications networks. *Swarm Intell.* (2009)
70. Eyckelhof, C.J., Snoek, M.: Ant systems for a dynamic TSP: ants caught in a traffic jam. In: Dorigo, M., Di Caro, G., Sampels, M. (eds.) *Ant Algorithms: Third International Workshop, ANTS 2002*. Lecture Notes in Computer Science, vol. 2463 pp. 88–99. Springer, Berlin (2002)
71. Farooq, M., Di Caro, G.: Routing protocols for next-generation intelligent networks inspired by collective behaviors of insect societies. In: Blum, C., Merkle, D. (eds.) *Swarm Intelligence: Introduction and Applications*, Natural Computing Series, pp. 101–160. Springer, Berlin (2008)
72. Favaretto, D., Moretti, E., Pellegrini, P.: Ant colony system for a VRP with multiple time windows and multiple visits. *J. Interdiscip. Math.* **10**(2), 263–284 (2007)
73. Fuellerer, G., Doerner, K.F., Hartl, R.F., Iori, M.: Ant colony optimization for the two-dimensional loading vehicle routing problem. *Comput. Oper. Res.* **36**(3), 655–673 (2009)
74. Gambardella, L.M., Dorigo, M.: Ant-Q: a reinforcement learning approach to the traveling salesman problem. In: Priditidis, A., Russell, S. (eds.) *Proceedings of the Twelfth International Conference on Machine Learning (ML-95)*, pp. 252–260. Morgan Kaufmann Publishers, Palo Alto, CA (1995)
75. Gambardella, L.M., Dorigo, M.: Solving symmetric and asymmetric TSPs by ant colonies. In: *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation (ICEC'96)*, pp. 622–627. IEEE Press, Piscataway, NJ (1996)

76. Gambardella, L.M., Dorigo, M.: Ant colony system hybridized with a new local search for the sequential ordering problem. *INFORMS J. Comput.* **12**(3), 237–255 (2000)
77. Gambardella, L.M., Taillard, É.D., Agazzi, G. MACS-VRPTW: a multiple ant colony system for vehicle routing problems with time windows. In: Corne, D., Dorigo, M., Glover, F. (eds.) *New Ideas in Optimization*, pp. 63–76. McGraw Hill, London, UK (1999)
78. García-Martínez, C., Cordón, O., Herrera, F.: A taxonomy and an empirical analysis of multiple objective ant colony optimization algorithms for the bi-criteria TSP. *Euro. J. Oper. Res.* **180**(1), 116–148 (2007)
79. Garey, M.R., Johnson, D.S.: *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, San Francisco, CA (1979)
80. Goss, S., Aron, S., Deneubourg, J.L., Pasteels, J.M.: Self-organized shortcuts in the Argentine ant. *Naturwissenschaften* **76**, 579–581 (1989)
81. Guntsch, M., Middendorf, M.: Pheromone modification strategies for ant algorithms applied to dynamic TSP. In: Boers, E.J.W., Gottlieb, J., Lanzi, P.L., Smith, R.E., Cagnoni, S., Hart, E., Raidl, G.R., Tijink, H. (eds.) *Applications of Evolutionary Computing: Proceedings of EvoWorkshops 2001*, Lecture Notes in Computer Science, vol. 2037, pp. 213–222. Springer, Berlin (2001)
82. Guntsch, M., Middendorf, M.: A population based approach for ACO. In: Cagnoni, S., Gottlieb, J., Hart, E., Middendorf, M., Raidl, G. R. editors, *Applications of Evolutionary Computing, Proceedings of EvoWorkshops2002: EvoCOP, EvoIASP, EvoSTim*. Lecture Notes in Computer Science, vol. 2279, pp. 71–80. Springer, Berlin (2002)
83. Gutjahr, W.J.: A graph-based ant system and its convergence. *Future Gen. Comput. Syst.* **16**(8), 873–888 (2000)
84. Gutjahr, W.J.: ACO algorithms with guaranteed convergence to the optimal solution. *Inf. Process. Lett.* **82**(3), 145–153 (2002)
85. Gutjahr, W.J.: S-ACO: an ant-based approach to combinatorial optimization under uncertainty. In: Dorigo, M., Gambardella, L., Mondada, F., Stützle, T., Birratiari, M., Blum, C. (eds.) *Ant Colony Optimization and Swarm Intelligence: 4th International Workshop, ANTS 2004*. Lecture Notes in Computer Science, vol. 3172, pp. 238–249. Springer, Berlin (2004)
86. Gutjahr, W.J.: On the finite-time dynamics of ant colony optimization. *Methodol. Comput. Appl. Probability* **8**(1), 105–133 (2006)
87. Gutjahr, W.J.: Mathematical runtime analysis of ACO algorithms: survey on an emerging issue. *Swarm Intell.* **1**(1), 59–79 (2007)
88. Gutjahr, W.J.: First steps to the runtime complexity analysis of ant colony optimization. *Comput. OR* **35**(9), 2711–2727 (2008)
89. Gutjahr, W.J., Sebastiani, G.: Runtime analysis of ant colony optimization with best-so-far reinforcement. *Methodol. Comput. Appl. Probability* **10**, 409–433 (2008)
90. Hadji, R., Rahoual, M., Talbi, E., Bachelet, V.: Ant colonies for the set covering problem. In: Dorigo, M., Middendorf, M., Stützle, T. (eds.) *Abstract proceedings of ANTS 2000 – From Ant Colonies to Artificial Ants: Second International Workshop on Ant Algorithms*, pp. 63–66. Université Libre de Bruxelles, Brussels, Belgium (2000)
91. Hernández, H., Blum, C.: Ant colony optimization for multicasting in static wireless ad-hoc networks. *Swarm Intell.* **3**(2), 125–148 (2009)
92. López Ibáñez, M., Paquete, L., Stützle, T.: On the design of ACO for the biobjective quadratic assignment problem. In: Dorigo, M., Gambardella, L., Mondada, F., Stützle, T., Birratiari, M., Blum, C. (eds.) *ANTS'2004*, Fourth International Workshop on Ant Algorithms and Swarm Intelligence, Lecture Notes in Computer Science, vol. 3172, pp. 214–225. Springer, Berlin (2004)
93. Iredi, S., Merkle, D., Middendorf, M.: Bi-criterion optimization with multi colony ant algorithms. In: Zitzler, E., Deb, K., Thiele, L., Coello Coello, C.A., Corne, D. (eds.) *First International Conference on Evolutionary Multi-Criterion Optimization, (EMO'01)*. Lecture Notes in Computer Science, vol. 1993, pp. 359–372. Springer, Berlin (2001)

94. Johnson, D.S., McGeoch, L.A.: The travelling salesman problem: a case study in local optimization. In: Aarts, E.H.L., Lenstra, J.K. (eds.) *Local Search in Combinatorial Optimization*, pp. 215–310. Wiley, Chichester, UK (1997)
95. Jünger, M., Reinelt, G., Thiel, S.: Provably good solutions for the traveling salesman problem. *Zeitschrift für Oper. Res.* **40**, 183–217 (1994)
96. Khichane, M., Albert, P., Solnon, C.: Integration of ACO in a constraint programming language. In: Dorigo, M., Birattari, M., Blum, C., Clerc, M., Stützle, T., Winfield, A.F.T. (eds.) *Ant Colony Optimization and Swarm Intelligence, 6th International Conference, ANTS 2008. Lecture Notes in Computer Science*, vol. 5217, pp. 84–95. Springer, Berlin (2008)
97. Korb, O., Stützle, T., Exner, T.E.: Application of ant colony optimization to structure-based drug design. In: Dorigo, M., Birattari, M., Blum, C., Clerc, M., Stützle, T., Winfield, A.F.T. (eds.) *Ant Colony Optimization and Swarm Intelligence, 5th International Workshop, ANTS 2006. Lecture Notes in Computer Science*, vol. 4150, pp. 247–258. Springer, Berlin (2006)
98. Korb, O., Stützle, T., Exner, T.E.: An ant colony optimization approach to flexible protein-ligand docking. *Swarm Intelli.* **1**(2), 115–134 (2007)
99. Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B.: *The Travelling Salesman Problem*. Wiley, Chichester, UK (1985)
100. Leguizamón, G., Michalewicz, Z.: A new version of ant system for subset problems. In: *Proceedings of the 1999 Congress on Evolutionary Computation (CEC'99)*, pp. 1459–1464. IEEE Press, Piscataway, NJ (1999)
101. Lessing, L., Dumitrescu, I., Stützle, T.: A comparison between ACO algorithms for the set covering problem. In: Dorigo, M., Gambardella, L., Mondada, F., Stützle, T., Birattari, M., Blum, C. (eds.) *Ant Colony Optimization and Swarm Intelligence: 4th International Workshop, ANTS 2004. Lecture Notes in Computer Science*, vol. 3172, pp. 1–12. Springer, Berlin (2004)
102. López-Ibáñez, M., Blum, C., Thiruvady, D., Ernst, A.T., Meyer, B.: Beam-ACO based on stochastic sampling for makespan optimization concerning the TSP with time windows. In: Cotta, C., Cowling, P. (eds.) *Evolutionary Computation in Combinatorial Optimization. Lecture Notes in Computer Science*, vol. 5482 pp. 97–108. Springer, Berlin (2009)
103. Manfrin, M., Birattari, M., Stützle, T., Dorigo, M.: Parallel ant colony optimization for the traveling salesman problem. In: Dorigo, M., Gambardella, L.M., Birattari, M., Martinoli, A., Poli, R., Stützle, T. (eds.) *Ant Colony Optimization and Swarm Intelligence: 5th International Workshop, ANTS 2006. Lecture Notes in Computer Science*, vol. 4150, pp. 224–234. Springer, Berlin (2006)
104. Maniezzo, V.: Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. Technical Report CSR 98-1, Scienze dell'Informazione, Università di Bologna, Sede di Cesena, Italy, 1998
105. Maniezzo, V.: Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. *INFORMS J. Comput.* **11**(4), 358–369 (1999)
106. Maniezzo, V., Carbonaro, A.: An ANTS heuristic for the frequency assignment problem. *Future Gen. Comput. Syst.* **16**(8), 927–935 (2000)
107. Martens, D., De Backer, M., Haesen, R., Vanthienen, J., Snoeck, M., Baesens, B.: Classification with ant colony optimization. *IEEE Trans. Evol. Comput.* **11**(5), 651–665 (2007)
108. Merkle, D., Middendorf, M.: Modeling the dynamics of ant colony optimization. *Evol. Comput.* **10**(3), 235–262 (2002)
109. Merkle, D., Middendorf, M.: Ant colony optimization with global pheromone evaluation for scheduling a single machine. *Appl. Intell.* **18**(1), 105–111 (2003)
110. Merkle, D., Middendorf, M., Schmeck, H.: Ant colony optimization for resource-constrained project scheduling. In: Whitley, D., Goldberg, D., Cantu-Paz, E., Spector, L., Parmee, I., Beyer, H.-G. (eds.) *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, pp. 893–900. Morgan Kaufmann, San Francisco, CA (2000)
111. Merkle, D., Middendorf, M., Schmeck, H.: Ant colony optimization for resource-constrained project scheduling. *IEEE Trans. Evol. Comput.* **6**(4), 333–346 (2002)

112. Meuleau, N., Dorigo, M.: Ant colony optimization and stochastic gradient descent. *Artif. Life* **8**(2), 103–121 (2002)
113. Meyer, B., Ernst, A.: Integrating ACO and constraint propagation. In: Dorigo, M., Birattari, M., Blum, C., Gambardella, L.M., Mondada, F., Stützle, T. (eds.) *Ant Colony Optimization and Swarm Intelligence, 4th International Workshop, ANTS 2004. Lecture Notes in Computer Science*, vol. 3172, pp. 166–177. Springer, Berlin (2004)
114. Michel, R., Middendorf, M.: An ACO algorithm for the shortest supersequence problem. In: Corne, D., Dorigo, M., Glover, F. (eds.) *New Ideas in Optimization*, pp. 51–61. McGraw Hill, London, UK (1999)
115. Middendorf, M., Reischle, F., Schmeck, H.: Multi colony ant algorithms. *J. Heuristics* **8**(3), 305–320 (2002)
116. Monmarché, N., Venturini, G.: On how *Pachycondyla apicalis* ants suggest a new search algorithm. *Future Gen. Comput. Syst.* **16**(8), 937–946 (2000)
117. Montemanni, R., Gambardella, L.M., Rizzoli, A.E., Donati, A.V.: Ant colony system for a dynamic vehicle routing problem. *J. Comb. Optimization* **10**, 327–343 (2005)
118. Morton, T.E., Rachamadugu, R.M., Vepsäläinen, A.: Accurate myopic heuristics for tardiness scheduling. *GSIA Working Paper 36-83-84*, Carnegie Mellon University, Pittsburgh, PA, 1984
119. Neumann, F., Sudholt, D., Witt, C.: Analysis of different MMAS ACO algorithms on unimodal functions and plateaus. *Swarm Intell.* **3**(1), 35–68 (2009)
120. Neumann, F., Witt, C.: Runtime analysis of a simple ant colony optimization algorithm. *Electron. Colloq. Comput. Complexity (ECCC)* **13**(084) (2006)
121. Otero, F.E.B., Freitas, A.A., Johnson, C.G.: cAnt-Miner: an ant colony classification algorithm to cope with continuous attributes. In: Dorigo, M., Birattari, M., Blum, C., Clerc, M., Stützle, T., Winfield, A.F.T. (eds.) *Ant Colony Optimization and Swarm Intelligence, 6th International Workshop, ANTS 2008. Lecture Notes in Computer Science*, vol. 5217, pp. 48–59. Springer, Berlin (2008)
122. Ow, P.S., Morton, T.E.: Filtered beam search in scheduling. *Int. J. Prod. Res.* **26**, 297–307 (1988)
123. Papadimitriou, C.H.: *Computational Complexity*. Addison-Wesley, Reading, MA (1994)
124. Parpinelli, R.S., Lopes, H.S., Freitas, A.A.: Data mining with an ant colony optimization algorithm. *IEEE Trans. Evol. Comput.* **6**(4), 321–332 (2002)
125. Rajendran, C., Ziegler, H.: Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. *Eur. J. Oper. Res.* **155**(2), 426–438 (2004)
126. Randall, M., Lewis, A.: A parallel implementation of ant colony optimization. *J. Parallel Distr. Comput.* **62**(9), 1421–1432 (2002)
127. Reimann, M., Doerner, K., Hartl, R.F.: D-ants: savings based ants divide and conquer the vehicle routing problems. *Comput. Oper. Res.* **31**(4), 563–591 (2004)
128. Reinelt, G.: *The Traveling Salesman: Computational Solutions for TSP Applications. Lecture Notes in Computer Science*, vol. 840, Springer, Berlin (1994)
129. Rizzoli, A.E., Montemanni, R., Lucibello, E., Gambardella, L.M.: Ant colony optimization for real-world vehicle routing problems. From theory to applications. *Swarm Intell.* **1**(2), 135–151 (2007)
130. Ruiz, R., Stützle, T.: A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *Euro. J. Oper. Res.* **177**(3), 2033–2049 (2007)
131. Schoonderwoerd, R., Holland, O., Bruten, J., Rothkrantz, L.: Ant-based load balancing in telecommunications networks. *Adaptive Behav.* **5**(2), 169–207 (1996)
132. Shmygelska, A., Hoos, H.H.: An ant colony optimisation algorithm for the 2D and 3D hydrophobic polar protein folding problem. *BMC Bioinformat.* **6**, 30 (2005)
133. Sim, K.M., Sun, W.H.: Ant colony optimization for routing and load-balancing: survey and new directions. *IEEE Trans. Syst. Man Cyber.-Part A: Syst. Hum.* **33**(5), 560–572 (2003)
134. Socha, K.: ACO for continuous and mixed-variable optimization. In: Dorigo, M., Gambardella, L., Mondada, F., Stützle, T., Birratti, M., Blum, C. (eds.) *Ant Colony Optimization*

- and Swarm Intelligence: 4th International Workshop, ANTS 2004. Lecture Notes in Computer Science, vol. 3172, pp. 25–36. Springer, Berlin (2004)
- 135. Socha, K., Blum, C.: An ant colony optimization algorithm for continuous optimization: An application to feed-forward neural network training. *Neural Comput. Appl.* **16**(3), 235–248 (2007)
  - 136. Socha, K., Dorigo, M.: Ant colony optimization for mixed-variable optimization problems. Technical Report TR/IRIDIA/2007-019, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, October 2007
  - 137. Socha, K., Dorigo, M.: Ant colony optimization for continuous domains. *Eur. J. Oper. Res.* **185**(3), 1155–1173 (2008)
  - 138. Socha, K., Knowles, J., Sampels, M.: A  $\mathcal{MAX}$ - $\mathcal{MIN}$  ant system for the university course timetabling problem. In: Dorigo, M., Di Caro, G., Sampels, M. (eds.) *Ant Algorithms: Third International Workshop, ANTS 2002*. Lecture Notes in Computer Science, vol. 2463, pp. 1–13. Springer, Berlin (2002)
  - 139. Socha, K., Sampels, M., Manfrin, M.: Ant algorithms for the university course timetabling problem with regard to the state-of-the-art. In: Raidl, G.R., Meyer, J.-A., Middendorf, M., Cagnoni, S., Cardalda, J.J.R., Corne, D.W., Gottlieb, J., Guillot, A., Hart, E., Johnson, C.G., Marchiori, E. (eds.) *Applications of Evolutionary Computing, Proceedings of EvoWorkshops 2003*. Lecture Notes in Computer Science, vol. 2611, pp. 334–345. Springer, Berlin (2003)
  - 140. Solnon, C.: Combining two pheromone structures for solving the car sequencing problem with ant colony optimization. *Eur. J. Oper. Res.* **191**(3), 1043–1055 (2008)
  - 141. Solnon, C., Fenet, S.: A study of ACO capabilities for solving the maximum clique problem. *J. Heuristics* **12**(3), 155–180 (2006)
  - 142. Stützle, T.: An ant approach to the flow shop problem. In: *Proceedings of the Sixth European Congress on Intelligent Techniques & Soft Computing (EUFIT'98)*, vol. 3, pp. 1560–1564. Verlag Mainz, Wissenschaftsverlag, Aachen, Germany, 1998
  - 143. Stützle, T.: Parallelization strategies for ant colony optimization. In: Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.-P. (eds.) *Proceedings of PPSN-V, Fifth International Conference on Parallel Problem Solving from Nature*. Lecture Notes in Computer Science, vol. 1498, pp. 722–731. Springer, Berlin (1998)
  - 144. Stützle, T.: Local Search Algorithms for Combinatorial Problems: Analysis, Improvements, and New Applications, DISKI, vol. 220, Infix, Sankt Augustin, Germany, 1999
  - 145. Stützle, T., Dorigo, M.: A short convergence proof for a class of ACO algorithms. *IEEE Trans. Evol. Comput.* **6**(4), 358–365 (2002)
  - 146. Stützle, T., Hoos, H.H.: Improving the ant system: a detailed report on the  $\mathcal{MAX}$ - $\mathcal{MIN}$  Ant System. Technical Report AIDA-96-12, FG Intellektik, FB Informatik, TU Darmstadt, Germany, August 1996
  - 147. Stützle, T., Hoos, H.H.: The  $\mathcal{MAX}$ - $\mathcal{MIN}$  Ant System and local search for the traveling salesman problem. In: Bäck, T., Michalewicz, Z., Yao, X. (eds.) *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation (ICEC'97)*, pp. 309–314. IEEE Press, Piscataway, NJ (1997)
  - 148. Stützle, T., Hoos, H.H.:  $\mathcal{MAX}$ - $\mathcal{MIN}$  ant system. *Future Gen. Comput. Syst.* **16**(8), 889–914 (2000)
  - 149. Sutton, R.S., Barto, A.G.: *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA (1998)
  - 150. Talbi, E.-G., Roux, O.H., Fonlupt, C., Robillard, D.: Parallel ant colonies for the quadratic assignment problem. *Future Gen. Comput. Syst.* **17**(4), 441–449 (2001)
  - 151. Tsutsui, S.: Ant colony optimisation for continuous domains with aggregation pheromones metaphor. In: *Proceedings of the The 5th International Conference on Recent Advances in Soft Computing (RASC-04)*, pp. 207–212, Nottingham, UK (2004)
  - 152. Tsutsui, S.: cAS: Ant colony optimization with cunning ants. In: Runarsson, T.P., Beyer, H.-G., Burke, E.K., Merelo Guervós, J.J., Whitley, L.D., Yao, X. (eds.) *Parallel Problem Solving from Nature—PPSN IX, 9th International Conference*. Lecture Notes in Computer Science, vol. 4193, pp. 162–171. Springer, Berlin (2006)

153. Tsutsui, S.: An enhanced aggregation pheromone system for real-parameter optimization in the ACO metaphor. In: Dorigo, M., Gambardella, L.M., Birattari, M., Martinoli, A., Poli, R., Stützle, T. (eds.) *Ant Colony Optimization and Swarm Intelligence: 5th International Workshop, ANTS 2006*. Lecture Notes in Computer Science, vol. 4150, pp. 60–71. Springer, Berlin (2006)
154. Twomey, C., Stützle, T., Dorigo, M., Manfrin, M., Birattari, M.: An analysis of communication policies for homogeneous multi-colony ACO algorithms. Technical Report TR/IRIDIA/2009-012, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, May 2009
155. Wiesemann, W., Stützle, T.: Iterated ants: an experimental study for the quadratic assignment problem. In: Dorigo, M., Gambardella, L.M., Birattari, M., Martinoli, A., Poli, R., Stützle, T. (eds.) *Ant Colony Optimization and Swarm Intelligence: 5th International Workshop, ANTS 2006*. Lecture Notes in Computer Science, vol. 4150, pp. 179–190. Springer, Berlin (2006)
156. Yagiura, M., Kishida, M., Ibaraki, T.: A 3-flip neighborhood local search for the set covering problem. *Eur. J. Oper. Res.* **172**, 472–499 (2006)
157. Yannakakis, M.: Computational complexity. In: Aarts, E.H.L., Lenstra, J.K. (eds.) *Local Search in Combinatorial Optimization*, pp. 19–55. Wiley, Chichester, UK (1997)
158. Yuan, Z., Fügenschuh, A., Homfeld, H., Balaprakash, P., Stützle, T., Schoch, M.: Iterated greedy algorithms for a real-world cyclic train scheduling problem. In: Blesa, M.J., Blum, C., Cotta, C., Fernández, A.J., Gallardo, J.E., Roli, A., Sampels, M. (eds.) *Hybrid Metaheuristics, 5th International Workshop, HM 2008*. Lecture Notes in Computer Science, vol. 5296, pp. 102–116. Springer, Berlin (2008)
159. Zhang, Y., Kuhn, L.D., Fromherz, M.P.J.: Improvements on ant routing for sensor networks. In: Dorigo, M., Gambardella, L.M., Mondada, F., Stützle, T., Birattari, M., Blum, C. (eds.) *Ant Colony Optimization and Swarm Intelligence: 4th International Workshop, ANTS 2004*. Lecture Notes in Computer Science, vol. 3172, pp. 154–165. Springer, Berlin (2004)
160. Zlochin, M., Birattari, M., Meuleau, N., Dorigo, M.: Model-based search for combinatorial optimization: a critical survey. *Ann. Oper. Res.* **131**(1–4), 373–395 (2004)



# Chapter 9

## Advanced Multi-start Methods

R. Martí, J. Marcos Moreno-Vega, and A. Duarte

**Abstract** Heuristic search procedures that aspire to find globally optimal solutions to hard combinatorial optimization problems usually require some type of diversification to overcome local optimality. One way to achieve diversification is to re-start the procedure from a new solution once a region has been explored. In this chapter we describe the best known multi-start methods for solving optimization problems. We propose classifying these methods in terms of their use of randomization, memory, and degree of rebuild. We also present a computational comparison of these methods on solving the maximum diversity problem in terms of solution quality and diversification power.

### 9.1 Introduction

Metaheuristics are high-level solution methods that provide guidelines to design and integrate subordinate heuristics to solve optimization problems. These high-level methods characteristically focus on strategies to escape from local optima and perform a robust search of a solution space. Most of them are based, at least partially,

---

R. Martí  
Departamento de Estadística e Investigación Operativa  
Universidad de Valencia, Spain  
e-mail: rafael.marti@uv.es

J. Marcos Moreno-Vega  
Departamento de Estadística, IO y Computación  
Universidad de La Laguna, La Laguna Santa Cruz de Tenerife, Spain  
e-mail: jmmoreno@ull.es

A. Duarte  
Departamento de Ciencias de la Computación  
Universidad Rey Juan Carlos, Madrid, Spain  
e-mail: abraham.duarte@urjc.es

on a neighborhood search, and the degree to which neighborhoods are exploited varies according to the type of method.

Multi-start procedures were originally conceived as a way to exploit a local or neighborhood search procedure, by simply applying it from multiple random initial solutions. It is well known that search methods based on local optimization that aspire to find global optima usually require some type of diversification to overcome local optimality. Without this diversification, such methods can become reduced to tracing paths that are confined to a small area of the solution space, making it impossible to find a global optimum. Multi-start methods, appropriately designed, incorporate a powerful form of diversification.

There are some problems in which it turns out to be much simpler and more effective to construct solutions than to apply neighborhood-based procedures. For example, in constrained scheduling problems it is difficult to define neighborhoods (i.e., structures that allow transitions from a given solution to so-called adjacent solutions) that maintain feasibility, whereas solutions can be created relatively easily by an appropriate process of construction. Something similar happens in simulation optimization where the model treats the objective-function evaluation as a black box, making the search algorithm context independent. In these problems the generation of solutions by stepwise constructions, according to information recorded during the search process, is more efficient than the exploration of solutions in the neighborhood of a given solution since the evaluation requires a simulation process that is usually very time consuming. Therefore, multi-start methods provide an appropriate framework within which to develop algorithms to solve combinatorial optimization problems.

The re-start mechanism of multi-start methods can be superimposed on many different search methods. Once a new solution has been generated, a variety of options can be used to improve it, ranging from a simple greedy routine to a complex metaheuristic. This chapter is focused on studying the different strategies and methods for generating solutions to launch a succession of new searches for a global optimum.

## 9.2 An Overview

Multi-start methods have two phases: the first one in which the solution is generated and the second one in which the solution is typically (but not necessarily) improved. Then, each global iteration produces a solution (usually a local optima) and the best overall is the algorithm's output.

In recent years, many heuristic algorithms have been proposed to solve some combinatorial optimization problems. Some of them are problem dependent and the ideas and strategies implemented are difficult to apply to different problems, while others are based on a framework that can be used directly to design-solving methods for other problems. In this section we describe the most relevant procedures in terms of applying them to a wide variety of problems. We pay special attention to the adaptation of *memory structures* to multi-start methods.

The explicit use of memory structures constitutes the core of a large number of intelligent-solving methods. They include tabu search [14], scatter search [19], evolutionary path relinking [35], and some hybridizations of multi-start procedures. These methods focus on exploiting a set of strategic memory designs. Tabu search (TS), the metaheuristic that launched this perspective, is the source of the term adaptive memory programming (AMP) to describe methods that use advanced memory strategies (and hence learning, in a non-trivial sense) to guide a search.

In the following sections we trace some of the more salient contributions to multi-start methods of the past two decades (though the origins of the methods go back somewhat farther). We have grouped them according to four categories: memory-based designs (Section 9.2.1), GRASP (Section 9.2.2), theoretical analysis (Section 9.2.3), constructive designs (Section 9.2.4), and hybrid designs (Section 9.2.5). Based on the analysis of these methods, we propose a classification of multi-start procedures (Section 9.3) in which the use of memory plays a central role.

### 9.2.1 Memory-Based Designs

Many papers on multi-start methods that appeared before the mid-1990s do not use explicit memory, as notably exemplified by the Monte Carlo random re-start approach in the context of nonlinear unconstrained optimization. Here, the method simply evaluates the objective function at randomly generated points. The probability of success approaches one as the sample size tends to infinity under very mild assumptions about the objective function. Many algorithms have been proposed that combine the Monte Carlo method with local search procedures [36]. The convergence for random re-start methods is studied in [40], where the probability distribution used to choose the next starting point can depend on how the search evolves. Some extensions of these methods seek to reduce the number of complete local searches that are performed and increase the probability that they start from points close to the global optimum [28]. More advanced probabilistic forms of re-starting based on memory functions were subsequently developed in [37] and [26].

Fleurent and Glover [11] propose some adaptive memory search principles to enhance multi-start approaches. The authors introduce a template of a constructive version of tabu search based on both a set of elite solutions and the intensification strategies based on identifying *strongly determined and consistent variables*. Strongly determined variables are those whose values cannot be changed without significantly eroding the objective function value or disrupting the values of other variables. A consistent variable is defined as one that receives a particular value in a significant portion of good solutions. The authors propose the inclusion of memory structures within the multi-start framework as it is done with tabu search. Computational experiments for the quadratic assignment problem show that these methods

improve significantly over previous multi-start methods like GRASP and random restart that do not incorporate memory-based strategies.

Patterson et al. [32] introduce a multi-start framework (Adaptive Reasoning Techniques, ART) based on memory structures. The authors implement the short-term and long-term memory functions, proposed in the tabu search framework, to solve the Capacitated Minimum Spanning Tree Problem. ART is an iterative, constructive solution procedure that implements learning methodologies on top of memory structures. ART derives its success from being able to learn about and modify the behavior of a primary greedy heuristic. The greedy heuristic is executed repeatedly, and for each new execution, constraints that prohibit certain solution elements from being considered by the greedy heuristic are probabilistically introduced. The active constraints are held in a short-term memory. A long-term memory holds information regarding the constraints that were in the active memory for the best set of solutions.

Glover [15] proposes approaches for creating improved forms of constructive multi-start and *strategic oscillation* methods based on new search principles: *persistent attractiveness* and *marginal conditional validity*. These concepts play a key role in deriving appropriate measures to capture information during prior search. Applied to constructive neighborhoods, strategic oscillation operates by alternating constructive and destructive phases, where each solution generated by a constructive phase is dismantled (to a variable degree) by the destructive phase, after which a new phase builds the solution anew. The conjunction of both phases and their associated memory structures provides the basis for an improved multi-start method.

The principle of *persistent attractiveness* says that good choices derive from making decisions that have often appeared attractive, but that have not previously been made within a particular region or phase of search. That is, persistent attractiveness also carries with it the connotation of persistently unselected within a specific domain or interval. The principle of *marginal conditional validity* specifies that as more decisions are made, the consequences of imposing them cause the problem to be more restricted. Consequently, as the search progresses future decisions face less complexity and less ambiguity about which choices are likely to be preferable. Therefore, early decisions are more likely to be bad ones or at least to look better than they should, once later decisions are made. Specific strategies for exploiting these concepts and their underlying principles are given in [15].

Beausoleil et al. [3] consider a multi-objective combinatorial optimization problem called Extended Knapsack Problem. By applying multi-start search and path relinking their solving method rapidly guide the search toward the most balanced zone of the Pareto-optimal front (the zone in which all the objectives are equally important). The Pareto relation is applied in order to designate a subset of the best generated solutions to be the current efficient set of solutions. A max–min criterion applied to the Hamming distance is used as a measure of dissimilarity in order to find diverse solutions to be combined. The performance of this approach is compared with several state-of-the-art multi-objective evolutionary algorithms for a suite of test problems taken from the literature.

### 9.2.2 GRASP

One of the most well-known multi-start methods is the Greedy Adaptive Search Procedures (GRASP), which was introduced by Feo and Resende [10]. It was first used to solve set covering problems [9]. Each GRASP iteration consists of constructing a trial solution and then applying a local search procedure to find a local optimum (i.e., the final solution for that iteration). The construction step is an adaptive and iterative process guided by a greedy evaluation function. It is iterative because the initial solution is built considering one element at a time. It is greedy because the addition of each element is guided by a greedy function. It is adaptive because the element chosen at any iteration in a construction is a function of previously chosen elements. (That is, the method is adaptive in the sense of updating relevant information from one construction step to the next.) At each stage, the next element to be added to the solution is randomly selected from a candidate list of high-quality elements according to the evaluation function. Once a solution has been obtained, it is typically improved by a local search procedure. The improvement phase performs a sequence of moves toward a local optimum solution, which becomes the output of a complete GRASP iteration. Some examples of successful applications are given in [24, 25, 34].

Laguna and Martí [25] introduce path-relinking within GRASP as a way to improve multi-start methods. Path-relinking has been suggested as an approach to integrate intensification and diversification strategies in the context of tabu search [18]. This approach generates new solutions by exploring trajectories that connect high-quality solutions, by starting from one of these solutions and generating a path in the neighborhood space that leads toward the other solutions. This is accomplished by selecting moves that introduce attributes contained in the *guiding* solutions. Relinking in the context of GRASP consists of finding a path between a solution found after an improvement phase and a chosen elite solution. Therefore, the relinking concept has a different interpretation within GRASP, since the solutions found from one iteration to the next are not linked by a sequence of moves (as in the case of tabu search). The proposed strategy can be applied to any method that produces a sequence of solutions; specifically, it can be used in any multi-start procedure. Based on these ideas, Binato et al. [4] proposed the Greedy Randomized Adaptive Path-Re-linking. Many different designs named *Evolutionary Path-Re-linking* have also been studied in [35].

Prais and Ribeiro [33] propose an improved GRASP implementation, called reactive GRASP, for a matrix decomposition problem arising in the context of traffic assignment in communication satellites. The method incorporates a memory structure to record information about previously found solutions. In reactive GRASP, the basic parameter which restricts the candidate list during the construction phase is self-adjusted, according to the quality of the previously found solutions. The proposed method matches most of the best solutions known.

### 9.2.3 Theoretical Analysis

From a theoretical point of view, Hu et al. [22] study the combination of the *gradient algorithm* with random initializations to find a global optimum. Efficacy of parallel processing, choice of the restart probability distribution, and number of restarts are studied for both discrete and continuous models. The authors show that the uniform probability is a good choice for restarting procedures.

Boese et al. [5] analyze relationships among local minima from the perspective of the best local minimum, finding convex structures in the cost surfaces. Based on the results of that study, they propose a multi-start method where starting points for greedy descent are adaptively derived from the best previously found local minima. In the first step, adaptive multi-start heuristics (AMS) generate  $r$  random starting solutions and run a greedy descent method from each one to determine a set of corresponding random local minima. In the second step, *adaptive starting solutions* are constructed based on the local minima obtained so far and improved with a greedy descent method. This improvement is applied several times from each adaptive starting solution to yield corresponding *adaptive local minima*. The authors test this method for the traveling salesman problem and obtain significant speedups over previous multi-start implementations. Hagen and Kahng [20] apply this method for the iterative partitioning problem.

Moreno et al. [30] propose a stopping rule for the multi-start method based on a statistical study of the number of iterations needed to find the global optimum. The authors introduce two random variables that together provide a way of estimating the number of global iterations needed to find the global optima: the number of initial solutions generated and the number of objective function evaluations performed to find the global optima. From these measures, the probability that the incumbent solution is the global optimum is evaluated via a normal approximation. Thus, at each global iteration, this value is computed and if it is greater than a fixed threshold, the algorithm stops, otherwise a new solution is generated. The authors illustrate the method using the median  $p$ -hub problem.

Simple forms of multi-start methods are often used to compare other methods and measure their relative contribution. Baluja [2] compares different genetic algorithms for six sets of benchmark problems commonly found in the GA literature: traveling salesman problem, job-shop scheduling, knapsack, bin packing, neural network weight optimization, and numerical function optimization. The author uses the multi-start method (multiple restart stochastic hill-climbing, MRSIH) as a baseline in the computational testing. Since solutions are represented with strings, the improvement step consists of a local search based on random flip of bits. The results indicate that using genetic algorithms for the optimization of static functions does not yield a benefit, in terms of the final result obtained, over simpler optimization heuristics. Other comparisons between MRSIH and GAs can be found, for example, in [1] or [43].

### 9.2.4 Constructive Designs

Multi-start procedures usually follow a global scheme in which generation and improvement are alternated for a certain number of iterations; but there are some applications in which the improvement can be applied several times within a global iteration. In the *incomplete construction methods*, the improvement phase is periodically invoked during the construction process of the partial solution rather than after the complete construction, as it is usually done. See [7] and [38] for successful applications of this approach to vehicle routing.

Hickernell and Yuan [21] present a multi-start algorithm for unconstrained global optimization based on *quasirandom samples*. Quasirandom samples are sets of deterministic points, as opposed to random points, that are evenly distributed over a set. The algorithm applies an inexpensive local search (steepest descent) on a set of quasirandom points to concentrate the sample. The sample is reduced replacing worse points with new quasirandom points. Any point that is retained for a certain number of iterations is used to start an efficient complete local search. The algorithm terminates when no new local minimum is found after several iterations. An experimental comparison shows that the method performs favorably with respect to other global optimization procedures.

Hagen and Kahng [20] implement an adaptive multi-start method for a VLSI partitioning optimization problem where the objective is to minimize the number of signals sent between components. The method consists of two phases: (1) generate a set of random starting points and perform the iterative (local search) algorithm, thus determining a set of local minimum solutions and (2) construct adaptive starting points derived from the best local minimum solutions found so far. The authors add a preprocessing cluster module to reduce the size of the problem. The resulting clustering adaptive multi-start method (CAMS) is fast and stable and improves upon previous partitioning results reported in the literature.

Tu and Mayne [41] describe a multi-start with a clustering strategy for constrained optimization problems. It is based on the characteristics of non-linear constrained global optimization problems and extends a strategy previously tested on unconstrained problems. In this study, variations of multi-start with clustering are considered including a simulated annealing procedure for sampling the design domain and a quadratic programming (QP) sub-problem for cluster formation. The strategies are evaluated by solving 18 non-linear mathematical problems and 6 engineering design problems. Numerical results show that the solution of a one-step QP sub-problem helps predict possible regions of attraction of local minima and can enhance robustness and effectiveness in identifying local minima without sacrificing efficiency. In comparison with other multi-start techniques, the strategies of this study are superior in terms of the number of local searches performed, the number of minima found, and the number of function evaluations required.

Bronmo et al. [6] present a multi-start local search heuristic for a typical ship scheduling problem. Their method generates a large number of initial solutions with an insertion heuristic partially based on random elements. The best initial solutions are improved by a local search heuristic that is split into a quick and an extended

version. The quick local search is used to improve a given number of the best initial solutions. The extended local search heuristic is then used to further improve some of the best solutions found. The multi-start local search heuristic is compared with an optimization-based solution approach with respect to computation time and solution quality. The computational study shows that the multi-start local search method consistently returns optimal or near-optimal solutions to real-life instances of the ship scheduling problem within a reasonable amount of CPU time.

### 9.2.5 Hybrid Designs

Ulder et al. [42] combine genetic algorithms with local search strategies to improve previous genetic approaches for the traveling salesman problem. They apply an iterative algorithm to improve each individual, either before or while being combined with other individuals to form a new solution (offspring). The combination of these three elements: *Generation*, *Combination*, and *Local Search*, extends the paradigm of re-start and establishes links with other metaheuristics such as scatter search [15] or memetic algorithms [31].

Mezmaz et al. [29] hybridize the multi-start framework with a model in which several evolutionary algorithms run simultaneously and cooperate to compute better solutions (called *island model*). They propose a solving method in the context of multi-objective optimization on the computational grid. The authors point out that although the combination of these two models usually provides very effective parallel algorithms, experiments on large-size problem instances are often stopped before convergence is achieved. The full exploitation of the cooperation model needs a large amount of computational resources and the management of the fault tolerance issue. In this chapter, a grid-based fault-tolerant approach for these models and their implementation on the *XtremWeb grid middleware* is proposed. The approach has been experimented on the bi-objective flow-shop problem on a computational grid made of 321 heterogeneous Linux PCs within a multi-domain education network. The preliminary results, obtained after an execution time of several days, demonstrate that the use of grid computing effectively and efficiently exploits the two parallel models and their combination for solving challenging optimization problems. In particular, the effectiveness is improved by over 60% when compared with a serial metaheuristic..

An open question in order to design a good search procedure is whether it is better to implement a simple improving method that allows a great number of global iterations or, alternatively, to apply a complex routine that significantly improves a few generated solutions. A simple procedure depends heavily on the initial solution but a more elaborate method takes much more running time and therefore can only be applied a few times, thus reducing the sampling of the solution space. Some metaheuristics, such as GRASP, launch limited local searches from numerous constructions (i.e., starting points). In most tabu search implementations, the search starts from one initial point and if a restarting procedure is also part of the method,

it is invoked only a limited number of times. However, the inclusion of re-starting strategies within the tabu search framework has been well documented in several papers (see, for example, [13] and [18]). In [27] the balance between restarting and search depth (i.e., the time spent searching from a single starting point) is studied in the context of the Bandwidth Matrix Problem. They tested both alternatives and concluded that it was better to invest the CPU time to search from a few starting points than re-starting the search more often. Although we cannot draw a general conclusion from these experiments, the experience in the current context and in previous projects indicates that some metaheuristics, like tabu search, need to reach a critical search depth to be effective. If this search depth is not reached, the effectiveness of the method is severely compromised.

### 9.3 A Classification

We have found three key elements in multi-start methods that can be used for classification purposes: memory, randomization, and degree of rebuild. The choices for each one of these elements are not restricted to the extreme cases where the element is simply present or absent, but represent a whole continuum between the extremes that can be labeled as

- *Memory/memoryless*
- *Systematic/randomized*
- *Rebuild/build-from-scratch*

The **Memory** classification refers to elements that are common to certain previously generated solutions. As in the tabu search framework [18], such memory provides a foundation for incentive-based learning, by means of incentives that reinforce actions leading to good solutions or deterrents that discourage actions leading to bad ones. Thus, instead of simply resorting to randomized re-starting processes, in which the current decisions do not get any benefit from the knowledge accumulated during prior search, specific types of information are identified to exploit history. On the other hand, memory avoidance (via the *memoryless* classification) is employed in a variety of methods where the construction of unconnected solutions is viewed as a means of strategically sampling the solution space. It should be noted that memory is not restricted to recording good solutions (or attributes of these solutions) but also includes recording solutions that exhibit diversity.

Starting solutions can be randomly generated or, on the contrary, they can be generated in a systematic way. **Randomization** is a very simple way of achieving diversification, but with no control over the diversity achieved since in some cases randomization can obtain very similar solutions. Moreover, there are a variety of forms of diversity that can be more important for conducting an effective search process than the haphazard outcomes of randomization. More systematic mechanisms are available to control the similarities among solutions produced, as a way to yield outcomes exhibiting a useful range of structural differences. Between the

extremes of *randomized* and *systematic* (or deterministic) generation of solutions lie a significant number of possibilities. These can range from imposing deterministic controls on a randomized process to joining randomized and deterministic processes in various forms of alternation. The GRASP method discussed later combines several of these intermediate possibilities.

The **Degree of Rebuild** is a measure of the number or proportion of elements that remain fixed from one generation to another. Most applications *build* the solution at each generation *from scratch*, but some strategies fix (or lock-in) some elements during the construction process that have appeared in previously generated solutions. Such an approach was proposed in the context of identifying and then iteratively exploiting strongly determined and consistent variables in [13]. This selective way of fixing elements, by reference to their previous impact and frequency of occurrence in various solution classes, is a memory-based strategy of the type commonly used in tabu search. This type of approach is also implicit in the operation of path-relinking [17] which generates new solutions by exploring trajectories that connect high-quality solutions. In this case the process seeks to incorporate the attributes of previously generated elite solutions by creating inducements to favor these attributes in currently generated solutions. In an extreme case all the elements in the new solution will be determined (and fixed) by the information generated from the set of elite solutions considered. This is labeled as (complete) Rebuild.

## 9.4 The Maximum Diversity Problem

The problem of choosing a subset of elements with maximum diversity from a given set is known as the Maximum Diversity Problem (MDP). This problem has a wide range of practical applications involving fields such as medical treatments, environmental balance, immigration policies, and genetic engineering [16]. The MDP has been studied by numerous authors, most prominent among them being Kuo et al. [23], who described four formulations of the problem, ranging from the most intuitive to the most efficient, and which also served to show that the MDP is NP hard. In 1996, Ghosh [12] proposed a multi-start method and proved the completeness of the problem. Later, Glover et al. [16] proposed four deterministic heuristic methods, two of them constructive and the other two destructive. Silva et al. [39] presented a multi-start algorithm based on the GRASP methodology. Specifically, they described three constructive methods, called KLD, KLDv2, and MDI, and two improvement methods: LS, which is an adaptation of the one proposed by Ghosh, and SOMA, based on a VNS implementation.

The MDP can be formally described as a combinational optimization problem which can be stated as follows: let  $S = \{s_i : i \in N\}$  be a set of elements where  $N = \{1, 2, \dots, n\}$  is the set of indexes. Each element of the set  $s_i \in S$  may be represented by a vector  $s_i = (s_{i_1}, s_{i_2}, \dots, s_{i_r})$ . Let  $d_{ij}$  be the distance between two elements  $s_i$  and  $s_j$  and let  $m$  (with  $m < n$ ) be the desired size of the maximum diversity set. Within this context, the solution of the MDP consists in finding a subset  $Sel$  of  $m$  elements of  $S$

( $Sel \subset S$  and  $|Sel| = m$ ) in order to maximize the sum of the distances between the selected elements. Mathematically, the MDP may be rewritten as a decision problem in the following terms:

$$\begin{aligned} \max \quad & z = \sum_{i < j} d_{ij} x_i x_j \\ \text{subject to} \quad & \sum_{i=1}^n x_i = m \\ & x_i \in \{0, 1\} \quad i = 1, \dots, n \end{aligned}$$

where  $x_i = 1$  indicates that element  $s_i$  has been selected.

Two constructive algorithms are proposed in order to use a multi-start scheme to solve the MDP, one of them with memory and the other without. Each algorithm is described in turn in the following sections.

#### 9.4.1 Multi-start Without Memory (MSWoM)

The Multi-start Without Memory (MSWoM) algorithm consists of a GRASP-based constructive procedure and a first improvement local search. This approach was inspired by a heuristic method proposed in Glover et al. [16]. In each step, the constructive procedure adds a high-quality element (given by a greedy function) to the set  $Sel$ . The non-selected elements are contained in the set  $S - Sel$ . The set  $Sel$  is initially empty, meaning that all the elements might be selected. The algorithm starts by selecting an element from  $S$  at random and placing it in the set  $Sel$ . The distance from all the non-selected elements  $s_i \in S - Sel$  to  $Sel$  is then computed as follows:

$$d(s_i, Sel) = \sum_{s_j \in Sel} d(s_i, s_j), \quad (9.1)$$

which serves to arrange all the non-selected elements. To select the next element for inclusion in the set  $Sel$ , an ordered list  $L$  is constructed with all the elements  $s_i \in S - Sel$  at a percent  $\alpha$  of the maximum distance. Mathematically,  $L$  is defined as follows:

$$L = \{s_i \in S - Sel / d(s_i, Sel) \geq d_{\min} + \alpha(d_{\max} - d_{\min})\}, \quad (9.2)$$

where

$$d_{\max} = \max_{s_i \in S - Sel} d(s_i, Sel) \quad d_{\min} = \min_{s_i \in S - Sel} d(s_i, Sel)$$

The next element introduced in set  $Sel$  is chosen at random from among the elements in  $L$  so as to ensure it has a minimum quality percentage, set by  $\alpha$ . So, it is

not a purely greedy selection which would lead to a local optimum. This procedure is repeated until  $m$  elements have been chosen ( $|Sel| = m$ ) such that  $Sel$  contains the solution to the problem. After  $niter$  executions of the process, the arithmetic mean of the  $niter$  solutions will typically be worse than if the solution had been constructed by taking the element with a maximum distance over those already selected, although some of the  $niter$  solutions will probably improve on this value.

For the algorithm to have a reactive behavior, the parameter  $\alpha$  is initially set at 0.5 and then adjusted dynamically depending on the quality of the solutions obtained; that is, if after  $niter/5$  consecutive iterations, the best solution has not improved, then  $\alpha$  is increased by 0.1 (up to a maximum of 0.9).

The improvement method is based on a simplification of the local search described in [12], which seeks to increase the efficiency of the local search. The proposed method is classified as a first improvement local search which, as described in [25], not only tends to yield better results than the best improvement strategies but also requires much less time. It does so by factoring the contribution from each element  $s_i$  in  $Sel$ ; that is, for each element  $s_i \in Sel$ , its contribution  $d_i$  to the objective function is as follows:

$$d_i = \sum_{s_j \in Sel} d_{ij} = d(s_i, Sel), \quad (9.3)$$

with the objective function defined as

$$z = \frac{1}{2} \sum_{s_i \in Sel} d_i. \quad (9.4)$$

Subsequently, the element  $s_{i^*} \in Sel$  with the lowest contribution to the current solution is selected; that is, the element  $s_{i^*} \in Sel$  with the lowest value of  $d_{i^*}$ , such that  $s_{i^*} \in Sel$  is exchanged with the first element  $s_j \in S - Sel$  that increases the value of the objective function (where the elements in  $S - Sel$  are examined in lexicographical order). The search procedure continues for as long as the objective function improves by extracting the element from the set  $Sel$  which contributes the least and inserting another from  $S - Sel$  which improves the value of the objective function. When there is no improvement, the second least contributing element is used and so on. This procedure is continued until no further improvement is obtained.

#### 9.4.2 Multi-start with Memory (MSWM)

Multi-start with memory (MSWM) is the second multi-start algorithm described in [8]. The method uses memory both in the solution construction and improvement phases. These strategies are integrated within the tabu search methodology [18].

In each iteration, the constructive algorithm penalizes the frequency of use of those elements which appeared in previous solutions. The procedure also rewards those elements which previously appeared in high-quality solutions. To implement this algorithm, the number of times element  $s_i$  was selected in previous constructions

is stored in  $freq[i]$ . The maximum value of  $freq[i]$  for all  $i$  is stored in  $maxfreq$ . The average value of the solutions in which element  $s_i$  has appeared is stored in  $quality[i]$ . In addition,  $max_q$  stores the maximum value of  $quality[i]$  for all  $i$ . The evaluation of each non-selected element in the current construction is modified depending on these values, thus favoring the selection of low-frequency, high-quality elements. This is achieved by using the following expression instead of the distance described in Equation (9.3) between an element and the set of selected elements:

$$d'(s_i, Sel) = d(s_i, Sel) - \beta range(Sel) \frac{freq[i]}{max\_freq} + \delta range(Sel) \frac{quality[i]}{max\_q}$$

with

$$range(Sel) = \max_{s_j \in S - Sel} d(s_j, Sel) - \min_{s_j \in S - Sel} d(s_j, Sel),$$

where  $\beta$  and  $\delta$  are parameters that quantify the contributions of the frequency penalization and the reward for quality. Both are adjusted experimentally. The purpose of the  $range(Sel)$  parameter is to smooth the changes in the penalty function.

The set  $Sel$  is initially empty, meaning any element can be selected. The algorithm starts by selecting an element from  $S$  at random and inserting it in the set  $Sel$ . It then computes the distance  $d'(s_i, Sel)$  for each element  $s_i \in S - Sel$ , which in the first construction would correspond with  $d(s_i, Sel)$ , since  $freq[i] = quality[i] = 0$ . The chosen element  $i^*$  is the one given as

$$d'(s_{i^*}, Sel) = \max_{s_i \in S} \{d'(s_i, Sel)\}.$$

It is then inserted in  $Sel$ , after which the frequency vector is updated. This procedure is repeated until  $m$  elements have been chosen. Once a solution is constructed, the quality vector is updated. The tabu multi-start method executes this procedure  $niter$  times, in such a way that with each construction the distances between an element and the set of those already selected are updated depending on its past history.

The improvement method is a modification of the one described above with the added feature of a short-term memory based on the exchange of an element between  $Sel$  and  $S - Sel$ . One iteration of this algorithm consists of randomly selecting an element  $s_i \in Sel$ . The probability of selecting this element is inversely proportional to its associated  $d_i$  value. That element of  $Sel$  is substituted by the first element  $s_j \in S - Sel$  which improves the value of the objective function. If this element does not exist, then the one which degrades the least objective function is chosen (in such a way that an exchange is always performed). When this exchange is carried out, both  $s_i$  and  $s_j$  take on a tabu status for  $TabuTenure$  iterations. Consequently, it is forbidden to remove element  $s_i$  from set  $Sel$  (respectively, element  $s_j$  from set  $S - Sel$ ) for that number of iterations. The tabu search process continues until  $MaxIter$  consecutive iterations are executed without improving the best value obtained thus far.

### 9.4.3 Experimental Results

To illustrate the behavior of the two multi-start algorithms summarized in this chapter and proposed in [8], we present a comparison with two other previously reported algorithms. Specifically, the MSWoM and MSWM algorithms are compared with the D2 constructive algorithm, proposed in [16] along with the improvement method described in [12], and the KLDv2 algorithm and its respective improvement procedure, presented in [39], which represent the best methods for this problem. All the algorithms were coded in C and compiled with Borland Builder 5.0, optimized for maximum speed. The experiments were carried out on a 3-GHz Pentium IV with 1 GB RAM.

The algorithms were executed on three sets of instances:

1. **Silva:** 20  $n \times n$  matrices with random integer values generated from a  $[0, 9]$  uniform distribution with  $n \in [100, 500]$  and  $m \in [0.1n, 0.4n]$ .
2. **Glover:** 20  $n \times n$  matrices in which the values are the distances between each pair of points with Euclidean coordinates randomly generated in  $[0, 10]$ . These  $n$  points have  $r$  coordinates, with  $r \in [2, 21]$ .
3. **Random:** 20  $n \times n$  matrices with real weights generated from a  $(0, 10)$  uniform distribution with  $n = 2000$  and  $m = 200$ . It should be noted that these were the largest problem instances solved in the references consulted.

Tables 9.1, 9.2, and 9.3 compare MSWoM, MSWM, and D2 + LS, and KLDv2+LS. These tables show the average percentage deviation for each procedure with respect to the best known solution (in each experiment, since the optimal values are unknown), the number of best solutions and the number of constructions and improvements made by the algorithm in 10 s (stopping criterion).

**Table 9.1** Constructive methods—*Silva*-type examples.

	D2 + LS	KLDv2 + LS	MSWoM	MSWM
Dev. (%)	1.722	1.079	0.0377	0.0130
# Best	2	5	12	13
# Const.	5140.5	5	12	13

**Table 9.2** Constructive methods—*Glover*-type examples.

	D2 + LS	KLDv2 + LS	MSWoM	MSWM
Dev. (%)	0.018	0.006	0.000	0.000
# Best	16	18	20	20
# Const.	2149.6	971.0	790.4	397.5

**Table 9.3** Constructive methods—*Random*-type examples.

	D2 + LS	KLDv2 + LS	MSWoM	MSWM
Dev. (%)	1.270	1.219	0.204	0.099
# Best	0	0	7	15
# Const.	128.1	3.5	12	14.8

The conclusion that may be drawn from these tables is that the proposed multi-start methods substantially improve on previous algorithms, with regard to both the deviation from the best known values and the number of times that value is found. Moreover, the experiments conducted also show that the use of memory, at least for the instances tested, leads to better results. Note that in the case of *Glover*-type examples, the algorithms studied yield very similar values. This fact indicates that these are the simplest problem instances and consequently say little about the quality of each algorithm. At the other extreme are the *Random*-type examples, where substantial improvements are obtained with the multi-start methods.

## 9.5 Conclusion

The objective of this study was to extend and advance the knowledge multi-start methods. Unlike other well-known methods, these procedures have not yet become widely implemented and tested as a metaheuristic themselves for solving complex optimization problems. We have shown new ideas that have recently emerged within the multi-start area that add a clear potential to this framework which has yet to be fully explored.

Our findings disclose the fact that memory appears to play an important role during both the constructive and the improvement phase of a multi-start procedure. This effect may be due to the fact that the repeated application of the constructive phase operates primarily as a diversification process, and the introduction of memory structures guides the diversification in an efficient way. On the other hand, the benefits associated with the inclusion of memory structures in the local search (improvement phase) have been extensively documented in the tabu search literature. Our results with the Maximum Diversity Problem are in line with these previous references. The comparison between memory-based and memory-less designs provides an interesting area for future research.

**Acknowledgments** The authors want to thank Prof. Fred Glover for his valuable comments to improve this chapter in both presentation and content. This research was partially supported by the *Ministerio de Educación y Ciencia* (TIN2006-02696, TIN2009-07516, TIN2009-13363), by the *Comunidad de Madrid-Universidad Rey Juan Carlos* project (CCG08-URJC/TIC-3731), and by the *Gobierno de Canarias* project (PI2007/019).

## References

1. Ackley, D.H.: An empirical study of bit vector function optimization. In: Davis, L. (ed.) *Genetic Algorithms and Simulated Annealing*, pp. 170–204. Morgan Kaufmann, Pitman, London (1987)
2. Baluja, S.: An Empirical Comparison of 7 iterative evolutionary function optimization heuristics. School of computer science, Carnegie Mellon University, Pittsburgh, PA (1995)

3. Beausoleil, R.P., Baldoquin, G., Montejo, R.: A Multi-start and path relinking methods to deal with multiobjective knapsack problems. *Ann Oper. Res.* **157**, 105–133 (2008)
4. Binato, S., Faria, Jr. H., Resende, M.G.C.: Greedy randomized adaptive path relinking. In: MIC2001 Conference Proceedings, Porto, Portugal (2001)
5. Boese, K.D., Kahng, A.B., Muddu, S.: A new adaptive multi-start technique for combinatorial global optimisation. *Oper. Res. Lett.* **16**, 103–113 (1994)
6. Bronmo, G., Christiansen, M., Fagerholt, K., Nygreen, B.: In: Corne, D., Dorigo, M., Glover, F. (eds.) A multi-start local search heuristic for ship scheduling-a computational study. *Comput. Oper. Res.* **34**, 900–917 (2007)
7. Chiang, W.C., Russell, R.A.: Simulated annealing metaheuristics for the vehicle routing problems with time windows. *Ann. Oper. Res.* **60**, 3–27 (1995)
8. Duarte, A., Martí, R.: Tabu Search and GRASP for the Maximum Diversity Problem. *Eur. J. Oper. Res.* **178**, 71–84 (2007)
9. Feo, T., Resende, M.G.C.: A probabilistic heuristic for a computationally difficult set covering problem. *Oper. Res. Lett.* **8**, 67–71 (1989)
10. Feo, T., Resende, M.G.C.: Greedy randomized adaptive search procedures. *J. Global Optim.* **2**, 1–27 (1995)
11. Fleurent, C., Glover, F.: Improved constructive multi-start strategies for the quadratic assignment problem using adaptive memory. *INFORMS J. Comput.* **11**, 198–204 (1999)
12. Ghosh, J.B.: Computational aspects of the maximum diversity problem. *Oper. Res. Lett.* **19**, 175–181 (1996)
13. Glover, F.: Heuristics for integer programming using surrogate constraints. *Decis. Sci.* **8**, 156–166 (1977)
14. Glover, F.: Tabu search: a tutorial. *Interfaces* **20**, 74–94 (1990)
15. Glover, F.: Multi-start and strategic oscillation methods – principles to exploit adaptive memory. In: Laguna, M., Gonzalez-Velarde, J.L. (eds.) Computing Tools for Modeling Optimization and Simulation, pp. 1–25 Kluwer, Boston, MA (2000)
16. Glover, F., Kuo, C.C., Dhir, K.S.: Heuristic algorithms for the maximum diversity problem. *J. Inf. Optim. Sci.* **19**, 109–132 (1998)
17. Glover, F., Laguna, M.: Tabu Search, Modern Heuristics Techniques for Combinatorial Optimization Problems, In: Reeves, C. (ed.) pp. 70–141 Blackwell Scientific Publishing, Oxford (1993)
18. Glover, F., Laguna, M.: Tabu Search. Kluwer, Boston, MA (1997)
19. Glover, F., Laguna, M., Martí, R.: Scatter Search in Theory and Applications of Evolutionary Computation: Recent Trends, In: Ghosh, A., Tsutsui, S. (eds.), pp. 519–529. Springer, New York, NY (2002)
20. Hagen, L.W., Kahng, A.B.: Combining problem reduction and adaptive multi-start: a new technique for superior iterative partitioning. *IEEE Trans. CAD* **16**, 709–717 (1997)
21. Hickernell, F.J., Yuan, Y.: A simple multistart algorithm for global optimization. *OR Trans.* **1**, 1–11 (1997)
22. Hu, X., Shonkwiler, R., Spruill, M.C.: Random Restarts in Global Optimization. Georgia Institute of Technology, Atlanta (1994)
23. Kuo, C.C., Glover, F., Dhir, K.: Analyzing and modeling the maximum diversity problem by zero-one programming. *Decis. Sci.* **24**, 1171–1185 (1993)
24. Laguna, M., Feo, T., Elrod, H.: A greedy randomized adaptive search procedure for the 2-partition problem. *Oper. Res.* **42**, 677–687 (1994)
25. Laguna, M., Martí, R.: GRASP and Path relinking for 2-layer straight line crossing minimization. *INFORMS J. Comput.* **11**, 44–52 (1999)
26. Lokketangen, A., Glover, F.: Probabilistic move selection in tabu search for 0/1 mixed integer programming problems. In: Osman, I.H., Kelly, J.P. (eds.) Meta-Heuristics: Theory and Practice, pp. 467–488 Kluwer, Boston, MA (1996)
27. Martí, R., Laguna, M., Glover, F., Campos, V.: Reducing the bandwidth of a sparse matrix with tabu search. *Eur. J. Oper. Res.* **135**, 450–459 (2001)

28. Mayne, D.Q., Meewella, C.C.: A non-clustering multistart algorithm for global optimization. In: Bensoussan, A., Lions, J.L. (eds.) *Analysis and Optimization of Systems. Lecture Notes in Control and Information Science*, vol. 111, pp. 334–345. Springer, New York, NY (1988)
29. Mezmaz, M., Melab, N., Talbi, E.G.: Using the multi-start and island models for parallel multi-objective optimization on the computational grid. In: 2nd IEEE International Conference on e-Science and Grid Computing, Washington, DC, USA, Art. No. 4031085 (2006)
30. Moreno, J.A., Mladenovic, N., Moreno-Vega, J.M.: An statistical analysis of strategies for multistart heuristic searches for p-facility location-allocation problems. In: 8th Meeting of the EWG on Locational Analysis Lambrecht, Germany (1995)
31. Moscato, P.: Memetic algorithms. *New Ideas in Optimization*, In: Corne, D., Dorigo, M., Glover, F. (eds.) pp. 219–235 Mc Graw Hill, New York, NY (1999)
32. Patterson, R., Pirkul, H., Rolland, E.: Adaptive reasoning technique for the capacitated minimum spanning tree problem. *J. Heuristics* **5**, 159–180 (1999)
33. Prais, M., Ribeiro, C.C.: Reactive GRASP: an application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS J. Comput.* **12**, 164–176 (2000)
34. Resende, M.G.C.: Computing approximate solutions for the maximum covering problem using GRASP. *J. Heuristics* **4**, 161–171 (1998)
35. Resende, M.G.C., Martí, R., Gallego, M., Duarte, A.: GRASP and path relinking for the max-min diversity problem. *Comput. Oper. Res.* **37**, 498–508 (2010)
36. Rinnooy Kan, A.H.G., Timmer, G.T.: Global optimization. In: Rinnooy Kan, A.H.G., Todd, M.J. (eds.) *Handbooks in Operations Research and Management Science*, vol. 1, pp. 631–662. North Holland, Amsterdam (1998)
37. Rochat, Y., Taillard, R.D.: Probabilistic diversification and intensification in local search for vehicle routing. *J. Heuristics* **1**, 147–167 (1995)
38. Russell, R.A.: Hybrid heuristics for the vehicle routing problem with time windows. *Trans. Sci.* **29**, 156–166 (1995)
39. Silva, G.C., Ochi, L.S., Martins, S.L.: Experimental comparison of greedy randomized adaptive search procedures for the maximum diversity problem. *Lect. Notes Comput. Sci.* **3059**, 498–512 (2004)
40. Solis, F., Wets, R.: Minimization by random search techniques. *Math. Oper. Res.* **6**, 19–30 (1981)
41. Tu, W., Mayne, R.W.: An approach to multi-start clustering for global optimization with nonlinear constraints. *Int. J. Numer. Methods Eng.* **53**, 2253–2269 (2002)
42. Ulder, N.L.J., Aarts, E.H.L., Bandelt, H.J., Van Laarhoven, P.J.M., Pesch, E.: Genetic local search algorithms for the traveling salesman problem. In: Schwefel, H.P., Männer, R. (eds.) *Parallel Problem Solving from Nature*, pp. 109–116 Springer (1990)
43. Wattenberg, M., Juels, A.: Stochastic hillclimbing as a baseline method for evaluating genetic algorithms. University of California at Berkeley, CSD94-834 (1994)



# Chapter 10

## Greedy Randomized Adaptive Search Procedures: Advances, Hybridizations, and Applications

Mauricio G.C. Resende and Celso C. Ribeiro

**Abstract** GRASP is a multi-start metaheuristic for combinatorial optimization problems, in which each iteration consists basically of two phases: construction and local search. The construction phase builds a feasible solution, whose neighborhood is investigated until a local minimum is found during the local search phase. The best overall solution is kept as the result. In this chapter, we first describe the basic components of GRASP. Successful implementation techniques are discussed and illustrated by numerical results obtained for different applications. Enhanced or alternative solution construction mechanisms and techniques to speed up the search are also described: alternative randomized greedy construction schemes, Reactive GRASP, cost perturbations, bias functions, memory and learning, local search on partially constructed solutions, hashing, and filtering. We also discuss implementation strategies of memory-based intensification and post-optimization techniques using path-relinking. Hybridizations with other metaheuristics, parallelization strategies, and applications are also reviewed.

### 10.1 Introduction

We consider in this chapter a combinatorial optimization problem, defined by a finite ground set  $E = \{1, \dots, n\}$ , a set of feasible solutions  $F \subseteq 2^E$ , and an objective function  $f : 2^E \rightarrow \mathbb{R}$ . In its minimization version, we search an optimal solution  $S^* \in F$  such that  $f(S^*) \leq f(S)$ ,  $\forall S \in F$ . The ground set  $E$ , the cost function  $f$ , and the set of feasible solutions  $F$  are defined for each specific problem. For instance,

---

Mauricio G.C. Resende  
AT&T Labs Research, Florham Park, NJ 07932 USA  
e-mail: mgcr@research.att.com

Celso C. Ribeiro  
Universidade Federal Fluminense, Niterói, RJ 22410-240 Brazil  
e-mail: celso@ic.uff.br

in the case of the traveling salesman problem, the ground set  $E$  is that of all edges connecting the cities to be visited,  $f(S)$  is the sum of the costs of all edges in  $S$ , and  $F$  is formed by all edge subsets that determine a Hamiltonian cycle.

GRASP (greedy randomized adaptive search procedure) [68, 69] is a multistart or iterative metaheuristic, in which each iteration consists of two phases: construction and local search. The construction phase builds a solution. If this solution is not feasible, then it is necessary to apply a repair procedure to achieve feasibility. Once a feasible solution is obtained, its neighborhood is investigated until a local minimum is found during the local search phase. The best overall solution is kept as the result. Extensive literature surveys are presented in [78–80, 156, 157, 160]. The pseudo-code in Figure 10.1 illustrates the main blocks of a GRASP procedure for minimization, in which `Max_Iterations` iterations are performed and `Seed` is used as the initial seed for the pseudo-random number generator.

```

procedure GRASP(Max_Iterations, Seed)
1   Read_Input();
2   for  $k = 1, \dots, \text{Max\_Iterations}$  do
3        $\text{Solution} \leftarrow \text{Greedy\_Randomized\_Construction}(\text{Seed});$ 
4       if  $\text{Solution}$  is not feasible then
5            $\text{Solution} \leftarrow \text{Repair}(\text{Solution});$ 
6       end;
7        $\text{Solution} \leftarrow \text{Local\_Search}(\text{Solution});$ 
8        $\text{Update\_Solution}(\text{Solution}, \text{Best\_Solution});$ 
9   end;
10  return Best_Solution;
end GRASP.

```

**Fig. 10.1** Pseudo-code of the GRASP metaheuristic.

Figure 10.2 illustrates the construction phase with its pseudo-code. At each iteration of this phase, let the set of candidate elements be formed by all elements of the ground set  $E$  that can be incorporated into the partial solution being built, without impeding the construction of a feasible solution with the remaining ground set elements. The selection of the next element for incorporation is determined by the evaluation of all candidate elements according to a greedy evaluation function. This greedy function usually represents the incremental increase in the cost function due to the incorporation of this element into the solution under construction. The evaluation of the elements by this function leads to the creation of a restricted candidate list (RCL) formed by the best elements, i.e., those whose incorporation to the current partial solution results in the smallest incremental costs (this is the greedy aspect of the algorithm). The element to be incorporated into the partial solution is randomly selected from those in the RCL (this is the probabilistic aspect of the heuristic). Once the selected element is incorporated into the partial solution, the candidate list is updated and the incremental costs are reevaluated (this is the adaptive aspect of the heuristic). The above steps are repeated while there exists at least one candidate element. This strategy is similar to the semi-greedy heuristic

```

procedure Greedy_Randomized_Construction(Seed)
1   Solution  $\leftarrow \emptyset$ ;
2   Initialize the set of candidate elements;
3   Evaluate the incremental costs of the candidate elements;
4   while there exists at least one candidate element do
5       Build the restricted candidate list (RCL);
6       Select an element  $s$  from the RCL at random;
7       Solution  $\leftarrow$  Solution  $\cup \{s\}$ ;
8       Update the set of candidate elements;
9       Reevaluate the incremental costs;
10  end;
11  return Solution;
end Greedy_Randomized_Construction.

```

**Fig. 10.2** Pseudo-code of the construction phase.

proposed by Hart and Shogan [97], which is also a multi-start approach based on greedy randomized constructions, but without local search.

Not always is a randomized greedy construction procedure able to produce a feasible solution. In case this occurs, it may be necessary to apply a repair procedure to achieve feasibility. Examples of repair procedures can be found in [60, 61, 129].

The solutions generated by a greedy randomized construction are not necessarily optimal, even with respect to simple neighborhoods. The local search phase usually improves the constructed solution. A local search algorithm works in an iterative fashion by successively replacing the current solution by a better solution in its neighborhood. It terminates when no better solution is found in the neighborhood. The pseudo-code of a basic local search algorithm starting from the solution Solution constructed in the first phase (and possibly made feasible by the repair heuristic) and using a neighborhood  $N$  is given in Figure 10.3.

```

procedure Local_Search(Solution)
1   while Solution is not locally optimal do
2       Find  $s' \in N(\text{Solution})$  with  $f(s') < f(\text{Solution})$ ;
3       Solution  $\leftarrow s'$ ;
4   end;
5   return Solution;
end Local_Search.

```

**Fig. 10.3** Pseudo-code of the local search phase.

The speed and the effectiveness of a local search procedure depend on several aspects, such as the neighborhood structure, the neighborhood search technique, the strategy used for the evaluation of the cost function value at the neighbors, and the starting solution itself. The construction phase plays a very important role with respect to this last aspect, building high-quality starting solutions for the local search. Simple neighborhoods are usually used. The neighborhood search may be implemented using either a *best-improving* or a *first-improving* strategy. In the case of the

best-improving strategy, all neighbors are investigated and the current solution is replaced by the best neighbor. In the case of a first-improving strategy, the current solution moves to the first neighbor whose cost function value is smaller than that of the current solution. In practice, we observed on many applications that quite often both strategies lead to the same final solution, but in smaller computation times when the first-improving strategy is used. We also observed that premature convergence to a bad local minimum is more likely to occur with a best-improving strategy.

## 10.2 Construction of the Restricted Candidate List

An especially appealing characteristic of GRASP is the ease with which it can be implemented. Few parameters need to be set and tuned. Therefore, development can focus on implementing appropriate data structures for efficient construction and local search algorithms. GRASP has two main parameters: one related to the stopping criterion and the other to the quality of the elements in the restricted candidate list.

The stopping criterion used in the pseudo-code described in Figure 10.1 is determined by the number `Max_Iterations` of iterations. Although the probability of finding a new solution improving the incumbent (current best solution) decreases with the number of iterations, the quality of the incumbent may only improve with the latter. Since the computation time does not vary much from iteration to iteration, the total computation time is predictable and increases linearly with the number of iterations. Consequently, the larger the number of iterations, the larger will be the computation time and the better will be the solution found.

For the construction of the RCL used in the first phase we consider, without loss of generality, a minimization problem as the one formulated in Section 10.1. We denote by  $c(e)$  the incremental cost associated with the incorporation of element  $e \in E$  into the solution under construction. At any GRASP iteration, let  $c^{\min}$  and  $c^{\max}$  be, the smallest and the largest incremental costs, respectively.

The restricted candidate list RCL is made up of the elements  $e \in E$  with the best (i.e., the smallest) incremental costs  $c(e)$ . This list can be limited either by the number of elements (cardinality based) or by their quality (value based). In the first case, it is made up of the  $p$  elements with the best incremental costs, where  $p$  is a parameter. In this chapter, the RCL is associated with a threshold parameter  $\alpha \in [0, 1]$ . The restricted candidate list is formed by all elements  $e \in E$  which can be inserted into the partial solution under construction without destroying feasibility and whose quality is superior to the threshold value, i.e.,  $c(e) \in [c^{\min}, c^{\min} + \alpha(c^{\max} - c^{\min})]$ . The case  $\alpha = 0$  corresponds to a pure greedy algorithm, while  $\alpha = 1$  is equivalent to a random construction. The pseudo-code in Figure 10.4 is a refinement of the greedy randomized construction pseudo-code shown in Figure 10.2. It shows that the parameter  $\alpha$  controls the amounts of greediness and randomness in the algorithm.

```

procedure Greedy_Randomized_Construction( $\alpha$ , Seed)
1   Solution  $\leftarrow \emptyset$ ;
2   Initialize the candidate set:  $C \leftarrow E$ ;
3   Evaluate the incremental cost  $c(e)$  for all  $e \in C$ ;
4   while  $C \neq \emptyset$  do
5      $c^{min} \leftarrow \min\{c(e) \mid e \in C\}$ ;
6      $c^{max} \leftarrow \max\{c(e) \mid e \in C\}$ ;
7     RCL  $\leftarrow \{e \in C \mid c(e) \leq c^{min} + \alpha(c^{max} - c^{min})\}$ ;
8     Select an element  $s$  from the RCL at random;
9     Solution  $\leftarrow$  Solution  $\cup \{s\}$ ;
10    Update the candidate set  $C$ ;
11    Reevaluate the incremental cost  $c(e)$  for all  $e \in C$ ;
12  end;
13  return Solution;
end Greedy_Randomized_Construction.

```

**Fig. 10.4** Refined pseudo-code of the construction phase.

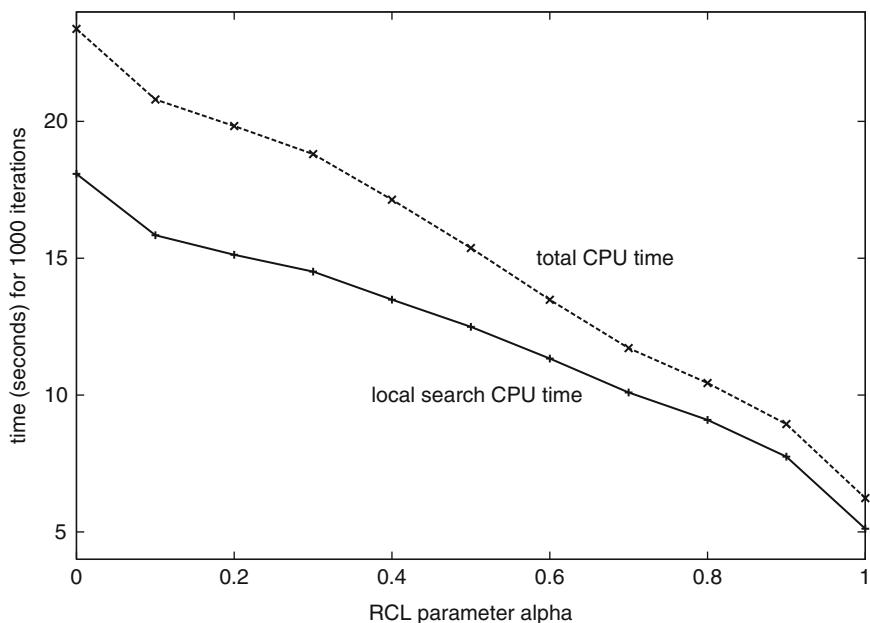
GRASP may be viewed as a repetitive sampling technique. Each iteration produces a sample solution from an unknown distribution, whose mean and variance are functions of the restrictive nature of the RCL. For example, if the RCL is restricted to a single element, then the same solution will be produced at all iterations. The variance of the distribution will be zero and the mean will be equal to the value of the greedy solution. If the RCL is allowed to have more elements, then many different solutions will be produced, implying a larger variance. Since greediness plays a smaller role in this case, the average solution value should be worse than that of the greedy solution. However, the value of the best solution found outperforms the average value and very often is optimal. It is unlikely that GRASP will find an optimal solution if the average solution value is high, even if there is a large variance in the overall solution values. On the other hand, if there is little variance in the overall solution values, it is also unlikely that GRASP will find an optimal solution, even if the average solution is low. What often leads to good solutions are relatively low average solution values in the presence of a relatively large variance, such as is the case for  $\alpha = 0.2$ .

Another interesting observation is that the distances between the solutions obtained at each iteration and the best solution found increase as the construction phase moves from more greedy to more random. This causes the average time taken by the local search to increase. Very often, many GRASP solutions may be generated in the same amount of time required for the local search procedure to converge from a single random start. In these cases, the time saved by starting the local search from good initial solutions can be used to improve solution quality by performing more GRASP iterations.

These results are illustrated in Table 10.1 and Figure 10.5, for an instance of the MAX-SAT problem where 1000 iterations were run. For each value of  $\alpha$  ranging from 0 (purely random construction for maximization problems) to 1 (purely greedy construction for maximization problems), we give in Table 10.1 the average Hamming distance between each solution built during the construction phase and

**Table 10.1** Average number of moves and local search time as a function of the RCL parameter  $\alpha$  for a maximization problem.

$\alpha$	Average distance	Average moves	Local search time (s)	Total time (s)
0.0	12.487	12.373	18.083	23.378
0.1	10.787	10.709	15.842	20.801
0.2	10.242	10.166	15.127	19.830
0.3	9.777	9.721	14.511	18.806
0.4	9.003	8.957	13.489	17.139
0.5	8.241	8.189	12.494	15.375
0.6	7.389	7.341	11.338	13.482
0.7	6.452	6.436	10.098	11.720
0.8	5.667	5.643	9.094	10.441
0.9	4.697	4.691	7.753	8.941
1.0	2.733	2.733	5.118	6.235



**Fig. 10.5** Total CPU time and local search CPU time as a function of the RCL parameter  $\alpha$  for a maximization problem (1000 repetitions for each value of  $\alpha$ ).

the corresponding local optimum obtained after local search, the average number of moves from the first to the latter, the local search time in seconds, and the total processing time in seconds. Figure 10.5 summarizes the values observed for the total processing time and the local search time. We notice that both time measures considerably decrease as  $\alpha$  tends to 1, approaching the purely greedy choice. In particular, we observe that the average local search time taken by  $\alpha = 0$  (purely random) is approximately 2.5 times that taken in the case  $\alpha = 0.9$  (almost greedy). In this example, two to three greedily constructed solutions can be investigated in the

same time needed to apply local search to one single randomly constructed solution. The appropriate choice of the value of the RCL parameter  $\alpha$  is clearly critical and relevant to achieve a good balance between computation time and solution quality.

Prais and Ribeiro [142] have shown that using a single fixed value for the value of the RCL parameter  $\alpha$  very often hinders finding a high-quality solution, which could be found if another value was used. They proposed an extension of the basic GRASP procedure, which they call *Reactive* GRASP, in which the parameter  $\alpha$  is self-tuned and its value is periodically modified accordingly to the quality of the solutions obtained along the search. In particular, computational experiments on the problem of traffic assignment in communication satellites [143] have shown that Reactive GRASP found better solutions than the basic algorithm for many test instances. These results motivated the study of the behavior of GRASP for different strategies for the variation of the value of the RCL parameter  $\alpha$ :

- (R)  $\alpha$  self-tuned according to the Reactive GRASP procedure;
- (E)  $\alpha$  randomly chosen from a uniform discrete probability distribution;
- (H)  $\alpha$  randomly chosen from a decreasing non-uniform discrete probability distribution; and
- (F) fixed value of  $\alpha$ , close to the purely greedy choice.

We summarize the results obtained by the experiments reported in [141, 142]. These four strategies were incorporated into the GRASP procedures developed for four different optimization problems: (P-1) matrix decomposition for traffic assignment in communication satellite [143], (P-2) set covering [68], (P-3) weighted MAX-SAT [153, 154], and (P-4) graph planarization [155, 161]. Let  $\Psi = \{\alpha_1, \dots, \alpha_m\}$  be the set of possible values for the parameter  $\alpha$  for the first three strategies. The strategy for choosing and self-tuning the value of  $\alpha$  in the case of the Reactive GRASP procedure (R) is described in Section 10.3. In the case of the strategy (E) based on using the discrete uniform distribution, all choice probabilities are equal to  $1/m$ . The third case corresponds to the a hybrid strategy (H), in which the authors considered  $p(\alpha = 0.1) = 0.5$ ,  $p(\alpha = 0.2) = 0.25$ ,  $p(\alpha = 0.3) = 0.125$ ,  $p(\alpha = 0.4) = 0.03$ ,  $p(\alpha = 0.5) = 0.03$ ,  $p(\alpha = 0.6) = 0.03$ ,  $p(\alpha = 0.7) = 0.01$ ,  $p(\alpha = 0.8) = 0.01$ ,  $p(\alpha = 0.9) = 0.01$ , and  $p(\alpha = 1.0) = 0.005$ . Finally, in the last strategy (F), the value of  $\alpha$  is fixed as recommended in the original references of problems P-1 to P-4 cited above, where this parameter was tuned for each problem. A subset of the literature instances was considered for each class of test problems. The results reported in [142] are summarized in Table 10.2. For each problem, we first list the number of instances considered. Next, for each strategy, we give the number of times it found the best solution (hits), as well as the average CPU time (in seconds) on an IBM 9672 model R34. The number of iterations was fixed at 10,000.

Strategy (F) presented the shortest average computation times for three out of the four problem types. It was also the one with the least variability in the constructed solutions and, in consequence, found the best solution the fewest times. The reactive strategy (R) is the one which most often found the best solutions, however, at the cost of computation times that are longer than those of some of the other strategies.

**Table 10.2** Computational results for different strategies for the variation of parameter  $\alpha$ .

Problem	Instances	R		E		H		F	
		Hits	Time	Hits	Time	Hits	Time	Hits	Time
P-1	36	34	579.0	35	358.2	32	612.6	24	642.8
P-2	7	7	1346.8	6	1352.0	6	668.2	5	500.7
P-3	44	22	2463.7	23	2492.6	16	1740.9	11	1625.2
P-4	37	28	6363.1	21	7292.9	24	6326.5	19	5972.0
Total	124	91		85		78		59	

The high number of hits observed by strategy (E) also illustrates the effectiveness of strategies based on the variation of the RCL parameter.

## 10.3 Alternative Construction Mechanisms

A possible shortcoming of the standard GRASP framework is the independence of its iterations, i.e., the fact that it does not learn from the search history or from solutions found in previous iterations. This is so because the basic algorithm discards information about any solution previously encountered that does not improve the incumbent. Information gathered from good solutions can be used to implement memory-based procedures to influence the construction phase, by modifying the selection probabilities associated with each element of the RCL or by enforcing specific choices. Another possible shortcoming of the greedy randomized construction is its complexity. At each step of the construction, each yet unselected candidate element has to be evaluated by the greedy function. In cases where the difference between the number of elements in the ground set and the number of elements that appear in a solution large, this may not be very efficient.

In this section, we consider enhancements and alternative techniques for the construction phase of GRASP. They include random plus greedy, sampled greedy, Reactive GRASP, cost perturbations, bias functions, memory and learning, and local search on partially constructed solutions.

### 10.3.1 Random Plus Greedy and Sampled Greedy Construction

In Section 10.2, we described the semi-greedy construction scheme used to build randomized greedy solutions that serve as starting points for local search. Two other randomized greedy approaches were proposed in [158], with smaller worst-case complexities than the semi-greedy algorithm.

Instead of combining greediness and randomness at each step of the construction procedure, the *random plus greedy* scheme applies randomness during the first  $p$  construction steps to produce a random partial solution. Next, the algorithm completes the solution with one or more pure greedy construction steps. The resulting solution is randomized greedy. One can control the balance between greediness and randomness in the construction by changing the value of the

parameter  $p$ . Larger values of  $p$  are associated with solutions that are more random, while smaller values result in greedier solutions.

Similar to the random plus greedy procedure, the *sampled greedy* construction also combines randomness and greediness but in a different way. This procedure is also controlled by a parameter  $p$ . At each step of the construction process, the procedure builds a restricted candidate list by sampling  $\min\{p, |C|\}$  elements of the candidate set  $C$ . Each element of the RCL is evaluated by the greedy function. The element with the smallest greedy function value is added to the partial solution. This two-step process is repeated until there are no more candidate elements. The resulting solution is also randomized greedy. The balance between greediness and randomness can be controlled by changing the value of the parameter  $p$ , i.e., the number of candidate elements that are sampled. Small sample sizes lead to more random solutions, while large sample sizes lead to greedier solutions.

### 10.3.2 Reactive GRASP

The first strategy to incorporate a learning mechanism in the memoryless construction phase of the basic GRASP was the Reactive GRASP procedure introduced in Section 10.2. In this case, the value of the RCL parameter  $\alpha$  is not fixed, but instead is randomly selected at each iteration from a discrete set of possible values. This selection is guided by the solution values found along the previous iterations. One way to accomplish this is to use the rule proposed in [143]. Let  $\Psi = \{\alpha_1, \dots, \alpha_m\}$  be a set of possible values for  $\alpha$ . The probabilities associated with the choice of each value are all initially made equal to  $p_i = 1/m$ , for  $i = 1, \dots, m$ . Furthermore, let  $z^*$  be the incumbent solution and let  $A_i$  be the average value of all solutions found using  $\alpha = \alpha_i$ , for  $i = 1, \dots, m$ . The selection probabilities are periodically reevaluated by taking  $p_i = q_i / \sum_{j=1}^m q_j$ , with  $q_i = z^*/A_i$  for  $i = 1, \dots, m$ . The value of  $q_i$  will be larger for values of  $\alpha = \alpha_i$  leading to the best solutions on average. Larger values of  $q_i$  correspond to more suitable values for the parameter  $\alpha$ . The probabilities associated with the more appropriate values will then increase when they are reevaluated.

The reactive approach leads to improvements over the basic GRASP in terms of robustness and solution quality, due to greater diversification and less reliance on parameter tuning. In addition to the applications in [141–143], this approach has been used in power system transmission network planning [41], job shop scheduling [40], channel assignment in mobile phone networks [93], rural road network development [171], capacitated location [57], strip-packing [11], and a combined production–distribution problem [43].

### 10.3.3 Cost Perturbations

The idea of introducing some noise into the original costs is similar to that in the so-called noising method of Charon and Hudry [48, 49]. It adds more flexibility

into algorithm design and may be even more effective than the greedy randomized construction of the basic GRASP procedure in circumstances where the construction algorithms are not very sensitive to randomization. This is indeed the case for the shortest path heuristic of Takahashi and Matsuyama [175], used as one of the main building blocks of the construction phase of the hybrid GRASP procedure proposed by Ribeiro et al. [165] for the Steiner problem in graphs. Another situation where cost perturbations can be very effective appears when no greedy algorithm is available for straightforward randomization. This happens to be the case of the hybrid GRASP developed by Canuto et al. [46] for the prize-collecting Steiner tree problem, which makes use of the primal–dual algorithm of Goemans and Williamson [92] to build initial solutions using perturbed costs.

In the case of the GRASP for the prize-collecting Steiner tree problem described in [46], a new solution is built at each iteration using node prizes updated by a perturbation function, according to the structure of the current solution. Two different prize perturbation schemes were used. In *perturbation by eliminations*, the primal–dual algorithm used in the construction phase is driven to build a new solution without some of the nodes that appeared in the solution constructed in the previous iteration. In *perturbation by prize changes*, some noise is introduced into the node prizes to change the objective function, similarly to what is proposed in [48, 49].

The cost perturbation methods used in the GRASP for the minimum Steiner tree problem described in [165] incorporate learning mechanisms associated with intensification and diversification strategies. Three distinct weight randomization methods were applied. At a given GRASP iteration, the modified weight of each edge is randomly selected from a uniform distribution from an interval which depends on the selected weight randomization method applied at that iteration. The different weight randomization methods use frequency information and may be used to enforce intensification and diversification strategies. The experimental results reported in [165] show that the strategy combining these three perturbation methods is more robust than any of them used in isolation, leading to the best overall results on a quite broad mix of test instances with different characteristics. The GRASP heuristic using this cost perturbation strategy is among the most effective heuristics currently available for the Steiner problem in graphs.

### 10.3.4 Bias Functions

In the construction procedure of the basic GRASP, the next element to be introduced in the solution is chosen at random from the candidates in the RCL. The elements of the RCL are assigned equal probabilities of being chosen. However, any probability distribution can be used to bias the selection toward some particular candidates. Another construction mechanism was proposed by Bresina [44], where a family of such probability distributions is introduced. They are based on the rank  $r(\sigma)$  assigned to each candidate element  $\sigma$ , according to its greedy function value. Several bias functions were proposed, such as

- random bias:  $\text{bias}(r) = 1$ ;
- linear bias:  $\text{bias}(r) = 1/r$ ;
- log bias:  $\text{bias}(r) = \log^{-1}(r+1)$ ;
- exponential bias:  $\text{bias}(r) = e^{-r}$ ; and
- polynomial bias of order  $n$ :  $\text{bias}(r) = r^{-n}$ .

Let  $r(\sigma)$  denote the rank of element  $\sigma$  and let  $\text{bias}(r(\sigma))$  be one of the bias functions defined above. Once these values have been evaluated for all elements in the candidate set  $C$ , the probability  $\pi(\sigma)$  of selecting element  $\sigma$  is

$$\pi(\sigma) = \frac{\text{bias}(r(\sigma))}{\sum_{\sigma' \in C} \text{bias}(r(\sigma'))}. \quad (10.1)$$

The evaluation of these bias functions may be restricted to the elements of the RCL. Bresina's selection procedure restricted to elements of the RCL was used in [40]. The standard GRASP uses a random bias function.

### 10.3.5 Intelligent Construction: Memory and Learning

Fleurent and Glover [82] observed that the basic GRASP does not use long-term memory (information gathered in previous iterations) and proposed a long-term memory scheme to address this issue in multi-start heuristics. Long-term memory is one of the fundamentals on which tabu search relies.

Their scheme maintains a pool of elite solutions to be used in the construction phase. To become an elite solution, a solution must be either better than the best member of the pool or better than its worst member and sufficiently different from the other solutions in the pool. For example, one can count identical solution vector components and set a threshold for rejection.

A *strongly determined variable* is one that cannot be changed without eroding the objective or changing significantly other variables. A *consistent variable* is one that receives a particular value in a large portion of the elite solution set. Let  $I(e)$  be a measure of the strong determination and consistency features of a solution element  $e \in E$ . Then,  $I(e)$  becomes larger as  $e$  appears more often in the pool of elite solutions. The intensity function  $I(e)$  is used in the construction phase as follows. Recall that  $c(e)$  is the greedy function, i.e., the incremental cost associated with the incorporation of element  $e \in E$  into the solution under construction. Let  $K(e) = F(c(e), I(e))$  be a function of the greedy and the intensification functions. For example,  $K(e) = \lambda c(e) + I(e)$ . The intensification scheme biases selection from the RCL to those elements  $e \in E$  with a high value of  $K(e)$  by setting its selection probability to be  $p(e) = K(e) / \sum_{s \in \text{RCL}} K(s)$ .

The function  $K(e)$  can vary with time by changing the value of  $\lambda$ . For example,  $\lambda$  may be set to a large value that is decreased when diversification is called for. Procedures for changing the value of  $\lambda$  are given by Fleurent and Glover [82] and Binato et al. [40].

### 10.3.6 POP in Construction

The Proximate Optimality Principle (POP) is based on the idea that “good solutions at one level are likely to be found ‘close to’ good solutions at an adjacent level” [90]. Fleurent and Glover [82] provided a GRASP interpretation of this principle. They suggested that imperfections introduced during steps of the GRASP construction can be “ironed out” by applying local search during (and not only at the end of) the GRASP construction phase.

Because of efficiency considerations, a practical implementation of POP to GRASP consists in applying local search a few times during the construction phase, but not at every construction iteration. Local search was applied by Binato et al. [40] after 40 and 80% of the construction moves have been taken, as well as at the end of the construction phase.

## 10.4 Path-Relinking

Path-relinking is another enhancement to the basic GRASP procedure, leading to significant improvements in both solution quality and running times. This technique was originally proposed by Glover [88] as an intensification strategy to explore trajectories connecting elite solutions obtained by tabu search or scatter search [89–91].

We consider the undirected graph associated with the solution space  $G = (S, M)$ , where the nodes in  $S$  correspond to feasible solutions and the edges in  $M$  correspond to moves in the neighborhood structure, i.e.,  $(i, j) \in M$  if and only if  $i \in S$ ,  $j \in S$ ,  $j \in N(i)$ , and  $i \in N(j)$ , where  $N(s)$  denotes the neighborhood of a node  $s \in S$ . Path-relinking is usually carried out between two solutions: one is called the *initial solution*, while the other is the *guiding solution*. One or more paths in the solution space graph connecting these solutions are explored in the search for better solutions. Local search is applied to the best solution in each of these paths, since there is no guarantee that the latter is locally optimal.

Let  $s \in S$  be a node on the path between an initial solution and a guiding solution  $g \in S$ . Not all solutions in the neighborhood  $N(s)$  are allowed to be the next on the path from  $s$  to  $g$ . We restrict the choice only to those solutions that are more similar to  $g$  than  $s$ . This is accomplished by selecting moves from  $s$  that introduce attributes contained in the guiding solution  $g$ . Therefore, path-relinking may be viewed as a strategy that seeks to incorporate attributes of high-quality solutions (i.e., the guiding elite solutions), by favoring these attributes in the selected moves.

The use of path-relinking within a GRASP procedure, as an intensification strategy applied to each locally optimal solution, was first proposed by Laguna and Martí [106]. It was followed by several extensions, improvements, and successful applications [6, 7, 18, 46, 75, 130, 146, 156, 158, 159, 163, 165, 171]. A survey of GRASP with path-relinking can be found in [157].

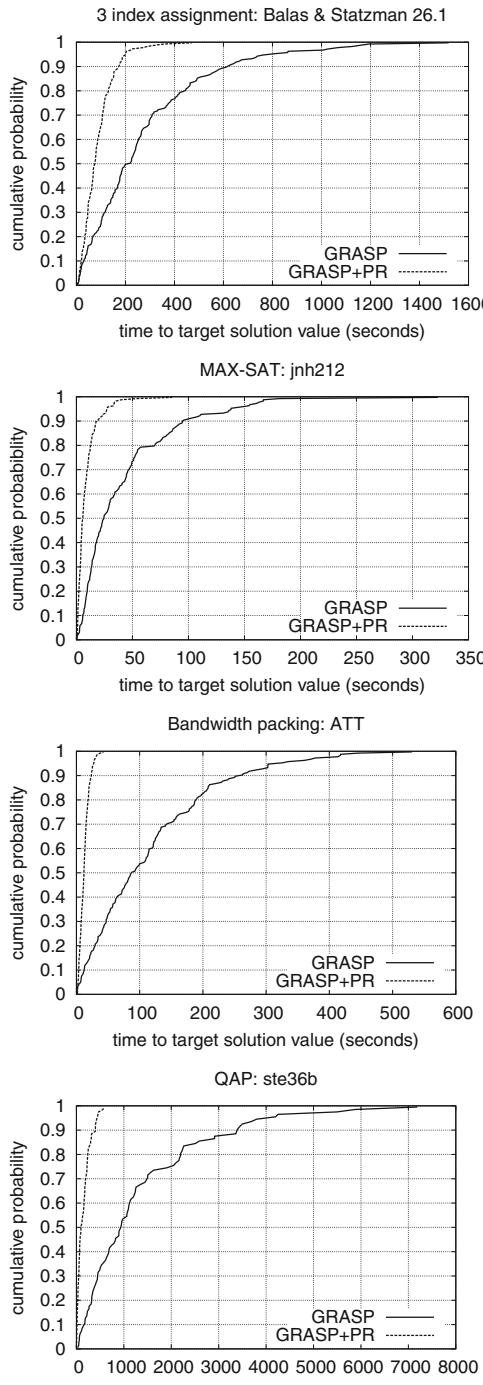
Enhancing GRASP with path-relinking almost always improves the performance of the heuristic. As an illustration, Figure 10.6 shows time-to-target plots for GRASP and GRASP with path-relinking implementations for four different applications. These time-to-target plots show the empirical cumulative probability distributions of the *time-to-target* random variable when using pure GRASP and GRASP with path-relinking, i.e., the time needed to find a solution at least as good as a prespecified target value. For all problems, the plots show that GRASP with path-relinking is able to find target solutions faster than GRASP.

GRASP with path-relinking makes use of an *elite set* to collect a diverse pool of high-quality solutions found during the search. This pool is limited in size, i.e., it can have at most `Max_Elite` solutions. Several schemes have been proposed for the implementation of path-relinking, which may be applied as

- an intensification strategy, between each local optimum obtained after the local search phase and one or more elite solutions;
- a post-optimization step, between every pair of elite solutions;
- an intensification strategy, periodically (after a fixed number of GRASP iterations since the last intensification phase) submitting the pool of elite solutions to an evolutionary process (see Section 10.4.7);
- a post-optimization phase, submitting the pool of elite solutions to an evolutionary process; or
- any other combination of the above schemes.

The pool of elite solutions is initially empty. Each locally optimal solution obtained by local search and each solution resulting from path-relinking is considered as a candidate to be inserted into the pool. If the pool is not yet full, the candidate is simply added to the pool. Otherwise, if the candidate is better than the incumbent, it replaces an element of the pool. In case the candidate is better than the worst element of the pool but not better than the best element, then it replaces some element of the pool if it is sufficiently different from every other solution currently in the pool. To balance the impact on pool quality and diversity, the element selected to be replaced is the one that is most similar to the entering solution among those elite solutions of quality no better than the entering solution [158].

Given a local optimum  $s_1$  produced at the end of a GRASP iteration, we need to select at random from the pool a solution  $s_2$  to be path-relinked with  $s_1$ . In principle, any pool solution could be selected. However, we may want to avoid pool solutions that are too similar to  $s_1$ , because relinking two solutions that are similar limits the scope of the path-relinking search. If the solutions are represented by  $|E|$ -dimensional incidence vectors, we should privilege pairs of solutions for which the Hamming distance (i.e., the number of components that take on different values in each solution) between them is high. A strategy introduced in [158] is to select a pool element  $s_2$  at random with probability proportional to the Hamming distance between the pool element and the local optimum  $s_1$ . Since the number of paths between two solutions grows exponentially with their Hamming distance, this strategy favors pool elements that have a large number of paths connecting them to and from  $s_1$ .



**Fig. 10.6** Time-to-target plots comparing running times of pure GRASP and GRASP with path-relinking on four instances of distinct problem types: three index assignment, maximum satisfiability, bandwidth packing, and quadratic assignment.

After determining which solution ( $s_1$  or  $s_2$ ) will be designated the initial solution  $i$  and which will be the guiding solution  $g$ , the algorithm starts by computing the set  $\Delta(i, g)$  of components in which  $i$  and  $g$  differ. This set corresponds to the moves which should be applied to  $i$  to reach  $g$ . Starting from the initial solution, the best move in  $\Delta(i, g)$  still not performed is applied to the current solution, until the guiding solution is reached. By best move, we mean the one that results in the highest quality solution in the restricted neighborhood. The best solution found along this trajectory is submitted to local search and returned as the solution produced by the path-relinking algorithm.

Several alternatives have been considered and combined in recent implementations. These include forward, backward, back and forward, mixed, truncated, greedy randomized adaptive, and evolutionary path-relinking. All these alternatives involve trade-offs between computation time and solution quality.

#### **10.4.1 Forward Path-Re-linking**

In *forward* path-relinking, the GRASP local optimum is designated as the initial solution and the pool solution is made the guiding solution. This is the original scheme proposed by Laguna and Martí [106].

#### **10.4.2 Backward Path-Re-linking**

In *backward* path-relinking, the pool solution is designated as the initial solution and the GRASP local optimum is made the guiding one. This scheme was originally proposed in Aiex et al. [7] and Resende and Ribeiro [156]. The main advantage of this approach over forward path-relinking comes from the fact that, in general, there are more high-quality solutions near pool elements than near GRASP local optima. Backward path-relinking explores more thoroughly the neighborhood around the pool solution, whereas forward path-relinking explores more the neighborhood around the GRASP local optimum. Experiments in [7, 156] have shown that backward path-relinking usually outperforms forward path-relinking.

#### **10.4.3 Back and Forward Path-Re-linking**

*Back and forward* path-relinking combines forward and backward path-relinking. As shown in [7, 156], it finds solutions at least as good as forward path-relinking or backward path-relinking, but at the expense of taking about twice as long to run. The reason that back and forward path-relinking often finds solutions of better quality than simple backward or forward path-relinking stems from the fact that it thoroughly explores the neighborhoods of both solutions  $s_1$  and  $s_2$ .

#### 10.4.4 Mixed Path-Relinking

*Mixed* path-relinking shares the benefits of back and forward path-relinking, i.e., it thoroughly explores both neighborhoods, but does so in about the same time as forward or backward path-relinking alone. This is achieved by interchanging the roles of the initial and guiding solutions at each step of the path-relinking procedure. Therefore, two paths are generated, one starting at  $s_1$  and the other at  $s_2$ . The paths evolve and eventually meet at some solution about half way between  $s_1$  and  $s_2$ . The joined path relinks these two solutions. Mixed path-relinking was suggested by Glover [88] and was first implemented and tested by Ribeiro and Rossetti [163], where it was shown to outperform forward, backward, and back and forward path-relinking. Figure 10.7 shows a comparison of pure GRASP and four variants of path-relinking: forward, backward, back and forward, and mixed. The time-to-target plots show that GRASP with mixed path-relinking has the best running time profile among the variants compared.

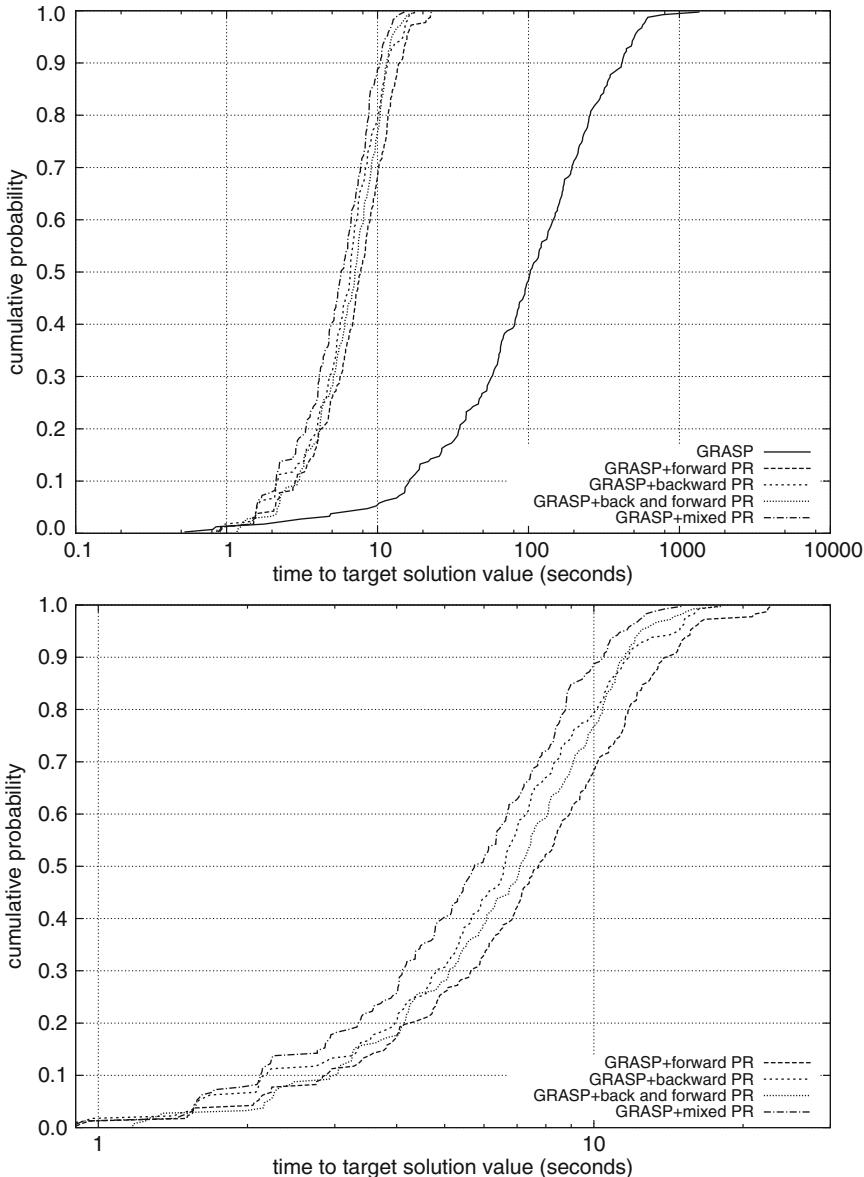
#### 10.4.5 Truncated Path-Relinking

Since good-quality solutions tend to be near other good-quality solutions, one would expect to find the best solutions with path-relinking near the initial or guiding solution. Indeed, Resende et al. [151] showed that this is the case for instances of the max–min diversity problem, as shown in Figure 10.8. In that experiment, a back and forward path-relinking scheme was tested. The figure shows the average number of best solutions found by path-relinking taken over several instances and several applications of path-relinking. The 0–10% range in this figure corresponds to subpaths near the initial solutions for the forward path-relinking phase as well as the backward phase, while the 90–100% range are subpaths near the guiding solutions. As the figure indicates, exploring the subpaths near the extremities may produce solutions about as good as those found by exploring the entire path. There is a higher concentration of better solutions close to the initial solutions explored by path-relinking.

*Truncated* path-relinking can be applied to either forward, backward, back and forward, or mixed path-relinking. Instead of exploring the entire path, truncated path-relinking only explores a fraction of the path and, consequently, takes a fraction of the time to run. Truncated path-relinking has been applied in [18, 151].

#### 10.4.6 Greedy Randomized Adaptive Path-Relinking

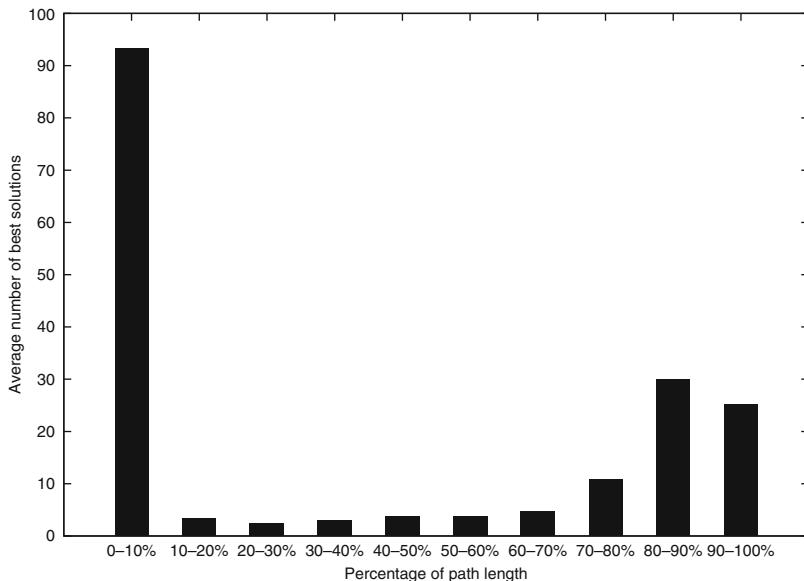
In path-relinking, the best not yet performed move in set  $\Delta(i, g)$  is applied to the current solution, until the guiding solution is reached. If ties are broken deterministically, this strategy will always produce the same path between the initial and



**Fig. 10.7** Time-to-target plots for pure GRASP and four variants of GRASP with path-relinking (forward, backward, back and forward, and mixed) on an instance of the two-path network design problem.

guiding solutions. Since the number of paths connecting  $i$  and  $g$  is exponential in  $|\Delta(i, g)|$ , exploring a single path can be somewhat limiting.

*Greedy randomized adaptive path-relinking*, introduced by Binato et al. [39], is a semi-greedy version of path-relinking. Instead of taking the best move in  $\Delta(i, g)$



**Fig. 10.8** Average number of best solutions found at different depths of the path from the initial solution to the guiding solution on instances of the max-min diversity problem.

still not performed, a restricted candidate list of good moves still not performed is set up and a randomly selected move from the latter is applied. By applying this strategy several times between the initial and guiding solutions, several paths can be explored. Greedy randomized adaptive path-relinking has been applied in [18, 63, 151].

#### 10.4.7 Evolutionary Path-Relinking

GRASP with path-relinking maintains a pool of elite solutions. Applying path-relinking between pairs of pool solutions may result in an even better pool of solutions. Aiex et al. [7] applied path-relinking between all pairs of elite solutions as an intensification scheme to improve the quality of the pool and as a post-optimization step. The application of path-relinking was repeated until no further improvement was possible.

Resende and Werneck [158, 159] described an *evolutionary* path-relinking scheme applied to pairs of elite solutions and used as a post-optimization step. The pool resulting from the GRASP with path-relinking iterations is referred to as population  $P_0$ . At step  $k$ , all pairs of elite set solutions of population  $P_k$  are relinked and the resulting solutions made candidates for inclusion in population  $P_{k+1}$  of the next generation. The same rules for acceptance into the pool during GRASP with path-relinking are used for acceptance into  $P_{k+1}$ . If the best solution in  $P_{k+1}$  is better than

the best in  $P_k$ , then  $k$  is incremented by one and the process is repeated. Resende et al. [151] describe another way to implement evolutionary path-relinking, where a single population is maintained. Each pair of elite solutions is relinked and the resulting solution is a candidate to enter the elite set. If accepted, it replaces an existing elite solution. The process is continued while there are still pairs of elite solutions that have not yet been relinked.

Andrade and Resende [17] used this evolutionary scheme as an intensification process every 100 GRASP iterations. During the intensification phase, every solution in the pool is relinked with the two best ones. Since two elite solutions might be relinked more than once in different calls to the intensification process, greedy randomized adaptive path-relinking was used.

Resende et al. [151] showed that a variant of GRASP with evolutionary path-relinking outperformed several other heuristics using GRASP with path-relinking, simulated annealing, and tabu search for the max–min diversity problem.

## 10.5 Extensions

In this section, we comment on some extensions, implementation strategies, and hybridizations of GRASP.

The use of hashing tables to avoid cycling in conjunction with tabu search was proposed by Woodruff and Zemel [178]. A similar approach was later explored by Ribeiro et al. [162] in their tabu search algorithm for query optimization in relational databases. In the context of GRASP implementations, hashing tables were first used by Martins et al. [122] in their multi-neighborhood heuristic for the Steiner problem in graphs, to avoid the application of local search to solutions already visited in previous iterations.

Filtering strategies have also been used to speed up the iterations of GRASP, see, e.g., [70, 122, 143]. In these cases, local search is not applied to all solutions obtained at the end of the construction phase, but instead only to some promising unvisited solutions, defined by a threshold with respect to the incumbent.

Almost all randomization effort in the basic GRASP algorithm involves the construction phase. Local search stops at the first local optimum. On the other hand, strategies such as VNS (variable neighborhood search), proposed by Hansen and Mladenović [96, 125], rely almost entirely on the randomization of the local search to escape from local optima. With respect to this issue, GRASP and variable neighborhood strategies may be considered as complementary and potentially capable of leading to effective hybrid methods. A first attempt in this direction was made by Martins et al. [122]. The construction phase of their hybrid heuristic for the Steiner problem in graphs follows the greedy randomized strategy of GRASP, while the local search phase makes use of two different neighborhood structures as a VND (variable neighborhood descent) procedure [96, 125]. Their heuristic was later improved by Ribeiro et al. [165], one of the key components of the new algorithm being another strategy for the exploration of different neighborhoods. Ribeiro and

Souza [164] also combined GRASP with VND in a hybrid heuristic for the degree-constrained minimum spanning tree problem. Festa et al. [81] studied different variants and combinations of GRASP and VNS for the MAX-CUT problem, finding and improving the best known solutions for some open instances from the literature.

GRASP has also been used in conjunction with genetic algorithms. Basically, the greedy randomized strategy used in the construction phase of a GRASP heuristic is applied to generate the initial population for a genetic algorithm. We may cite, e.g., the genetic algorithm of Ahuja et al. [5] for the quadratic assignment problem, which makes use of the GRASP heuristic proposed by Li et al. [108] to create the initial population of solutions. A similar approach was used by Armony et al. [27], with the initial population made up of both randomly generated solutions and those built by a GRASP algorithm.

The hybridization of GRASP with tabu search was first studied by Laguna and González-Velarde [105]. Delmaire et al. [57] considered two approaches. In the first, GRASP is applied as a powerful diversification strategy in the context of a tabu search procedure. The second approach is an implementation of the Reactive GRASP algorithm presented in Section 10.3.2, in which the local search phase is strengthened by tabu search. Results reported for the capacitated location problem show that the hybrid approaches perform better than the isolated methods previously used. Two two-stage heuristics are proposed in [1] for solving the multi-floor facility layout problem. GRASP/TS applies a GRASP to find the initial layout and tabu search to refine it.

Iterated local search (ILS) iteratively builds a sequence of solutions generated by the repeated application of local search and perturbation of the local optima found by local search [37]. Lourenço et al. [112] point out that ILS has been rediscovered many times and is also known as iterated descent [35, 36], large step Markov chains [120], iterated Lin–Kernighan [100], and chained local optimization [119]. ILS can be hybridized with GRASP by replacing the standard local search. The GRASP construction produces a solution which is passed to the ILS procedure. Ribeiro and Urrutia [166] presented a hybrid GRASP with ILS heuristic for the mirrored traveling tournament problem, in which perturbations are achieved by randomly generating solutions in the game rotation ejection chain [86, 87] neighborhood.

## 10.6 Parallel GRASP

Cung et al. [55] noted that parallel implementations of metaheuristics not only appear as quite natural alternatives to speed up the search for good approximate solutions but also facilitate solving larger problems and finding improved solutions, with respect to their sequential counterparts. This is due to the partitioning of the search space and to the increased possibilities for search intensification and diversification. As a consequence, parallelism can improve the effectiveness and robustness of metaheuristic-based algorithms. Parallel metaheuristic-based algorithms are less dependent on time-consuming parameter tuning experiments and their success is not limited to a few or small classes of problems.

Recent years have witnessed huge advances in computer technology and communication networks. The growing computational power requirements of large-scale applications and the high costs of developing and maintaining supercomputers have fueled the drive for cheaper high-performance computing environments. With the considerable increase in commodity computers and network performance, cluster computing, and, more recently, grid computing [83, 84] have emerged as real alternatives to traditional supercomputing environments for executing parallel applications that require significant amounts of computing power.

### 10.6.1 Cluster Computing

A computing cluster generally consists of a fixed number of homogeneous resources, interconnected on a single administrative network, which together execute one parallel application at a time.

Most parallel implementations of GRASP follow the *multiple-walk independent thread* strategy, based on the distribution of the iterations over the processors [12, 13, 70, 108, 121, 123, 128, 134, 135]. In general, each search thread has to perform  $\text{Max\_Iterations}/p$  iterations, where  $p$  and  $\text{Max\_Iterations}$  are the number of processors and the total number of iterations, respectively. Each processor has a copy of the sequential algorithm, a copy of the problem data, and an independent seed to generate its own pseudo-random number sequence. A single global variable is required to store the best solution found over all processors. One of the processors acts as the master, reading and distributing problem data, generating the seeds which will be used by the pseudo-random number generators at each processor, distributing the iterations, and collecting the best solution found by each processor. Since the iterations are completely independent and very little information is exchanged, linear speedups are easily obtained provided that no major load imbalance problems occur. The iterations may be evenly distributed over the processors or according to their demands, to improve load balancing.

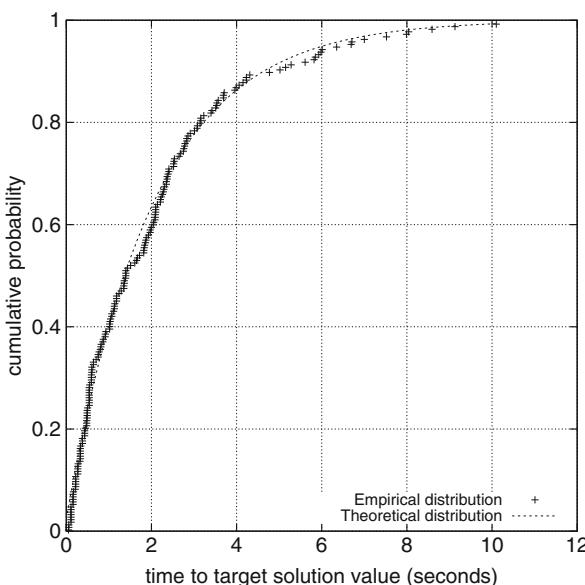
Martins et al. [123] implemented a parallel GRASP for the Steiner problem in graphs. Parallelization is achieved by the distribution of the iterations over the processors, with the value of the RCL parameter  $\alpha$  randomly chosen in the interval  $[0.0, 0.3]$  at each iteration. Almost-linear speedups were observed on benchmark problems from the OR-Library [38] for 2, 4, 8, and 16 processors, with respect to the sequential implementation. Path-relinking may be used in conjunction with parallel implementations of GRASP. Almost-linear speedups were also obtained with the multiple-walk independent-thread implementation of Aiex et al. [7] for the 3-index assignment problem, in which each processor applies path-relinking to pairs of elite solutions stored in a local pool.

Alvim and Ribeiro [12, 13] have shown that multiple-walk independent-thread approaches for the parallelization of GRASP may benefit much from load balancing techniques, whenever heterogeneous processors are used or if the parallel machine is simultaneously shared by several users. In this case, almost-linear

speedups may be obtained with a heterogeneous distribution of the iterations over the  $p$  processors in  $q$  packets, with  $q > p$ . Each processor starts performing one packet of  $\lceil \text{Max\_Iterations}/q \rceil$  iterations and informs the master when it finishes its packet of iterations. The master stops the execution of each worker processor when there are no more iterations to be performed and collects the best solution found. Faster or less loaded processors will perform more iterations than the others. In the case of the parallel GRASP heuristic implemented for the problem of traffic assignment described in [143], this dynamic load balancing strategy allowed reductions in the elapsed times of up to 15% with respect to the times observed for the static strategy, in which the iterations were uniformly distributed over the processors.

For a given problem instance and a target value `look4`, let *time-to-target* be a random variable representing the time taken by a GRASP implementation to find a solution whose cost is at least as good as `look4` for this instance. Aiex et al. [8] have shown experimentally that this random variable fits an exponential distribution or, in the case where the setup times are not negligible, a shifted (two-parameter) exponential distribution. The probability density function  $p(t)$  of the random variable time-to-target is given by  $p(t) = (1/\lambda) \cdot e^{-t/\lambda}$  in the first case or by  $p(t) = (1/\lambda) \cdot e^{-(t-\mu)/\lambda}$  in the second, with the parameters  $\lambda \in \mathbb{R}^+$  and  $\mu \in \mathbb{R}^+$  being associated with the shape and the shift of the exponential function, respectively.

Figure 10.9 illustrates this result, depicting the superimposed empirical and theoretical distributions observed for one of the cases studied along the computational experiments reported in [8], which involved 2400 runs of GRASP procedures for



**Fig. 10.9** Superimposed empirical and theoretical distributions (time-to-target solution values measured in seconds on an SGI Challenge computer with 28 processors).

each of five different problem types: maximum independent set [70, 150], quadratic assignment [108, 152], graph planarization [155, 161], maximum weighted satisfiability [154], and maximum covering [148].

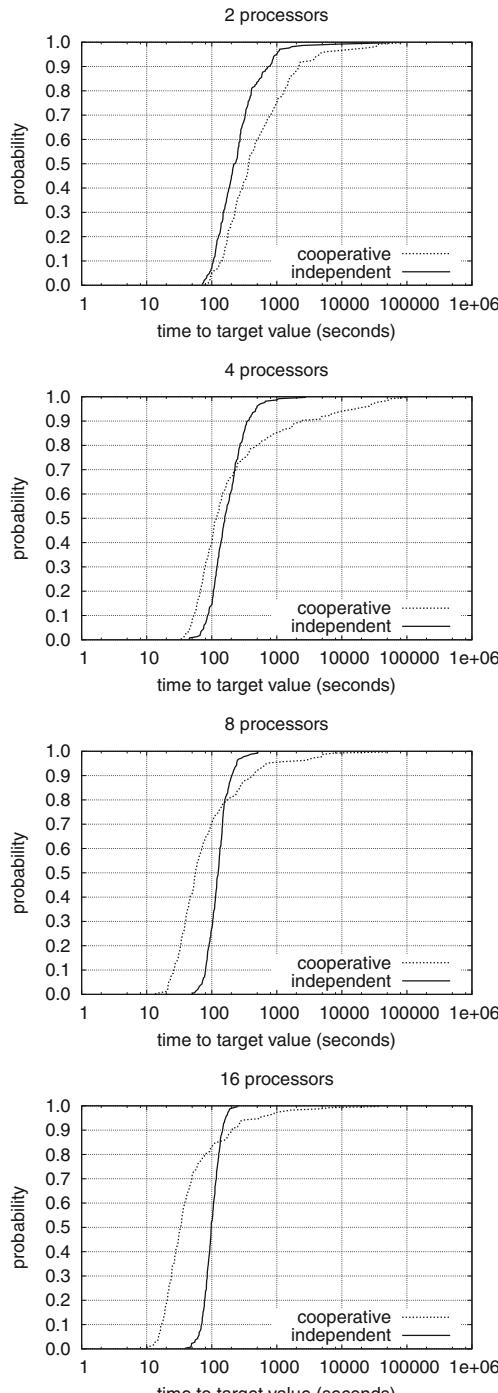
We now assume that  $p$  identical processors are available and used to search in parallel for the same target value  $\text{look4}$ . Let  $X_i$  be the time taken by processor  $i = 1, \dots, p$  to find a solution whose cost is at least as good as  $\text{look4}$  and consider the random variable  $Y = \min\{X_1, \dots, X_p\}$ . Since all processors are independent and fit the same exponential distribution with average equal to  $\lambda$ , the random variable  $Y$  fits an exponential distribution whose average is  $\lambda/p$ . Therefore, linear speedups can be achieved if multiple identical processors are used independently to search in parallel for the same target value.

However, we notice that if path-relinking is applied as an intensification step at the end of each GRASP iteration (see, e.g., [46, 157]), then the iterations are no longer independent and the memoryless characteristic of GRASP may be destroyed. Consequently, the time-to-target random variable may not fit an exponential distribution. Aiex et al. [7] have shown experimentally that, even in this case, the time-to-target random variable may be reasonably approximated by a shifted (two-parameter) exponential distribution in some circumstances.

In the case of *multiple-walk cooperative-thread* strategies, the search threads running in parallel exchange and share information collected along the trajectories they investigate. One expects not only to speed up the convergence to the best solution but, also, to find better solutions than independent-thread strategies. The most difficult aspect to be set up is the determination of the nature of the information to be shared or exchanged to improve the search, without taking too much additional memory or time to be collected. Cooperative-thread strategies may be implemented using path-relinking, by combining elite solutions stored in a central pool with the local optima found by each processor at the end of each GRASP iteration.

Ribeiro and Rossetti [163] applied this scheme to implement a parallel GRASP heuristic for the two-path network design problem. One of the processors acts as the master and handles a centralized pool of elite solutions, collecting and distributing them upon request. The other processors act as workers and exchange the elite solutions found along their search trajectories. Cooperation between the processors is implemented via path-relinking using the centralized pool of elite solutions. In this implementation, each worker may send up to three different solutions to the master at each GRASP iteration: the solution obtained by local search and the solutions obtained by forward and backward path-relinking. The performance of the parallel implementation is quite uniform over all problem instances.

Computational results illustrating the behavior of the independent and cooperative parallel implementations for an instance with 100 nodes, 4950 edges, and 1000 origin-destination pairs are presented below. The plots in Figure 10.10 display the empirical probability distribution of the time-to-target random variable for both the independent and the cooperative parallel implementations in C and MPI, for 200 runs on 2, 4, 8, and 16 processors of a 32-machine Linux cluster, with the  $\text{look4}$  target value set at 683. We notice that the independent strategy performs better when only two processors are used. This is so because the independent strategy makes use



**Fig. 10.10** Empirical distributions of the time-to-target random variable for the independent and the cooperative parallelizations on 2, 4, 8, and 16 processors (target value set at 683).

of the two processors to perform GRASP iterations, while the cooperative strategy makes use of one processor to perform iterations and the other to handle the pool. However, as the number of processors increases, the gain obtained through cooperation becomes more important than the loss of one processor to perform iterations. The cooperative implementation is already faster than the independent one for eight processors. These plots establish the usefulness and the efficiency of the cooperative implementation. Other implementations of multiple-walk cooperative-thread GRASP heuristics can be found, e.g., in Aiex et al. [6, 7].

### 10.6.2 Grid Computing

Grids aim to harness available computing power from a diverse pool of resources available over the Internet to execute a number of applications simultaneously. Grids aggregate geographically distributed collections (or sites) of resources which typically have different owners and thus are shared between multiple users. The fact that these resources are distributed, heterogeneous, and non-dedicated requires careful consideration when developing grid-enabled applications and makes writing parallel grid-aware heuristics very challenging [83].

Araújo et al. [22] described some strategies based on the master–worker paradigm for the parallelization in grid environments of the hybrid GRASP with ILS heuristic for the mirrored traveling tournament problem proposed in [166]. In the best of these strategies, PAR-MP, the master is dedicated to managing a centralized pool of elite solutions, including collecting and distributing them upon request. The workers start their searches from different initial solutions and exchange elite solutions found along their trajectories. Although it leads to improvements in the results obtained by the sequential implementation, it was not able to make full use of the characteristics of grid environments.

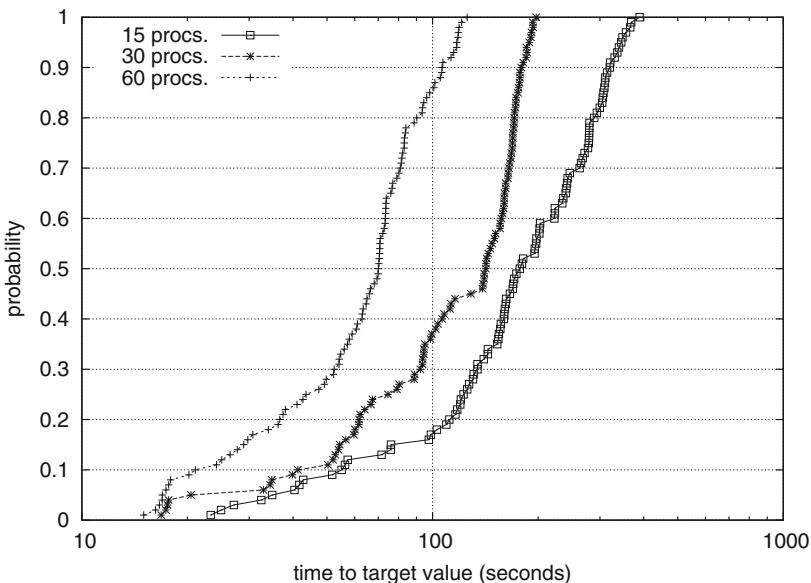
Araújo [21] proposed an autonomic hierarchical distributed strategy for the implementation of cooperative metaheuristics in grids, in which local pools of elite solutions internal to each site support intensification strategies, while a global pool is used to ensure diversification. This autonomic strategy is much more adapted to grid computations and leads to better results with respect to both the master–worker PAR-MP parallel strategy for the mirrored traveling tournament problem and the sequential hybrid heuristic combining GRASP and ILS for the diameter-constrained minimum spanning tree problem [114].

Table 10.3 displays comparative results reported in [21] for large National Football League instances of the mirrored traveling tournament problem with the number of teams ranging from 16 to 32. For each instance, we give the costs of the solutions obtained by the sequential implementation and by the hierarchical strategy running on 10 processors. The running times range from approximately 3 to 10 h, as observed for instances nf118 and nf124, respectively. We notice that the hierarchical strategy improved the solutions obtained by the sequential heuristic for eight out of the nine test instances.

**Table 10.3** Solution costs found by the sequential and grid implementations of the hybrid GRASP with ILS heuristic for the mirrored traveling tournament problem.

Instance	Sequential	Grid
nf116	25.1289	24.9806
nf118	29.9903	29.9112
nf120	35.9748	35.9748
nf122	41.8086	41.8022
nf124	46.7135	46.5491
nf126	55.4670	54.8643
nf128	61.8801	60.9788
nf130	74.0458	73.9697
nf132	92.4559	91.4620

Figure 10.11 displays time-to-target plots obtained after 100 runs of the hierarchical distributed implementation of the GRASP with ILS heuristic for the diameter-constrained minimum spanning tree on a typical instance with 100 nodes, using 15, 30, and 60 processors. These plots show that the approach scales appropriately when the number of processors increases. We display in Table 10.4 some results obtained by the sequential and the hierarchical distributed implementations of the GRASP with ILS heuristic for this problem. The distributed strategy runs on 10 processors. The sequential heuristic is allowed to run by as much as 10 times the time taken by the grid implementation. We give the number of nodes and edges for each instance, together with the costs of the best solutions found by each implementation and the



**Fig. 10.11** Time-to-target plots for the hierarchical distributed implementation of the GRASP with ILS heuristic for the diameter-constrained minimum spanning tree on an instance with 100 nodes running on a grid environment using 15, 30, and 60 processors.

**Table 10.4** Best solutions found by the sequential heuristic and by the grid implementation running on 10 processors. The sequential heuristic is allowed to run by as much as 10 times the time taken by the grid implementation.

Nodes	Edges	Grid	Sequential	Time (s)
60	600	738.000	740.000	2300.00
60	600	150.000	152.000	400.00
70	2415	6.981	6.983	230.00
70	2415	7.486	7.499	3000.00
70	2415	7.238	7.245	690.00
100	4950	7.757	7.835	1400.00
100	4950	7.930	7.961	5000.00
100	4950	8.176	8.204	3400.00

time given to the sequential heuristic. These results illustrate the robustness of the hierarchical distributed strategy (due to the effectiveness of the cooperation through the pools in two different levels), since it was able to systematically find better solutions than those obtained by the sequential strategy in computation times 10 times larger.

## 10.7 Applications

The first application of GRASP described in the literature concerned the set covering problem [68]. Since then, GRASP has been applied to a wide range of problems. The main applications areas are summarized below with links to specific references:

- routing [25, 29, 34, 45, 47, 53, 103, 110]
- logic [58, 75, 135, 149, 153, 154]
- covering and partitioning [10, 23, 26, 68, 85, 94]
- location [1, 50, 54, 57, 95, 98, 101, 132, 176, 177]
- minimum Steiner tree [46, 121–123, 165]
- optimization in graphs [2–4, 28, 70, 76, 77, 99, 104, 107, 116, 117, 122, 133, 137, 148, 150, 155, 161, 165, 173]
- assignment [5, 7, 67, 82, 108, 111, 113, 124, 127, 128, 130, 134, 136, 139, 143, 144, 152, 169]
- timetabling, scheduling, and manufacturing [6, 9, 11, 16, 18, 20, 31–33, 40, 43, 52, 56, 59, 64–66, 71, 72, 102, 105, 109, 126, 145, 166–168, 170, 179, 180]
- transportation [25, 30, 64, 67, 172]
- power systems [41, 42, 63]
- telecommunications [2, 14–16, 18, 27, 51, 101, 111, 138, 140, 143, 147, 148, 156, 174]
- graph and map drawing [54, 73, 106, 115, 116, 118, 131, 155, 161]
- biology [19, 62, 74]
- VLSI [23, 24]

The reader is referred to Festa and Resende [80] for a complete annotated bibliography of GRASP applications.

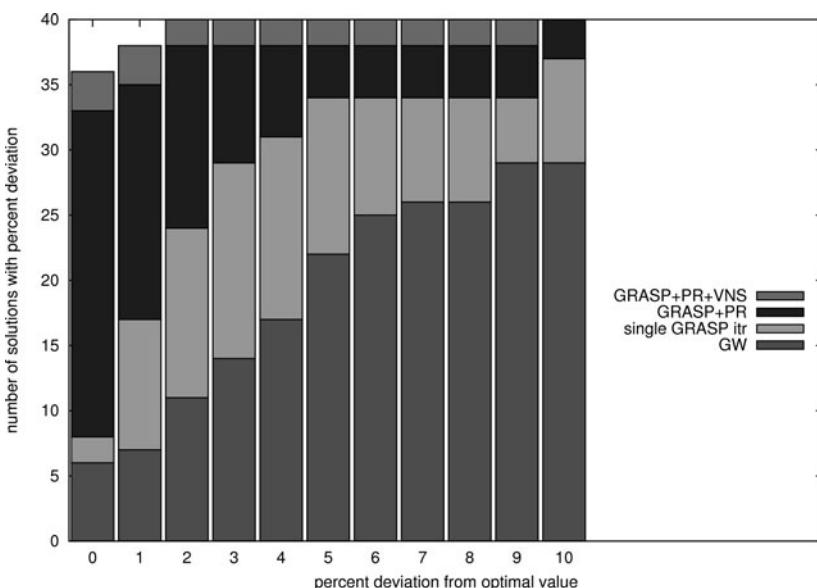
## 10.8 Concluding Remarks

The results described in this chapter reflect successful applications of GRASP to a large number of classical combinatorial optimization problems, as well as to those that arise in real-world situations in different areas of business, science, and technology.

We underscore the simplicity of implementation of GRASP, which makes use of simple building blocks (solution construction procedures and local search methods) that are often readily available. Contrary to what occurs with other metaheuristics, such as tabu search or genetic algorithms, which make use of a large number of parameters in their implementations, the basic version of GRASP requires the adjustment of a single parameter.

Recent developments, presented in this chapter, show that different extensions of the basic procedure allow further improvements in the solutions found by GRASP. Among these, we highlight reactive GRASP, which automates the adjustment of the restricted candidate list parameter; variable neighborhoods, which permit accelerated and intensified local search; and path-relinking, which beyond allowing the implementation of intensification strategies based on the memory of elite solutions opens the way for the development of very effective cooperative parallel strategies.

These and other extensions make up a set of tools that can be added to simpler heuristics to find better quality solutions. To illustrate the effect of additional extensions on solution quality, Figure 10.12 shows some results obtained for the



**Fig. 10.12** Performance of GW approximation algorithm, a single GRASP iteration (GW followed by local search), 500 iterations of GRASP with path-relinking, and 500 iterations of GRASP with path-relinking followed by VNS for series C prize-collecting Steiner tree problems.

prize-collecting Steiner tree problem, as discussed in [46]. We consider the 40 instances of series C. The figure shows results for 11 different levels of solution accuracy (varying from optimal to 10% from optimal). For each level of solution accuracy, the figure shows the number of instances for which each component found solutions within the accuracy level. The components were the primal-dual constructive algorithm (GW) of Goemans and Williamson [92], GW followed by local search (GW+LS), corresponding to the first GRASP iteration, 500 iterations of GRASP with path-relinking (GRASP+PR), and the complete algorithm, using variable neighborhood search as a post-optimization procedure (GRASP+PR+VNS). For example, we observe that the number of optimal solutions found goes from 6, using only the constructive algorithm, to a total of 36, using the complete algorithm described in [46]. The largest relative deviation with respect to the optimal value decreases from 36.4% in the first case to only 1.1% for the complete algorithm. It is easy to notice the contribution made by each additional extension.

Parallel implementations of GRASP are quite robust and lead to linear speedups both in independent and in cooperative strategies. Cooperative strategies are based on the collaboration between processors through path-relinking and a global pool of elite solutions. This allows the use of more processors to find better solutions in less computation time.

## References

1. Abdinour-Helm, S., Hadley, S.W.: Tabu search based heuristics for multi-floor facility layout. *Int. J. Prod. Res.* **38**, 365–383 (2000)
2. Abello, J., Pardalos, P.M., Resende, M.G.C.: On maximum clique problems in very large graphs. In: Abello, J., Vitter, J. (eds.) *External Memory Algorithms and Visualization*, DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol. 50 pages 199–130. American Mathematical Society Boston, MA (1999)
3. Abello, J., Resende, M.G.C., Sudarsky, S.: Massive quasi-clique detection. In: Rajsbaum, S. (ed.) *LATIN 2002: Theoretical Informatics*, Lecture Notes in Computer Science, vol. 2286 pp. 598–612. Springer, Cancun (2002)
4. Ahuja, R.K., Orlin, J.B., Sharma, D.: Multi-exchange neighborhood structures for the capacitated minimum spanning tree problem. *Math. Program.* **91**, 71–97 (2001)
5. Ahuja, R.K., Orlin, J.B., Tiwari, A.: A greedy genetic algorithm for the quadratic assignment problem. *Comput. Op. Res.* **27**, 917–934 (2000)
6. Aiex, R.M., Binato, S., Resende, M.G.C.: Parallel GRASP with path-relinking for job shop scheduling. *Parallel Comput.* **29**, 393–430 (2003)
7. Aiex, R.M., Pardalos, P.M., Resende, M.G.C., Toraldo, G.: GRASP with path-relinking for three-index assignment. *INFORMS J. Comput.* **17**, 224–247 (2005)
8. Aiex, R.M., Resende, M.G.C., Ribeiro, C.C.: Probability distribution of solution time in GRASP: An experimental investigation. *J. Heuristics* **8**, 343–373 (2002)
9. Álvarez-Valdés, R., Crespo, E., Tamarit, J.M., Villa, F.: GRASP and path relinking for project scheduling under partially renewable resources. *Eur. J. Op. Res.* **189**, 1153–1170 (2008)
10. Álvarez-Valdés, R., Parreno, F., Tamarit, J.M.: A GRASP algorithm for constrained two-dimensional non-guillotine cutting problems. *J. Op. Res. Soc.* **56**, 414–425 (2005)
11. Alvarez-Valdés, R., Parreno, F., Tamarit, J.M.: Reactive GRASP for the strip-packing problem. *Comput. Op. Res.* **35**, 1065–1083 (2008)

12. Alvim, A.C.: Parallelization strategies for the metaheuristic GRASP. Master's thesis, Department of Computer Science, Catholic University of Rio de Janeiro, Brazil [In Portuguese] (1998)
13. Alvim, A.C., Ribeiro, C.C.: Load balancing for the parallelization of the GRASP metaheuristic. In: Proceedings of the X Brazilian Symposium on Computer Architecture, pp. 279–282, Búzios [In Portuguese] (1998)
14. Amaldi, E., Capone, A., Malucelli, F.: Planning UMTS base station location: Optimization models With power control and algorithms. *IEEE Trans. Wireless Commun.* **2**, 939–952 (2003)
15. Amaldi, E., Capone, A., Malucelli, F., Signori, F.: Optimization models and algorithms for downlink UMTS radio planning. *Proc. Wireless Commun. Networking*, **2**, 827–831 (2003)
16. Andrade, D.V., Resende, M.G.C.: A GRASP for PBX telephone migration scheduling. In: Proceedings of The 8th INFORMS Telecommunications Conference, Dallas, TX (2006)
17. Andrade, D.V., Resende, M.G.C.: GRASP with evolutionary path-relinking. Technical Report TD-6XPTS7, AT&T Labs Research, Florham Park (2007)
18. Andrade, D.V., Resende, M.G.C.: GRASP with path-relinking for network migration scheduling. In: Proceedings of the International Network Optimization Conference, Spa (2007)
19. Andreatta, A.A., Ribeiro, C.C.: Heuristics for the phylogeny problem. *J. Heuristics* **8**, 429–447 (2002)
20. Andrés, C., Miralles, C., Pastor, R.: Balancing and scheduling tasks in assembly lines with sequence-dependent setup times. *Eur. J. Op. Res.* **187**, 1212–1223 (2008)
21. Araújo, A.P.F.: Autonomic Parallelization of Metaheuristics in Grid Environments. PhD thesis, Department of Computer Science, Catholic University of Rio de Janeiro [In Portuguese] (2008)
22. Araújo, A.P.F., Boeres, C., Rebello, V.E.F., Ribeiro, C.C., Urrutia, S.: Exploring grid implementations of parallel cooperative metaheuristics: A case study for the mirrored traveling tournament problem. In: Doerner, K.F., Gendreau, M., Greistorfer, P., Gutjahr, W., Hartl, R.F., Reimann, M. (eds.) *Metaheuristics: Progress in Complex Systems Optimization*, pp. 297–322. Springer, New York, NY (2007)
23. Areibi, S., Vannelli, A.: A GRASP clustering technique for circuit partitioning. In: Gu J., Pardalos, P.M. (eds.) *Satisfiability Problems*, DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol. 35 pp. 711–724. American Mathematical Society, Providence, Rhode Island, USA (1997)
24. Areibi, S.M.: GRASP: An effective constructive technique for VLSI circuit partitioning. In: Proceedings of IEEE Canadian Conference on Electrical and Computer Engineering, Edmonton, AB (1999)
25. Argüello, M.F., Bard, J.F., Yu, G.: A GRASP for aircraft routing in response to groundings and delays. *J. Comb. Op.* **1**, 211–228 (1997)
26. Argüello, M.F., Feo, T.A., Goldschmidt, O.: Randomized methods for the number partitioning problem. *Comput. Op. Res.* **23**, 103–111 (1996)
27. Armony, M., Klincewicz, J.C., Luss, H., Rosenwein, M.B.: Design of stacked self-healing rings using a genetic algorithm. *J. Heuristics* **6**, 85–105 (2000)
28. Arroyo, J.E.C., Vieira, P.S., Vianna, D.S.: A GRASP algorithm for the multi-criteria minimum spanning tree problem. *Ann. Op. Res.* **159**, 125–133 (2008)
29. Atkinson, J.B.: A greedy randomised search heuristic for time-constrained vehicle scheduling and the incorporation of a learning strategy. *J. Op. Res. Soc.* **49**, 700–708 (1998)
30. Bard, J.F.: An analysis of a rail car unloading area for a consumer products manufacturer. *J. Op. Res. Soc.* **48**, 873–883 (1997)
31. Bard, J.F., Feo, T.A.: Operations sequencing in discrete parts manufacturing. *Manage. Sci.* **35**, 249–255 (1989)
32. Bard, J.F., Feo, T.A.: An algorithm for the manufacturing equipment selection problem. *IIE Trans.* **23**, 83–92 (1991)

33. Bard, J.F., Feo, T.A., Holland, S.: A GRASP for scheduling printed wiring board assembly. *IIE Trans.* **28**, 155–165 (1996)
34. Bard, J.F., Huang, L., Jaillet, P., Dror, M.: A decomposition approach to the inventory routing problem with satellite facilities. *Transp. Sci.* **32**, 189–203 (1998)
35. Baum, E.B.: Iterated descent: A better algorithm for local search in combinatorial optimization problems. Technical report, California Institute of Technology (1986)
36. Baum, E.B.: Towards practical ‘neural’ computation for combinatorial optimization problems. In: AIP Conference Proceedings 151 on Neural Networks for Computing, pp. 53–58, Woodbury, American Institute of Physics (1987)
37. Baxter, J.: Local optima avoidance in depot location. *J. Op. Res. Soc.* **32**, 815–819 (1981)
38. Beasley, J.E.: OR-Library: Distributing test problems by electronic mail. *J. Op. Res. Soc.* **41**, 1069–1072 (1990)
39. Binato, S., Faria Jr., H., Resende, M.G.C.: Greedy randomized adaptive path relinking. In: Sousa, J.P. (ed.) *Proceedings of the IV Metaheuristics International Conference* Kluwer, Porto, pp. 393–397 (2001)
40. Binato, S., Hery, W.J., Loewenstern, D., Resende, M.G.C.: A GRASP for job shop scheduling. In: Ribeiro, C.C., Hansen, P. (eds.) *Essays and Surveys in Metaheuristics*, pp. 59–79. Kluwer, Norwell, MA (2002)
41. Binato, S., Oliveira, G.C.: A reactive GRASP for transmission network expansion planning. In: Ribeiro, C.C., Hansen, P. (eds.) *Essays and Surveys in Metaheuristics*, pp. 81–100. Kluwer, Norwell, MA (2002)
42. Binato, S., Oliveira, G.C., Araújo, J.L.: A greedy randomized adaptive search procedure for transmission expansion planning. *IEEE Trans. Power Sys.* **16**, 247–253 (2001)
43. Boudia, M., Louly, M.A.O., Prins, C.: A reactive GRASP and path relinking for a combined production-distribution problem. *Comput. Op. Res.* **34**, 3402–3419 (2007)
44. Bresina, J.L.: Heuristic-biased stochastic sampling. In: *Proceedings of the 13th National Conference on Artificial Intelligence*, pp. 271–278, Portland (1996)
45. Campbell, A.M., Thomas, B.W.: Probabilistic traveling salesman problem with deadlines. *Transp. Sci.* **42**, 1–21 (2008)
46. Canuto, S.A., Resende, M.G.C., Ribeiro, C.C.: Local search with perturbations for the prize-collecting Steiner tree problem in graphs. *Networks* **38**, 50–58 (2001)
47. Carreto, C., Baker, B.: A GRASP interactive approach to the vehicle routing problem with backhauls. In: Ribeiro, C.C., Hansen, P. (eds.) *Essays and Surveys in Metaheuristics*, pp. 185–199. Kluwer, Norwell, MA (2002)
48. Charon, I., Hudry, O.: The noising method: A new method for combinatorial optimization. *Op. Res. Lett.* **14**, 133–137 (1993)
49. Charon, I., Hudry, O.: The noising methods: A survey. In: Ribeiro, C.C., Hansen, P. (eds.) *Essays and Surveys in Metaheuristics*, pp. 245–261. Kluwer, Norwell, MA (2002)
50. Colomé, R., Serra, D.: Consumer choice in competitive location models: Formulations and heuristics. *Pap. Reg. Sci.* **80**, 439–464 (2001)
51. Commander, C., Oliveira, C.A.S., Pardalos, P.M., Resende, M.G.C.: A GRASP heuristic for the cooperative communication problem in ad hoc networks. In: *Proceedings of the VI Metaheuristics International Conference*, San Diego, CA pp. 225–330 (2005)
52. Commander, C.W., Butenko, S.I., Pardalos, P.M., Oliveira, C.A.S.: Reactive GRASP with path relinking for the broadcast scheduling problem. In: *Proceedings of the 40th Annual International Telemetry Conference*, pp. 792–800. San Diego, California, USA (2004)
53. Corberán, A., Martí, R., Sanchís, J.M.: A GRASP heuristic for the mixed Chinese postman problem. *Eur. J. Op. Res.* **142**, 70–80 (2002)
54. Cravo, G.L., Ribeiro, G.M., Nogueira Lorena, L.A.: A greedy randomized adaptive search procedure for the point-feature cartographic label placement. *Comput. Geosci.* **34**, 373–386 (2008)
55. Cung, V.-D., Martins, S.L., Ribeiro, C.C., Roucairol, C.: Strategies for the parallel implementation of metaheuristics. In: Ribeiro, C.C., Hansen, P. (eds.) *Essays and Surveys in Metaheuristics*, pp. 263–308. Kluwer, Norwell, MA (2002)

56. De, P., Ghosj, J.B., Wells, C.E.: Solving a generalized model for con due date assignment and sequencing. *Int. J. Prod. Econ.* **34**, 179–185 (1994)
57. Delmaire, H., Díaz, J.A., Fernández, E., Ortega, M.: Reactive GRASP and Tabu Search based heuristics for the single source capacitated plant location problem. *INFOR* **37**, 194–225 (1999)
58. Deshpande, A.S., Triantaphyllou, E.: A greedy randomized adaptive search procedure (GRASP) for inferring logical clauses from examples in polynomial time and some extensions. *Math. Comput. Model.* **27**, 75–99 (1998)
59. Drexl, A., Salewski, F.: Distribution requirements and compactness constraints in school timetabling. *Eur. J. Op. Res.* **102**, 193–214 (1997)
60. Duarte, A., Ribeiro, C.C., Urrutia, S.: A hybrid ILS heuristic to the referee assignment problem with an embedded MIP strategy. *Lect. Notes Comput. Sci.* **4771**, 82–95 (2007)
61. Duarte, A.R., Ribeiro, C.C., Urrutia, S., Haeusler, E.H.: Referee assignment in sports leagues. *Lect. Notes Comput. Sci.* **3867**, 158–173 (2007)
62. Ribeiro, C.C., Vianna, D.S.: A GRASP/VND heuristic for the phylogeny problem using a new neighborhood structure. *Int. Trans. Op. Res.*, **12**, 325–338 (2005)
63. Faria Jr, H., Binato, S., Resende, M.G.C., Falcão, D.J.: Transmission network design by a greedy randomized adaptive path relinking approach. *IEEE Trans. Power Syst.* **20**, 43–49 (2005)
64. Feo, T.A., Bard, J.F.: Flight scheduling and maintenance base planning. *Manag. Sci.* **35**, 1415–1432 (1989)
65. Feo, T.A., Bard, J.F.: The cutting path and tool selection problem in computer-aided process planning. *J. Manufact. Syst.* **8**, 17–26 (1989)
66. Feo, T.A., Bard, J.F., Holland, S.: Facility-wide planning and scheduling of printed wiring board assembly. *Op. Res.* **43**, 219–230 (1995)
67. Feo, T.A., González-Velarde, J.L.: The intermodal trailer assignment problem: Models, algorithms, and heuristics. *Trans. Sci.* **29**, 330–341 (1995)
68. Feo, T.A., Resende, M.G.C.: A probabilistic heuristic for a computationally difficult set covering problem. *Op. Res. Lett.* **8**, 67–71 (1989)
69. Feo, T.A., Resende, M.G.C.: Greedy randomized adaptive search procedures. *J. Global Optim.* **6**, 109–133 (1995)
70. Feo, T.A., Resende, M.G.C., Smith, S.H.: A greedy randomized adaptive search procedure for maximum independent set. *Op. Res.* **42**, 860–878 (1994)
71. Feo, T.A., Sarathy, K., McGahan, J.: A GRASP for single machine scheduling with sequence dependent setup costs and linear delay penalties. *Comput. Op. Res.* **23**, 881–895 (1996)
72. Feo, T.A., Venkatraman, K., Bard, J.F.: A GRASP for a difficult single machine scheduling problem. *Comput. Op. Res.* **18**, 635–643 (1991)
73. Fernández, E., Martí, R.: GRASP for seam drawing in mosaicking of aerial photographic maps. *J. Heuristics* **5**, 181–197 (1999)
74. Festa, P.: On some optimization problems in molecular biology. *Math. Biosci.* **207**, 219–234 (2007)
75. Festa, P., Pardalos, P.M., Pitsoulis, L.S., Resende, M.G.C.: GRASP with path-relinking for the weighted MAXSAT problem. *ACM J. Exp. Algorithms* **11**, 1–16 (2006)
76. Festa, P., Pardalos, P.M., Resende, M.G.C.: Algorithm **815**: FORTRAN subroutines for computing approximate solution to feedback set problems using GRASP. *ACM Trans. Math. Softw.* **27**, 456–464 (2001)
77. Festa, P., Pardalos, P.M., Resende, M.G.C., Ribeiro, C.C.: Randomized heuristics for the MAX-CUT problem. *Optim. Methods Softw.* **7**, 1033–1058 (2002)
78. Festa, P., Resende, M.G.C.: GRASP: An annotated bibliography. In: Ribeiro, C.C., Hansen, P. (eds.) *Essays and Surveys in Metaheuristics*, pp. 325–367. Kluwer Norwell, MA (2002)
79. Festa, P., Resende, M.G.C.: An annotated bibliography of GRASP, Part I: Algorithms. *Int. Trans. Op. Res.* **16**, 1–24 (2009)
80. Festa, P., Resende, M.G.C.: An annotated bibliography of GRASP, Part II: Applications. *Int. Trans. Op. Res.*, **16**, 131–172 (2009)

81. Festa, P., Resende, M.G.C., Pardalos, P., Ribeiro, C.C.: GRASP and VNS for Max-Cut. In: Extended Abstracts of the Fourth Metaheuristics International Conference, Porto pp. 371–376, Porto, July 2001
82. Fleurent, C., Glover, F.: Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory. *INFORMS J. Comput.* **11**, 198–204 (1999)
83. Foster, I., Kesselman, C. (eds.): *The GRID: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann San Francisco, CA (2004)
84. Foster, I., Kesselman, C., Tuecke, S. The anatomy of the grid: Enabling scalable virtual organizations. *Int. J. Supercomput. Appl.* **15**, 200–222 (2001)
85. Ghosh, J.B., Computational aspects of the maximum diversity problem. *Op. Res. Lett.* **19**, 175–181 (1996)
86. Glover, F.: New ejection chain and alternating path methods for traveling salesman problems. In: Balci, O., Sharda, R., Zenios, S. (eds.) *Computer Science and Operations Research: New Developments in Their Interfaces*, pp. 449–509. Elsevier, Oxford, UK (1992)
87. Glover, F.: Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Appl. Math.* **65**, 223–254 (1996)
88. Glover, F.: Tabu search and adaptive memory programming – Advances, applications and challenges. In: Barr, R.S., Helgason, R.V., Kennington, J.L. (eds.) *Interfaces in Computer Science and Operations Research*, pp. 1–75. Kluwer, Boston, MA, USA (1996)
89. Glover, F.: Multi-start and strategic oscillation methods – Principles to exploit adaptive memory. In: Laguna, M., González-Velarde, J.L. (eds.) *Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research*, pp. 1–24. Kluwer, Boston, MA, USA (2000)
90. Glover, F., Laguna, M. *Tabu Search*. Kluwer, Boston, MA, USA (1997)
91. Glover, F., Laguna, M., Martí, R.: Fundamentals of scatter search and path relinking. *Control Cybern.* **39**, 653–684 (2000)
92. Goemans, M.X., Williamson, D.P.: The primal dual method for approximation algorithms and its application to network design problems. In: Hochbaum, D. (ed.) *Approximation algorithms for NP-hard problems*, pp. 144–191. PWS Publishing, Boston, MA, USA (1996)
93. Gomes, F.C., Oliveira, C.S., Pardalos, P.M., Resende, M.G.C. Reactive GRASP with path relinking for channel assignment in mobile phone networks. In: *Proceedings of the 5th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pp. 60–67. ACM Press, Rome, Italy (2001)
94. Hammer, P.L., Rader Jr, D.J. Maximally disjoint solutions of the set covering problem. *J. Heuristics* **7**, 131–144 (2001)
95. Han, B.T., Raja, V.T. A GRASP heuristic for solving an extended capacitated concentrator location problem. *Int. J. Inf. Technol. Decis. Mak.* **2**, 597–617 (2003)
96. Hansen, P., Mladenović, N. Developments of variable neighborhood search. In: Ribeiro, C.C., Hansen, P. (eds.) *Essays and Surveys in Metaheuristics*, pp. 415–439. Kluwer Norwoll, MA (2002)
97. Hart, J.P., Shogan, A.W.: Semi-greedy heuristics: An empirical study. *Op. Res. Lett.* **6**, 107–114 (1987)
98. Holmqvist, K., Migdalas, A., Pardalos, P.M.: Greedy randomized adaptive search for a location problem with economies of scale. In: Bomze, I.M. et al. (eds.) *Developments in Global Optimization*, pp. 301–313. Kluwer, Boston, MA, USA (1997)
99. Holmqvist, K., Migdalas, A., Pardalos, P.M.: A GRASP algorithm for the single source uncapacitated minimum concave-cost network flow problem. In: Pardalos, P.M., Du, D.-Z. (eds.) *Network design: Connectivity and Facilities Location*, DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol. 40 pp. 131–142. American Mathematical Society, Providence, Rhode Island, USA (1998)
100. Johnson, D.S.: Local optimization and the traveling salesman problem. In: *Proceedings of the 17th Colloquium on Automata, LNCS*, vol. 443 pp. 446–461. Springer, London, UK (1990)
101. Klincewicz, J.G.: Avoiding local optima in the  $p$ -hub location problem using tabu search and GRASP. *Ann. Op. Res.* **40**, 283–302 (1992)

102. Klincewicz, J.G., Rajan, A.: Using GRASP to solve the component grouping problem. *Nav. Res. Logist.* **41**:893–912 (1994)
103. Kontoravdis, G., Bard, J.F.: A GRASP for the vehicle routing problem with time windows. *ORSA J. Comput.* **7**, 10–23 (1995)
104. Laguna, M., Feo, T.A., Elrod, H.C.: A greedy randomized adaptive search procedure for the two-partition problem. *Op. Res.* **42**, 677–687 (1994)
105. Laguna, M., González-Velarde, J.L.: A search heuristic for just-in-time scheduling in parallel machines. *J. Intell. Manufact.* **2**, 253–260 (1991)
106. Laguna, M., Martí, R.: GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS J. Comput.* **11**, 44–52 (1999)
107. Laguna, M., Martí, R.: A GRASP for coloring sparse graphs. *Comput. Op. Appl.* **19**, 165–178 (2001)
108. Li, Y., Pardalos, P.M., Resende, M.G.C.: A greedy randomized adaptive search procedure for the quadratic assignment problem. In: Pardalos, P.M., Wolkowicz, H. (eds.) *Quadratic Assignment and Related Problems*, DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol. 16 pp. 237–261. American Mathematical Society, Providence, Rhode Island, USA (1994)
109. Lim, A., Rodrigues, B., Wang, C.: Two-machine flow shop problems with a single server. *J. Scheduling* **9**, 515–543 (2006)
110. Lim, A., Wang, F.: A smoothed dynamic tabu search embedded GRASP for  $m$ -VRPTW. In: *Proceedings of ICTAI 2004*, pp. 704–708, Boca Raton, Florida, USA (2004)
111. Liu, X., Pardalos, P.M., Rajasekaran, S., Resende, M.G.C.: A GRASP for frequency assignment in mobile radio networks. In: Badrinath, B.R., Hsu, F., Pardalos, P.M., Rajasekaran, S. (eds.) *Mobile Networks and Computing*, DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol. 52 pp. 195–201. American Mathematical Society, Providence, Rhode Island, USA (2000)
112. Lourenço, H.R., Martin, O.C., Stützle, T.: Iterated local search. In: Glover, F., Kochenberger, G. (eds.) *Handbook of Metaheuristics*, pp. 321–353. Kluwer, Boston, MA, USA (2003)
113. Lourenço, H.R., Serra, D.: Adaptive approach heuristics for the generalized assignment problem. *Mathware Soft Comput.* **9**, 209–234 (2002)
114. Lucena, A.P., Ribeiro, C.C., Santos, A.C.: A hybrid heuristic for the diameter constrained minimum spanning tree problem. *J. Global Optim.* **46**, 363–381 (2010)
115. Martí, R.: Arc crossing minimization in graphs with GRASP. *IEE Trans.* **33**, 913–919 (2001)
116. Martí, R.: Arc crossing minimization in graphs with GRASP. *IEEE Trans.* **33**, 913–919 (2002)
117. Martí, R., Estruch, V.: Incremental bipartite drawing problem. *Comput. Op. Res.* **28**, 1287–1298 (2001)
118. Martí, R., Laguna, M.: Heuristics and meta-heuristics for 2-layer straight line crossing minimization. *Discrete App. Math.* **127**, 665–678 (2003)
119. Martin, O., Otto, S.W.: Combining simulated annealing with local search heuristics. *Ann. Op. Res.* **63**, 57–75 (1996)
120. Martin, O., Otto, S.W., Felten, E.W.: Large-step Markov chains for the traveling salesman problem. *Comp. Syst.* **5**, 299–326 (1991)
121. Martins, S.L., Pardalos, P.M., Resende, M.G.C., Ribeiro, C.C.: Greedy randomized adaptive search procedures for the steiner problem in graphs. In: Pardalos, P.M., Rajasekaran, S., Rolim, J. (eds.) *Randomization Methods in Algorithmic Design*, DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol. 43 pp. 133–145. American Mathematical Society, Providence, Rhode Island, USA (1999)
122. Martins, S.L., Resende, M.G.C., Ribeiro, C.C., Pardalos, P.: A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy. *J. Global Optim.* **17**, 267–283 (2000)
123. Martins, S.L., Ribeiro, C.C., Souza, M.C.: A parallel GRASP for the Steiner problem in graphs. In: Ferreira, A., Rolim, J. (eds.) *Proceedings of IRREGULAR'98 – 5th International*

- Symposium on Solving Irregularly Structured Problems in Parallel, Lecture Notes in Computer Science, vol. 1457 pp. 285–297. Springer, Berkeley, California, USA (1998)
- 124. Mavridou, T., Pardalos, P.M., Pitsoulis, L.S., Resende, M.G.C.: A GRASP for the biquadratic assignment problem. *Eur. J. Op. Res.* **105**, 613–621 (1998)
  - 125. Mladenović, N., Hansen, P.: Variable neighborhood search. *Comput. Op. Res.* **24**, 1097–1100 (1997)
  - 126. Monkman, S.K., Morrice, D.J., Bard, J.F.: A production scheduling heuristic for an electronics manufacturer with sequence-dependent setup costs. *Eur. J. Op. Res.* **187**, 1100–1114 (2008)
  - 127. Murphrey, R.A., Pardalos, P.M., Pitsoulis, L.S.: A greedy randomized adaptive search procedure for the multitarget multisensor tracking problem. In: Pardalos, P.M., Du, D.-Z. (eds.) *Network Design: Connectivity and Facilities Location*, DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol. 40 pp. 277–301. American Mathematical Society, Providence, Rhode Island, USA (1998)
  - 128. Murphrey, R.A., Pardalos, P.M., Pitsoulis, L.S.: A parallel GRASP for the data association multidimensional assignment problem. In: Pardalos, P.M. (ed.) *Parallel Processing of Discrete Problems*, The IMA Volumes in Mathematics and Its Applications, vol. 106 pp. 159–180. Springer, New York, NY, USA (1998)
  - 129. Nascimento, M.C.V., Resende, M.G.C., Toledo, F.M.B.: GRASP with path-relinking for the multi-plant capacitated plot sizing problem. *Euro. J. Oper. Res.* **200**(3), 747–754 (2010)
  - 130. Oliveira, C.A., Pardalos, P.M., Resende, M.G.C.: GRASP with path-relinking for the quadratic assignment problem. In: Ribeiro, C.C., Martins, S.L. (eds.) *Proceedings of III Workshop on Efficient and Experimental Algorithms*, LNCS, vol. 3059, pp. 356–368. Springer, Angra dos Reis, Brazil, (2004)
  - 131. Osman, I.H., Al-Ayoubi, B., Barake, M.: A greedy random adaptive search procedure for the weighted maximal planar graph problem. *Comput. Ind. Eng.* **45**, 635–651 (2003)
  - 132. Pacheco, J.A., Casado, S.: Solving two location models with few facilities by using a hybrid heuristic: A real health resources case. *Comput. Op. Res.* **32**, 3075–3091 (2005)
  - 133. Pardalos, P.M., Qian, T., Resende, M.G.C.: A greedy randomized adaptive search procedure for the feedback vertex set problem. *J. Comb. Optim.* **2**, 399–412 (1999)
  - 134. Pardalos, P.M., Pitsoulis, L.S., Resende, M.G.C.: A parallel GRASP implementation for the quadratic assignment problem. In: Ferreira, A., Rolim, J. (eds.) *Parallel Algorithms for Irregularly Structured Problems – Irregular’94*, pp. 115–133. Kluwer, Dordrecht, The Netherlands (1995)
  - 135. Pardalos, P.M., Pitsoulis, L.S., Resende, M.G.C.: A parallel GRASP for MAX-SAT problems. *Lect. Notes Comput. Sci.* **1184**, 575–585 (1996)
  - 136. Pardalos, P.M., Pitsoulis, L.S., Resende, M.G.C.: Algorithm 769: Fortran subroutines for approximate solution of sparse quadratic assignment problems using GRASP. *ACM Trans. Math. Softw.* **23**, 196–208 (1997)
  - 137. Patterson, R.A., Pirkul, H., Rolland, E.: A memory adaptive reasoning technique for solving the capacitated minimum spanning tree problem. *J. Heuristics* **5**, 159–180 (1999)
  - 138. Pinana, E., Plana, I., Campos, V., Martí, R.: GRASP and path relinking for the matrix bandwidth minimization. *Eur. J. Op. Res.* **153**, 200–210 (2004)
  - 139. Pitsoulis, L.S., Pardalos, P.M., Hearn, D.W.: Approximate solutions to the turbine balancing problem. *Eur. J. Op. Res.* **130**, 147–155 (2001)
  - 140. Poppe, F., Pickavet, M., Arijs, P., Demeester, P.: Design techniques for SDH mesh-restorable networks. In: *Proceedings of the European Conference on Networks and Optical Communications*, Volume 2: Core and ATM Networks, pp. 94–101. Antwerp, Belgium (1997)
  - 141. Prais, M., Ribeiro, C.C.: Parameter variation in GRASP implementations. In: *Extended Abstracts of the Third Metaheuristics International Conference*, pp. 375–380, Angra dos Reis, Brazil (1999)
  - 142. Prais, M., Ribeiro, C.C.: Parameter variation in GRASP procedures. *Investigación Operativa* **9**, 1–20 (2000)

143. Prais, M., Ribeiro, C.C.: Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS J. Comput.* **12**, 164–176 (2000)
144. Rangel, M.C., Abreu, N.M.M., Boaventura Netto, P.O.: GRASP in the QAP: An acceptance bound for initial solutions. *Pesqui. Operacional* **20**, 45–58 (2000)
145. Ravetti, M.G., Nakamura, F.G., Meneses, C.N., Resende, M.G.C., Mateus, G.R., Pardalos, P.M.: Hybrid heuristics for the permutation flow shop problem. Technical report, AT&T Labs Research Technical Report, Florham Park, New Jersey, USA (2006)
146. Reighoui, M., Prins, C., Nacima, L.: GRASP with path relinking for the capacitated arc routing problem with time windows. In: Giacobini, M. et al., (ed.) *Applications of Evolutionary Computing, Lecture Notes in Computer Science*, vol. 4448, pp. 722–731. Springer, Valencia, Spain (2007)
147. Resende, L.I.P., Resende, M.G.C.: A GRASP for frame relay permanent virtual circuit routing. In: Ribeiro, C.C., Hansen, P. (eds.) *Extended Abstracts of the III Metaheuristics International Conference*, pp. 397–401, Angra dos Reis, Brazil (1999)
148. Resende, M.G.C.: Computing approximate solutions of the maximum covering problem using GRASP. *J. Heuristics* **4**, 161–171 (1998)
149. Resende, M.G.C., Feo, T.A.: A GRASP for satisfiability. In: Johnson, D.S., Trick, M.A. (eds.) *Cliques, Coloring, and Satisfiability: The Second DIMACS Implementation Challenge*, DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol. 26 pp. 499–520. American Mathematical Society (1996)
150. Resende, M.G.C., Feo, T.A., Smith, S.H.: Algorithm 787: Fortran subroutines for approximate solution of maximum independent set problems using GRASP. *ACM Trans. Math. Softw.* **24**, 386–394 (1998)
151. Resende, M.G.C., Martí, R., Gallego, M., Duarte, A.: GRASP and path relinking for the max-min diversity problem. *Computers and Operations Research*, vol. 37, pp. 498–508 (2010)
152. Resende, M.G.C., Pardalos, P.M., Li, Y.: Algorithm 754: Fortran subroutines for approximate solution of dense quadratic assignment problems using GRASP. *ACM Trans. Math. Softw.* **22**, 104–118 (1996)
153. Resende, M.G.C., Pitsoulis, L.S., Pardalos, P.M.: Approximate solution of weighted MAX-SAT problems using GRASP. In: Gu, J., Pardalos, P.M., (eds.) *Satisfiability Problems*, DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol. 35 pp. 393–405. American Mathematical Society, Providence, Rhode Island, USA (1997)
154. Resende, M.G.C., Pitsoulis, L.S., Pardalos, P.M.: Fortran subroutines for computing approximate solutions of MAX-SAT problems using GRASP. *Discrete Appl. Math.* **100**, 95–113 (2000)
155. Resende, M.G.C., Ribeiro, C.C.: A GRASP for graph planarization. *Networks* **29**, 173–189 (1997)
156. Resende, M.G.C., Ribeiro, C.C.: A GRASP with path-relinking for private virtual circuit routing. *Networks* **41**, 104–114 (2003)
157. Resende, M.G.C., Ribeiro, C.C.: GRASP with path-relinking: Recent advances and applications. In: Ibaraki, T., Nonobe, K., Yagiura, M. (eds.) *Metaheuristics: Progress as Real Problem Solvers*, pp. 29–63. Springer, Boston, MA, USA (2005)
158. Resende, M.G.C., Werneck, R.F.: A hybrid heuristic for the  $p$ -median problem. *J. Heuristics* **10**, 59–88 (2004)
159. Resende, M.G.C., Werneck, R.F.: A hybrid multistart heuristic for the uncapacitated facility location problem. *Eur. J. Op. Res.* **174**, 54–68 (2006)
160. Ribeiro, C.C.: GRASP: Une météahéuristique gloutonne et probabiliste. In: Teghem, J., Pirlot, M. (eds.) *Optimisation approchée en recherche opérationnelle*, pp. 153–176. Hermès, Paris, France (2002)
161. Ribeiro, C.C., Resende, M.G.C.: Algorithm 797: Fortran subroutines for approximate solution of graph planarization problems using GRASP. *ACM Trans. Math. Softw.* **25**, 342–352 (1999)

162. Ribeiro, C.C., Ribeiro, C.D., Lanzelotte, R.S.: Query optimization in distributed relational databases. *J. Heuristics* **3**, 5–23 (1997)
163. Ribeiro, C.C., Rossetti, I.: Efficient parallel cooperative implementations of GRASP heuristics. *Parall. Comput.* **33**, 21–35 (2007)
164. Ribeiro, C.C., Souza, M.C.: Variable neighborhood search for the degree constrained minimum spanning tree problem. *Discrete Appl. Math.* **118**, 43–54 (2002)
165. Ribeiro, C.C., Uchoa, E., Werneck, R.F.: A hybrid GRASP with perturbations for the Steiner problem in graphs. *INFORMS J. Comput.* **14**, 228–246 (2002)
166. Ribeiro, C.C., Urrutia, S.: Heuristics for the mirrored traveling tournament problem. *Eur. J. Op. Res.* **179**, 775–787 (2007)
167. Ríos-Mercado, R.Z., Bard, J.F.: Heuristics for the flow line problem with setup costs. *Eur. J. Op. Res.* **110**, 76–98 (1998)
168. Ríos-Mercado, R.Z., Bard, J.F.: An enhanced TSP-based heuristic for makespan minimization in a flow shop with setup costs. *J. Heuristics* **5**, 57–74 (1999)
169. Robertson, A.J.: A set of greedy randomized adaptive local search procedure (GRASP) implementations for the multidimensional assignment problem. *Comput. Optim. Appl.* **19**, 145–164 (2001)
170. Rocha, P.L., Ravetti, M.G., Mateus, G.R.: The metaheuristic GRASP as an upper bound for a branch and bound algorithm in a scheduling problem with non-related parallel machines and sequence-dependent setup times. In: Proceedings of the 4th EU/ME Workshop: Design and Evaluation of Advanced Hybrid Meta-Heuristics, vol. 1, pp. 62–67, Nottingham, UK (2004)
171. Scaparra, M., Church, R.: A GRASP and path relinking heuristic for rural road network development. *J. Heuristics* **11**, 89–108 (2005)
172. Sosnowska, D.: Optimization of a simplified fleet assignment problem with metaheuristics: Simulated annealing and GRASP. In: Pardalos, P.M. (ed.) *Approximation and complexity in numerical optimization*. Kluwer, Boston, MA, USA (2000)
173. Souza, M.C., Duhamel, C., Ribeiro, C.C.: A GRASP heuristic for the capacitated minimum spanning tree problem using a memory-based local search strategy. In: Resende, M.G.C., Souza, J. (eds.) *Metaheuristics: Computer Decision-Making*, pp. 627–658. Kluwer, Boston, MA, USA (2004)
174. Srinivasan, A., Ramakrishnan, K.G., Kumaram, K., Aravamudam, M., Naqvi, S.: Optimal design of signaling networks for Internet telephony. In: IEEE INFOCOM 2000, vol. 2, pp. 707–716, Tel-Aviv, Israel (2000)
175. Takahashi, H., Matsuyama, A.: An approximate solution for the Steiner problem in graphs. *Math. J.* **24**, 573–577 (1980)
176. Urban, T.L.: Solution procedures for the dynamic facility layout problem. *Ann. Op. Res.* **76**, 323–342 (1998)
177. Urban, T.L., Chiang, W.-C., Russel, R.A.: The integrated machine allocation and layout problem. *Int. J. Prod. Res.* **38**, 2913–2930 (2000)
178. Woodruff, D.L., Zemel, E.: Hashing vectors for tabu search. *Ann. Op. Res.* **41**, 123–137 (1993)
179. Xu, J.Y., Chiu, S.Y.: Effective heuristic procedure for a field technician scheduling problem. *J. Heuristics* **7**, 495–509 (2001)
180. Yen, J., Carlsson, M., Chang, M., Garcia, J.M., Nguyen, H.: Constraint solving for inkjet print mask design. *J. Imaging Sci. Technol.* **44**, 391–397 (2000)



# Chapter 11

## Guided Local Search

Christos Voudouris, Edward P.K. Tsang and Abdullah Alshedy

**Abstract** Combinatorial explosion is a well-known phenomenon that prevents complete algorithms from solving many real-life combinatorial optimization problems. In many situations, heuristic search methods are needed. This chapter describes the principles of Guided Local Search (GLS) and Fast Local Search (FLS) and surveys their applications. GLS is a penalty-based metaheuristic algorithm that sits on top of other local search algorithms, with the aim to improve their efficiency and robustness. FLS is a way of reducing the size of the neighbourhood to improve the efficiency of local search. The chapter also provides guidance for implementing and using GLS and FLS. Four problems, representative of general application categories, are examined with detailed information provided on how to build a GLS-based method in each case.

### 11.1 Introduction

Many practical problems are NP-hard in nature, which means complete, constructive search is unlikely to satisfy our computational demand. For example, suppose we have to schedule 30 jobs on 10 machines, satisfying various production constraints. The search space has  $10^{30}$  leaf nodes. Let us assume that we use a very

---

Christos Voudouris

Group Chief Technology Office, BT plc, Orion Building, m1b1/pp12, Marthlesham Heath, Ipswich, IP5 3RE, UK

e-mail: chris.voudouris@bt.com

Edward P.K. Tsang

Department of Computer Science, University of Essex, Wivenhoe Park, Colchester, CO4 3SQ, UK  
e-mail: edward@essex.ac.uk

Abdullah Alshedy

Department of Computer Science, University of Essex, Wivenhoe Park, Colchester, CO4 3SQ, UK  
e-mail: aalshe@essex.ac.uk

clever backtracking algorithm that explores only one in every  $10^{10}$  leaf nodes. Let us generously assume that our implementation examines  $10^{10}$  nodes per second (with today’s hardware, even the most naïve backtracking algorithm should be expected to examine only  $10^5$  nodes per second). This still leaves us with approximately 300 years to solve the problem, in the worst case. Many real-life problems cannot be realistically and reliably solved by complete search. This motivates the development of local search or heuristic methods.

In this chapter, we describe GLS, a general metaheuristic algorithm and its applications. GLS sits on top of other heuristic methods with the aim to improve their efficiency or robustness. GLS has been applied to a non-trivial number of problems and found to be efficient and effective. It is relatively simple to implement and apply, with only a few parameters to tune.

The rest of this chapter will be divided into two parts: the first part describes the GLS and surveys its variants. The second part provides guidelines on how to use GLS in practical applications.

## Part I: Survey of Guided Local Search

### 11.2 Background

Local search (LS) is the basis of most heuristic search methods. It searches in the space of candidate solutions, such as the assignment of one machine to each job in the above scheduling example. The solution representation issue is significant, though it is not the subject of our discussion here. Starting from a (possibly randomly generated) candidate solution, LS moves to a “neighbour” that is “better” than the current candidate solution according to the objective function. LS stops when all neighbours are inferior to the current solution.

LS can find good solutions very quickly. However, it can be trapped in local optima—positions in the search space that are better than all their neighbours, but not necessarily representing the best possible solution (the global optimum). To improve the effectiveness of LS, various techniques have been introduced over the years. Simulated Annealing (SA), Tabu Search (TS) and Guided Local Search (GLS) all attempt to help LS escape local optimum. This chapter focuses on GLS [81].

GLS can be seen as a generalization of techniques such as GENET [15, 78, 79, 87, 88] and the min-conflicts heuristic repair method by Minton et al. [56] developed for constraint satisfaction problems. GLS also relates to ideas from the area of Search Theory on how to distribute the search effort (e.g. see [41, 68]).

The principles of GLS can be summarized as follows. As a metaheuristic method, GLS sits on top of LS algorithms. To apply GLS, one defines a set of features for the candidate solutions. When LS is trapped in local optima, certain features are selected and penalized. LS searches the solution space using the objective function augmented by the accumulated penalties. The novelty of GLS is in the way that

it selects features to penalize. GLS effectively distributes the search effort in the search space, favouring promising areas.

### 11.3 Guided Local Search

As mentioned earlier, GLS augments the given objective function with penalties. To apply GLS, one needs to define features for the problem. For example, in the travelling salesman problem [21], a feature could be *whether the candidate tour travels immediately from city A to city B*. GLS associates a cost and a penalty with each feature. The costs can often be defined by taking the terms and their coefficients from the objective function. For example, in the travelling salesman problem, the cost of the above feature can simply be the distance between cities A and B. The penalties are initialized to 0 and will only be increased when the local search reaches a local optimum. Given an objective function  $g$  that maps every candidate solution  $s$  to a numerical value, GLS defines a function  $h$  that will be used by LS (replacing  $g$ ):

$$h(s) = g(s) + \lambda \times \sum_{i \text{ is a feature}} (p_i \times I_i(s)), \quad (11.1)$$

where  $s$  is a candidate solution,  $\lambda$  is a parameter of the GLS algorithm,  $i$  ranges over the features,  $p_i$  is the penalty for feature  $i$  (all  $p_i$ 's are initialized to 0) and  $I_i$  is an indication of whether  $s$  exhibits feature  $i$ :

$$I_i(s) = 1 \text{ if } s \text{ exhibits feature } i; 0 \text{ otherwise.} \quad (11.2)$$

Sitting on top of local search algorithms, GLS helps them to escape local optima in the following way. Whenever the local search algorithm settles in a local optimum, GLS augments the cost function by adding penalties to selected features. The novelty of GLS is mainly in the way that it selects features to penalize. The intention is to penalize “unfavourable features” or features that “matter most” when a local search settles in a local optimum. A feature with high cost has more impact on the overall cost. Another factor that should be considered is the current penalty value of that feature. The utility of penalizing feature  $i$ ,  $\text{util}_i$ , under a local optimum,  $s_*$ , is defined as follows:

$$\text{util}_i(s_*) = I_i(s_*) \times \frac{c_i}{1 + p_i} \quad (11.3)$$

where  $c_i$  is the cost and  $p_i$  is the current penalty value of feature  $i$ . In other words, if a feature is not exhibited in the local optimum (indicated by  $I_i$ ), then the utility of penalizing it is 0. The higher the cost of this feature (the greater  $c_i$ ), the greater the utility of penalizing it. Besides, the larger the number of times it has been penalized (the greater  $p_i$ ), the lower the utility of penalizing it again. In a local optimum, the feature with the greatest util value will be penalized. When a feature is penalized, its penalty value is always increased by 1. The scaling of the penalty is adjusted by  $\lambda$ .

By taking the cost and current penalty into consideration in selecting the feature to penalize, GLS focuses its search effort on more promising areas of the search space: areas that contain candidate solutions that exhibit “good features”, i.e. features involving lower cost. On the other hand, penalties help to prevent the search from directing all effort to any particular region of the search space.

Naturally the choice of the features, their costs and the setting of  $\lambda$  may affect the efficiency of a search. Experience shows that the features and their costs normally come directly from the objective function. In many problems, the performance of GLS is not too sensitive to the value  $\lambda$ . It means that not too much effort is required to apply GLS to a new problem. In certain problems, one needs expertise in selecting the features and the  $\lambda$  parameter. Research aiming to reduce the sensitivity of the  $\lambda$  parameter in such cases is reported in [55].

## 11.4 Implementing Guided Local Search

A local search procedure for the particular problem is required for the algorithm to be implemented. Guided Local Search is repeatedly using this procedure to optimize the augmented objective function of the problem. The augmented objective function is modified each time a local minimum is reached by increasing the penalties of one or more of the features present in the local minimum. These features are selected by using the utility function (11.3). The pseudo-code for implementing a Guided Local Search method is presented and explained in Section 11.4.1.

### 11.4.1 Pseudo-code for Guided Local Search

The pseudo-code for the Guided Local Search procedure is the following:

```

procedure GuidedLocalSearch( $p, g, \lambda, [I_1, \dots, I_M], [c_1, \dots, c_M], M$ )
begin
     $k \leftarrow 0;$ 
     $s_0 \leftarrow ConstructionMethod(p);$ 
    /* set all penalties to 0 */
    for  $i \leftarrow 1$  until  $M$  do
         $p_i \leftarrow 0;$ 
        /* define the augmented objective function */
         $h \leftarrow g + \lambda * \sum p_i * I_i;$ 
        while StoppingCriterion do
            begin
                 $s_{k+1} \leftarrow ImprovementMethod(s_k, h);$ 
                /* compute the utility of features */
                for  $i \leftarrow 1$  until  $M$  do

```

```

utili ←  $I_i(s_{k+1}) * c_i / (1 + p_i)$ ;
/* penalize features with maximum utility */
for each  $i$  such that  $util_i$  is maximum do
     $p_i \leftarrow p_i + 1$ ;
     $k \leftarrow k + 1$ ;
end
 $s^* \leftarrow$  best solution found with respect to objective function  $g$ ;
return  $s^*$ ;
end

```

where  $p$  is the problem,  $g$  is the objective function,  $h$  is the augmented objective function,  $\lambda$  is a parameter,  $I_i$  is the indicator function of feature  $i$ ,  $c_i$  is the cost of feature  $i$ ,  $M$  is the number of features,  $p_i$  is the penalty of feature  $i$ ,  $ConstructionMethod(p)$  is the method for constructing an initial solution for problem  $p$  and  $ImprovementMethod(s_k, h)$  is the method for improving solution  $s_k$  according to the augmented objective function  $h$ .

### 11.4.2 Guidelines for Implementing the GLS Pseudo-code

To understand the pseudo-code, let us first explain the methods for constructing and improving a solution, as they are both prerequisites for building a GLS algorithm.

#### 11.4.2.1 Construction Method

As with other metaheuristics, GLS requires a construction method to generate an initial (starting) solution for the problem. In the pseudo-code, this is denoted by  $ConstructionMethod$ . This method can generate a random solution or a heuristic solution based on some known technique for constructing solutions for the particular problem. GLS is not very sensitive to the starting solution given that sufficient time is allocated to the algorithm to explore the search space of the problem.

#### 11.4.2.2 Improvement Method

A method for improving the solution is also required. In the pseudo-code, this is denoted by  $ImprovementMethod$ . This method can be a simple local search algorithm or a more sophisticated one such as Variable neighbourhood Search [30], Variable Depth Search [50], Ejection Chains [25] or combinations of local search methods with exact search algorithms [60].

It is not essential for the improvement method to generate high-quality local minima. Experiments with GLS and various local heuristics reported in [85] have shown that high-quality local minima take time to produce, resulting in less intervention

by GLS in the overall allocated search time. This may sometimes lead to inferior results compared to a simple but more computationally efficient improvement method.

Note also that the improvement method is using the augmented objective function instead of the original one.

#### **11.4.2.3 Indicator Functions and Feature Penalization**

Given that a construction and an improvement method are available for the problem, the rest of the pseudo-code is straightforward to apply. The penalties of features are initialized to zero and they are incremented for features that maximize the utility formula, after each call to the improvement method.

The indicator functions  $I_i$  for the features rarely need to be implemented. Looking at the values of the decision variables can directly identify the features present in a local minimum. When this is not possible, data structures with constant time deletion/addition operations (e.g. based on double-linked lists) can incrementally maintain the set of features present in the working solution, thus avoiding the need for an expensive computation when GLS reaches a local minimum.

The selection of features to penalize can be efficiently implemented by using the same loop for computing the utility formula for features present in the local minimum (the other features can be ignored) and also placing features with maximum utility in a vector. With a second loop, the features with maximum utility contained in this vector have their penalties incremented by one.

#### **11.4.2.4 Parameter $\lambda$**

Parameter  $\lambda$  is the only parameter of the GLS method (at least in its basic version) and in general is instance dependent. Fortunately, for several problems, it has been observed that good values for  $\lambda$  can be found by dividing the value of the objective function of a local minimum with the number of features present in it. In these problems,  $\lambda$  is dynamically computed after the first local minimum and before penalties are applied to features for the first time. The user only provides parameter  $\alpha$ , which is relatively instance independent. The recommended formula for  $\lambda$  as a function of  $\alpha$  is the following:

$$\lambda = \alpha * g(x_*) / (\text{no. of features present in } x_*), \quad (11.4)$$

where  $g$  is the objective function of the problem and  $x_*$  is a local minimum. Tuning  $\alpha$  can result in  $\lambda$  values, which work for many instances of a problem class.

Another benefit from using  $\alpha$  is that, once tuned, it can be fixed in industrialized versions of the software, resulting in ready-to-use GLS algorithms for the end user.

### 11.4.2.5 Augmented Objective Function and Move Evaluations

With regard to the objective function and the augmented objective function, the program should keep track of the actual objective value in all operations related to storing the best solution or finding a new best solution. Keeping track of the value of the augmented objective value (e.g. after adding the penalties) is not necessary since local search methods will be looking only at the differences in the augmented objective value when evaluating moves.

However, the move evaluation mechanism needs to be revised to work efficiently with the augmented objective function. Normally, the move evaluation mechanism is not directly evaluating the objective value of the new solution generated by the move. Instead, it calculates the difference  $\Delta g$  in the objective function. This difference should be combined with the difference in the amount of penalties. This can be easily done and has no significant impact on the time needed to evaluate a move. In particular, we have to take into account only features whose state changes (being deleted or added). The penalties of the features deleted are summed together. The same is done for the penalties of added features. The change in the amount of penalties due to the move is then simply given by the difference:

$$\sum_{\text{over all features } j \text{ added}} p_j - \sum_{\text{over all features } k \text{ deleted}} p_k, \quad (11.5)$$

which then has to be multiplied by  $\lambda$  and added to  $\Delta g$ .

Another minor improvement is to monitor the actual objective value not only for the solutions accepted by the local search but also for those evaluated. Since local search is using the augmented objective function, a move that generates a new best solution may be missed. From our experience, this modification does not improve significantly the performance of the algorithm although it can be useful when GLS is used to find new best known solutions to hard benchmark instances.

### 11.4.2.6 Stopping Criterion

There are many choices possible for the *StoppingCritetion*. Since GLS is not trapped in local minima, it is not clear when to stop the algorithm. Like other meta-heuristics, we usually resort to a measure related to the length of the search process. For example, we may choose to set a limit on the number of moves performed, the number of moves evaluated or the CPU time spent by the algorithm. If a lower bound is known, we can utilize it as a stopping criterion by setting the gap to be achieved between the best known solution and the lower bound. Criteria can also be combined to allow for a more flexible way to stop the GLS method.

In the next section, we look at the combination of Guided Local Search with Fast Local Search, a generalized algorithm for speeding up local search, resulting in the Guided Fast Local Search method. Guided Fast Local Search addresses the issue

of slow local search procedures and it is particularly useful when applying GLS to tackle large-scale problem instances.

## 11.5 Guided Fast Local Search

One factor which affects the efficiency of a local search algorithm is the size of the neighbourhood. If too many neighbours are considered, then the search could be very costly. This is especially true if the search takes many steps to reach a local optimum and/or each evaluation of the objective function requires a significant amount of computation. Bentley presented in [5] the *approximate 2-Opt* method to reduce the neighbourhood of 2-Opt in the TSP. We have generalized this method to a method called *Fast Local Search* (FLS). The principle is to use heuristics to identify (and ignore) neighbours that are unlikely to lead to improving moves in order to enhance the efficiency of a search.

The neighbourhood chosen for the problem is broken down into a number of small sub-neighbourhoods and an activation bit is attached to each one of them. The idea is to scan continuously the sub-neighbourhoods in a given order, searching only those with the activation bit set to 1. These sub-neighbourhoods are called active sub-neighbourhoods. Sub-neighbourhoods with the bit set to 0 are called inactive sub-neighbourhoods and they are not being searched. The neighbourhood search process does not restart whenever we find a better solution but it continues with the next sub-neighbourhood in the given order. This order may be static or dynamic (i.e. change as a result of the moves performed).

Initially, all sub-neighbourhoods are active. If a sub-neighbourhood is examined and does not contain any improving moves then it becomes inactive. Otherwise, it remains active and the improving move found is performed. Depending on the move performed, a number of other sub-neighbourhoods are also activated. In particular, we activate all the sub-neighbourhoods where we expect other improving moves to occur as a result of the move just performed. As the solution improves the process dies out with fewer and fewer sub-neighbourhoods being active until all the sub-neighbourhood bits turn to 0. The solution formed up to that point is returned as an approximate local optimum.

The overall procedure could be many times faster than conventional local search. The bit setting scheme encourages chains of moves that improve specific parts of the overall solution. As the solution becomes locally better the process is settling down, examining fewer moves and saving enormous amounts of time which would otherwise be spent on examining predominantly bad moves.

Although fast local search procedures do not generally find very good solutions, when they are combined with GLS they become very powerful optimization tools. Combining GLS with FLS is straightforward. The key idea is to associate features to sub-neighbourhoods. The associations to be made are such that for each feature we know which sub-neighbourhoods contain moves that have an immediate effect upon the state of the feature (i.e. moves that remove the feature from the solution).

By reducing the size of the neighbourhood, one may significantly reduce the amount of computation involved in each local search iteration. The idea is to enable more local search iterations in a fixed amount of time. The danger of ignoring certain neighbours is that some improvements may be missed. The hope is that the gain in “search speed” outweighs the loss in “search quality”.

## 11.6 Implementing Guided Fast Local Search

Guided Fast Local Search (GFLS) is more sophisticated than the basic GLS algorithm as it uses a number of sub-neighbourhoods, which are enabled/disabled during the search process. The main advantage of GFLS lies in its ability to focus the search after the penalties of features are increased. This can dramatically shorten the time required by an improvement method to re-optimize the solution each time the augmented objective function is modified.

In the following sections, we provide the pseudo-code for the method and also some suggestions on how to achieve an efficient implementation. We first look at the pseudo-code for Fast Local Search, which is part of the overall Guided Fast Local Search algorithm.

### **11.6.1 Pseudo-code for Fast Local Search**

The pseudo-code for Fast Local Search is the following:

```

procedure FastLocalSearch( $s, h, [bit_1, \dots, bit_L], L$ )
begin
  while  $\exists bit_i, bit_i = 1$  do
    /* i.e. while active sub-neighbourhood exists */
    for  $i \leftarrow 1$  until  $L$  do
      begin
        if  $bit_i = 1$  then
          /* search sub-neighbourhood i */
          begin
            Moves  $\leftarrow$  MovesForSubneighbourhood( $i$ );
            for each move  $m$  in Moves do
              begin
                 $s' \leftarrow m(s);$ 
                /*  $s'$  is the result of move  $m$  */
                if  $h(s') < h(s)$  then
                  /* minimization case is assumed here */
                  begin
                    /* spread activation */

```

```

ActivateSet ←
SubneighbourhoodsForMove( $m$ );
for each sub-neighbourhood  $j$  in
  ActivateSet do
     $bit_j \leftarrow 1$ ;
     $s \leftarrow s'$ ;
    goto ImprovingMoveFound
  end
end
 $bit_i \leftarrow 0$ ; /* no improving move found */
end
ImprovingMoveFound:
  continue;
end
return  $s$ ;
end

```

where  $s$  is the solution,  $h$  is the augmented objective function,  $L$  is the number of sub-neighbourhoods,  $bit_i$  is the activation bit for sub-neighbourhood  $i$ ,  $MovesForSubneighbourhood(i)$  is the method which returns the set of moves contained in sub-neighbourhood  $i$  and  $SubneighbourhoodsForMove(m)$  is the method which returns the set of sub-neighbourhoods to activate when move  $m$  is performed.

### 11.6.2 Guidelines for Implementing the FLS Pseudo-code

As explained in Section 11.5, the problem's neighbourhood is broken down into a number of sub-neighbourhoods and an activation bit is attached to each one of them. The idea is to examine sub-neighbourhoods in a given order, searching only those with the activation bit set to 1. The neighbourhood search process does not restart whenever we find a better solution but it continues with the next sub-neighbourhood in the given order. The pseudo-code given above is flexible since it does not specify which bits are initially switched on or off, something which is an input to the procedure. This allows the procedure to be focused to certain sub-neighbourhoods and not the whole neighbourhood, which may be a large one.

The procedure has two points that need to be customized. The first is the breaking down of the neighbourhood into sub-neighbourhoods (*MovesForSubneighbourhood* method in pseudo-code). The second is the mapping from moves to sub-neighbourhoods for spreading activation (*SubneighbourhoodsForMove* method in pseudo-code). Both points are strongly dependent on the move operator used.

In general, the move operator depends on the solution representation. Fortunately, several problems share the same solution representation which is typically based

on some well-known simple or composite combinatorial structure (e.g. selection, permutation, partition, composition, path, cyclic path, tree and graph). This allows us to use the same move operators for many different problems (e.g. 1-Opt, 2-Opt, Swaps and Insertions).

### 11.6.2.1 Breaking Down the Neighbourhood into Sub-neighbourhoods

The method for mapping sub-neighbourhoods to moves, which is denoted in the pseudo-code by *SubneighbourhoodToMoves*, can be defined by looking at the implementation of a typical local search procedure for the problem. This implementation, at its core, will usually contain a number of nested for-loops for generating all possible move combinations. The variable in the outer-most loop in the move generation code can be used to define the sub-neighbourhoods. The moves in each sub-neighbourhood will be those generated by the inner loops for the particular sub-neighbourhood index value at the outer-most loop.

In general, the sub-neighbourhoods can be overlapping. Fast local search is usually examining a limited number of moves compared to exhaustive neighbourhood search methods and therefore duplication of moves is not a problem. Moreover, this can be desirable sometimes to give a greater range to each sub-neighbourhood. Since not all sub-neighbourhoods are active in the same iteration, if there is no overlapping, some improving moves may be missed.

### 11.6.2.2 Spreading Activation When Moves Are Executed

The method for spreading activation, denoted by *SubneighbourhoodsForMove*, returns a set of sub-neighbourhoods to activate after a move is performed. The lower bound for this set is the sub-neighbourhood where the move originated. The upper bound (although not useful) is all the sub-neighbourhoods in the problem.

A way to define this method is to look at the particular move operator used. Moves will affect part of the solution directly or indirectly while leaving other parts unaffected. If a sub-neighbourhood contains affected parts then it needs to be activated since an opportunity could arise there for an improving move as a result of the original move performed.

The Fast Local Search loop is settling down in a local minimum when all the bits of sub-neighbourhoods turn to zero (i.e. no improving move can be found in any of the sub-neighbourhoods). Fast Local Search in that respect is similar to other local search procedures. The main differences are that the method can be focused to search particular parts of the overall neighbourhood and second, it is working in an opportunistic way looking at parts of the solution which are likely to contain improving moves rather than the whole solution. In the next section, we look at Guided Fast Local Search, which uses Fast Local Search as its improvement method.

### 11.6.3 Pseudo-code for Guided Fast Local Search

The pseudo-code for Guided Fast Local Search is given below:

```

procedure GuidedFastLocalSearch( $p, g, \lambda, [I_1, \dots, I_M], [c_1, \dots, c_M], M, L$ )
begin
     $k \leftarrow 0;$ 
     $s_0 \leftarrow ConstructionMethod(p);$ 
    /* set all penalties to 0 */
    for  $i \leftarrow 1$  until  $M$  do
         $p_i \leftarrow 0;$ 
    /* set all sub-neighbourhoods to the active state */
    for  $i \leftarrow 1$  until  $L$  do
         $bit_i \leftarrow 1;$ 
    /* define the augmented objective function */
     $h \leftarrow g + \lambda * \sum p_i * I_i;$ 
    while StoppingCriterion do
        begin
             $s_{k+1} \leftarrow FastLocalSearch(s_k, h, [bit_1, \dots, bit_L], L);$ 
            /* compute the utility of features */
            for  $i \leftarrow 1$  until  $M$  do
                 $util_i \leftarrow I_i(s_{k+1}) * c_i / (1 + p_i);$ 
            /* penalize features with maximum utility */
            for each  $i$  such that  $util_i$  is maximum do
                begin
                     $p_i \leftarrow p_i + 1;$ 
                    /* activate sub-neighbourhoods related
                     to penalized feature  $i$  */
                    ActivateSet  $\leftarrow SubneighbourhoodsForFeature(i);$ 
                    for each sub-neighbourhood  $j$  in ActivateSet do
                         $bit_j \leftarrow 1;$ 
                    end
                 $k \leftarrow k + 1;$ 
            end
             $s^* \leftarrow$  best solution found with respect to objective function  $g$ ;
            return  $s^*$ ;
        end

```

where  $FastLocalSearch(s_k, h, [bit_1, \dots, bit_L], L)$  is the fast local search method as described in Section 11.6.1,  $SubneighbourhoodsForFeature(i)$  is the method which returns the set of sub-neighbourhoods to activate when feature  $i$  is penalized, and the rest of the definitions are the same than those used in the pseudo-code for GLS described in Section 11.4.1.

### 11.6.4 Guidelines for Implementing the GFLS Pseudo-code

This pseudo-code is similar to that of Guided Local Search explained in Section 11.4. All differences relate to the manipulation of activation bits for the purpose of focusing Fast Local Search. These bits are initialized to 1. As a result, the first call to Fast Local Search is examining the whole neighbourhood for improving moves.

Subsequent calls to Fast Local Search examine only part of the neighbourhood and in particular all the sub-neighbourhoods that relate to the features penalized by GLS.

#### 11.6.4.1 Identifying Sub-neighbourhoods to Activate When Features Are Penalized

Identifying the sub-neighbourhoods that are related to a penalized feature is the task of *SubneighbourhoodsForFeature* method. The role of this method is similar to that of *SubneighbourhoodsForMove* method in Fast Local Search (see Section 11.6.2.2).

The *SubneighbourhoodsForFeature* method is usually defined based on an analysis of the move operator. After the application of penalties, we are looking for moves which remove or have the potential to remove penalized features from the solution. The sub-neighbourhoods, which contain such moves, are prime candidates for activation. Specific examples will be given later in the chapter and in the context of GLS applications.

Guided Fast Local Search is much faster than basic Guided Local Search especially in large problem instances when repeatedly and exhaustively searching the whole neighbourhood is computationally expensive.

## 11.7 GLS and Other Metaheuristics

GLS is closely related to other heuristic and metaheuristic methods. In this section, we shall discuss the relationship between GLS and Tabu Search (TS) and GLS and Genetic Algorithms (GA) and also review the different hybrids and extensions of GLS and FLS that have been developed in recent years.

### 11.7.1 GLS and Tabu Search

GLS is closely related to Tabu Search (TS). For example, penalties in GLS can be seen as soft taboos in TS that guide LS away from local minima. There are many ways to adopt TS ideas in GLS. For example, taboo lists and aspiration

ideas have been used in later versions of GLS. Penalties augment the original objective function. They help the local search to escape local optima. However, if too many penalties are built up during the search, the local search could be misguided. Resembling the tabu lists idea, a limited number of penalties are used when GLS is applied to the radio link frequency assignment problem [58]. When the list is full, old penalties are overwritten [83].

In another GLS work, aspiration (inspired by TS) is used to favour promising moves [55].

### ***11.7.2 GLS and Genetic Algorithms***

As a metaheuristic method, GLS can also sit on top of genetic algorithms (GA) [27, 33]. This has been demonstrated in Guided Genetic Algorithm (GGA) [44–47].

GGA is a hybrid of GA and GLS. It is designed to extend the domain of both GA and GLS. One major objective is to further improve the robustness of GLS. It can be seen as a GA with GLS to bring it out of local optima: if no progress has been made after a specific number of iterations (this number is a parameter of GGA), GLS modifies the fitness function (which is the objective function) by means of penalties, using the criteria defined in Equation (11.3). GA will then use the modified fitness function in future generations. The penalties are also used to bias crossover and mutation in GA—genes that contribute more to the penalties are more likely to change by these two GA operators. This allows GGA to be more focussed in its search.

On the other hand, GGA can roughly be seen as a number of GLSs running in parallel from different starting points and exchanging material in a GA manner. The difference is that only one set of penalties is used in GGA whereas parallel GLSs could have used one independent set of penalties per run. Besides, learning in GGA is more selective than parallel GLS: the updating of penalties is only based on the best chromosome found at the point of penalization.

### ***11.7.3 GLS Hybrids***

Being simple and general, GLS ideas can easily be combined with other techniques. GLS has been hybridized with several metaheuristics creating efficient frameworks which were successfully applied to several applications. Below, we review and comment on some of these hybrids of GLS.

GLS was hybridized with two major Evolutionary Computation (EC) techniques, namely Estimate Distribution Algorithm (EDA) and Evolution Strategy (ES). The hybrid of GLS with EDA was introduced by Zhang et al. [92]. They proposed a framework that incorporates GLS within EDA, in which GLS is applied to each solution in the population of EDA. The framework is successfully applied to the

Quadratic Assignment Problem. The results show the superiority of EDA/GLS over GLS alone.

The hybrid of GLS with ES was first studied by Mester and Braysy [52]. The resulting framework combines GLS and ES into an iterative two-stage procedure. GLS is used in both phases to improve the local search in the first stage and to regulate the objective function and the neighbourhood of the modified ES in the second stage. The principle of FLS is also incorporated into the idea of Penalty Variable Neighbourhood in which the neighbourhood considered by the local search is limited to a small set of the neighbours of the penalized feature.

GLS has also been hybridized with Variable Neighbourhood Search (VNS) and Large Neighbourhood Search (LNS). Kytojoki et al. [42] combine GLS with VNS in an efficient variable neighbourhood search heuristic, named Guided VNS (GVNS), which was applied to the vehicle routing problem. The addition to VNS is the use of GLS to escape local minima. The idea of threshold value borrowed from Threshold Accepting (TA) is used as a termination condition for every GLS stage. The hybrid of GLS with LNS is introduced in [89]. In the proposed framework, LNS is applied when the GLS cannot escape a local optimum after a number of penalizations, with the aim of increasing the diversity and exploring more promising parts of the search space. The effectiveness of this hybrid was demonstrated through high-quality results obtained in a planning optimization problem.

Guided Tabu Search (GTS) is a hybrid metaheuristic which combines GLS with TS proposed by Tarantilis et al. [73, 74] to solve the vehicle routing problem with heterogeneous fleet, and then extended to solve another variant of the same general problem. The basic idea is to control the exploration of TS by a guiding mechanism, based on GLS, that continuously modifies the objective function of the problem. The authors propose a new arc (as a feature) selection strategy which consider the relative arc length according to the rest of customers ( $d_{i,j}/\text{avg}_{i,j}$  rather than  $d_{i,j}$ , where  $\text{avg}_{i,j}$  is the average value of all outgoing arcs from i and j). They argue that this would lead to a more balanced arc selection, which should improve upon the most frequently employed strategy based on  $d_{i,j}$  only. Experimental results confirm the effectiveness of GTS, producing new best results for several benchmarks. De Backer et al. [3] also proposed a Guided Tabu Search hybrid in their work on the VRP.

GLS has been also successfully hybridized with Ant Colony Optimization (ACO) by Hani et al. [29]. This hybrid algorithm was applied to the facility layout problem, a variant of the Quadratic Assignment Problem (QAP). The basic idea is simple: GLS sits on top of the basic LS in the ACO.

The hybridization of GLS and Constraint Programming (CP) was introduced by Gomes et al. [28]. This method, named Guided Constraint Search, borrows ideas from GLS to improve the efficiency of pure CP methods. The basic principle is to use a fitness function to choose at each iteration only the  $N$  most promising values of each variable's domain, defining a sub-space for the CP method. The selection strategy is inspired from GLS; for each pair, a utility function, penalty parameter and cost are defined. At each iteration, those features (variable/value pairs) which were considered but did not belong to a new best solution are deemed bad features and are penalized.

### 11.7.4 Variations and Extensions

The success of GLS motivated researchers to invent new algorithms inspired from GLS, borrowing the ideas of features, penalties and utilities. Below, we briefly describe such GLS-inspired algorithms.

Partially based on GLS, which is a centralized algorithm, Basharu et al. [4] introduced an improved version for solving distributed constraint satisfaction problems. The Distributed Guided Local Search (Dis-GLS) incorporates additional heuristics to enhance its efficiency in distributed scenarios. The algorithm has been successfully applied to the distributed version of the Graph Colouring problem producing promising results compared to other distributed search algorithms.

Hifi et al. [32] introduced a variant of GLS by proposing a new penalization strategy. The principle is to distinguish two phases in the search process, namely the penalty and normal phases. The search process switches between the two phases in order to either escape local optima or diversify the search to explore another feasible space. The computational results confirm the high quality of solutions obtained by the proposed variant.

Tamura et al. [72] propose an improved version of GLS, named the Objective function Adjustment (OA) algorithm which incorporates the idea of features (from GLS) alongside the concept of energy function.

## Part II: Applications of Guided Local Search

### 11.8 Overview of Applications

GLS and its descendants have been applied to a number of non-trivial problems and have achieved state-of-the-art results.

#### 11.8.1 Radio Link Frequency Assignment Problem

In the *radio link frequency assignment problem* (RLFAP), the task is to assign available frequencies to communication channels satisfying constraints that prevent interference [7]. In some RLFAPs, the goal is to minimize the number of frequencies used. Bouju et al. [7] is an early work that applied GENET to radio length frequency assignment. For the CALMA set of benchmark problems, which have been widely used, GLS+FLS reported the best results compared to all work published previously [84]. In the NATO Symposium on RLFAP in Denmark, 1998, GGA was shown to improve the robustness of GLS [46]. In the same symposium, new and significantly improved results by GLS were reported [83]. At the time, GLS and GGA held some of the best known results in the CALMA set of benchmark problems.

### 11.8.2 Workforce Scheduling Problem

In the *workforce scheduling problem* (WSP) [2], the task is to assign technicians from various bases to serve the jobs, which may include customer requests and repairs, at various locations. Customer requirements and working hours restrict the service times at which certain jobs can be served by certain technicians. The objective is to minimize a function that takes into account the travelling cost, overtime cost and unserved jobs. In the WSP, GLS+FLS holds the best published results for the benchmark problem available to the authors [77].

### 11.8.3 Travelling Salesman Problem

The most significant results of GLS and FLS are probably in their application to the *travelling salesman problem* (TSP). The Lin-Kernighan algorithm (LK) is a specialized algorithm for the TSP that has long been perceived as the champion of this problem [50, 51]. We tested GLS+FLS+2Opt against LK [85] on a set of benchmark problems from a public TSP library [61]. Given the same amount of time, GLS+FLS+2Opt found better results than LK on average. GLS+FLS+2Opt also out-performed Simulated Annealing [36], Tabu Search [40] and Genetic Algorithm [23] implementations for the TSP. One must be cautious when interpreting such empirical results as they could be affected by many factors, including implementation details. But given that the TSP is an extensively studied problem, it takes something special for an algorithm to outperform the champions under any reasonable measure (“find the best results within a given amount of time” must be a realistic requirement). It must be emphasized that LK is specialized for the TSP but GLS and FLS are much simpler general-purpose algorithms.

GLS hybrids have also been proposed for the TSP including the combination of GLS with Memetic Algorithms [34] and also with the dynamic-programming based Dynasearch technique with encouraging preliminary results reported in [12].

Padron and Balaguer [59] have applied GLS to the related Rural Postman Problem (RPP), Vansteenwegen et al. [80] applied GLS to the related Team Orienteering Problem (TOP) and Mester et al. [53] applied the Guided Evolution Strategy hybrid metaheuristic to a genetic ordering problem (a Unidimensional Wandering Salesperson Problem, UWSP).

### 11.8.4 Function Optimization

GLS has been applied to general function optimization problems to illustrate that artificial features can be defined for problems in which the objective function suggests no obvious features. As expected, the results show that GLS spreads its search effort across solution candidates depending on their quality (as measured by the

objective function). Besides, GLS consistently found solutions in a landscape with many local sub-optima [82].

### ***11.8.5 Satisfiability and Max-SAT Problem***

Given a set of propositions in conjunctive normal form, the Satisfiability (SAT) problem is to determine whether the propositions can all be satisfied. The MAX-SAT problem is a SAT problem in which each clause is given a weight. The task is to minimize the total weight of the violated clauses. In other words, the weighted MAX-SAT problem is an optimization problem. Many researchers believe that many problems, including scheduling and planning can be formulated as SAT and MAX-SAT problems, hence these problems have received significant attention in recent years, e.g. see Gent et al. [24].

GLSSAT, an extension of GLS, was applied to both the SAT and the weighted MAX-SAT problems [54]. On a set of SAT problems from DIMACS, GLSSAT produced more frequently better or comparable solutions than those produced by WalkSAT [64], a variation of GSAT [65], which was specifically designed for the SAT problem.

On a popular set of benchmark weighted MAX-SAT problems, GLSSAT produced better or comparable solutions, more frequently than state-of-the-art algorithms, such as DLM [66], WalkSAT [64] and GRASP [63].

### ***11.8.6 Generalized Assignment Problem***

The Generalized Assignment Problem is a generic scheduling problem in which the task is to assign agents to jobs. Each job can only be handled by one agent, and each agent has a finite resource capacity that limits the number of jobs that it can be assigned to. Assigning different agents to different jobs bear different utilities. On the other hand, different agents will consume different amounts of resources when doing the same job. In a set of benchmark problems, GGA found results as good as those produced by a state-of-the-art algorithm (which was also a GA algorithm) by Chu and Beasley [11], with improved robustness [47].

GLS hybrids have been proposed for the related QAP. Zhang et al. [92] proposed the GLS/EDA hybrid metaheuristic. In addition, the hybrid of GLS with ACO (ACO\_GLS) has been applied to a variation of the QAP [29].

### ***11.8.7 Processor Configuration Problem***

In the Processor Configuration Problem, one is given a set of processors, each of which with a fixed number of connections. In connecting the processors, one objective is to minimize the maximum distance between processors. Another possible

objective is to minimize the average distance between pairs of processors [9]. In applying GGA to the Processor Configuration Problem, representation was a key issue. To avoid generating illegal configurations, only mutation is used. GGA found configurations with shorter average communication distance than those found by other previously reported algorithms [45, 46].

### ***11.8.8 Vehicle Routing Problem***

In a vehicle routing problem, one is given a set of vehicles, each with its specific capacity and availability, and a set of customers to serve, each with specific weight and/or time demand on the vehicles. The vehicles are grouped at one or more depots. Both the depots and the customers are geographically distributed. The task is to serve the customers using the vehicles, satisfying time and capacity constraints. This is a practical problem which, like many practical problems, is NP-hard.

Kilby et al. applied GLS to vehicle routing problems and achieved outstanding results [38, 39]. As a result, their work was incorporated in Dispatcher, a commercial package developed by ILOG [3].

Recently, the application of GLS and its hybrids to the VRP have been considerably extended to several variants of the problem. GLS has been applied to the vehicle routing problem with backhauls and time windows [93], and to the capacitated arc routing problem [6]. Guided Tabu Search has been applied to the VRP with time window [73, 74] and also extended to other variants of the VRP, namely the VRP with two-dimensional loading constraints [90], the VRP with simultaneous pick up and delivery [91] and the VRP with Replenishment Facility [74]. GLS with VNS [42], as well as GLS with ES [52] hybrids, has been proposed to solve large-scale VRPs.

### ***11.8.9 Constrained Logic Programming***

Lee and Tam [48] and Stuckey and Tam [69] embedded GENET in logic programming languages in order to enhance programming efficiency. In these logic programming implementations, unification is replaced by constraint satisfaction [76]. This enhances efficiency and extends applicability of logic programming. Hoos and Tsang [35] provide a good overview of local search in constraint programming.

### ***11.8.10 Other Applications of GENET and GLS***

We have experimented with GLS and FLS on a variety of other problems, including the Maximum Channel Assignment problem, a Bandwidth Packing problem variant, graph colouring and the car sequencing problem. Some of these works are available

for download over the Internet from Essex university's website [26] but are largely undocumented due to lack of time during the original development phase of the algorithm.

GLS and FLS have been successfully applied to the three-dimensional Bin Packing Problem and its variants [18, 19, 43], VLSI design problems [20] and network planning problems [22, 89]. GLS has been applied to the natural language parsing problem [14], Graph Set T-colourings Problem [10], query reformulation [57]. Variations of GLS have been applied to graph colouring [4] and the Multidimensional Knapsack problem [32]. Other applications of GENET include rail traffic control [37].

GLS and FLS have been incorporated into new software packages, namely iOpt which is a software toolkit for heuristic search methods [86] and iSchedule [17], which is an extension of iOpt for planning and scheduling applications (e.g. for solving problems such as the VRP [16]).

## 11.9 Useful Features for Common Applications

Applying Guided Local Search or Guided Fast Local Search to a problem requires identifying a *suitable* set of features to guide the search process. As explained in Section 11.3, features need to be defined in the form of indicator functions that, given a solution, return 1 if the feature is present in the solution or 0 otherwise.

Features provide the heuristic search expert with quite a powerful tool since any solution property can be potentially captured and used to guide local search. Usually, we are looking for solution properties, which have a direct impact on the objective function. These can be modelled as features with feature costs equal or analogous to their contribution to the objective function value. By applying penalties to features, GLS can guide the improvement method to avoid costly ("bad") properties, converging faster towards areas of the search space, which are of high quality.

Features are not necessarily specific to a particular problem and they can be used in several problems of similar structure. Real-world problems, which sometimes incorporate elements from several academic problems, can benefit from using more than one feature set to guide the local search in better optimizing the different terms of a complex objective function.

Below, we provide examples of features that can be deployed in the context of various problems. The reader may find them helpful and use them in his/her own optimization application.

### 11.9.1 Routing/Scheduling Problems

In routing/scheduling problems, one is seeking to minimize the time required by a vehicle to travel between customers or for a resource to be set-up from one activity to the next. Problems in this category include the Travelling Salesman Problem,

## Vehicle Routing Problem and Machine Scheduling with Sequence Dependent Set-up Times.

Travel or set-up times are modelled as edges in a path or graph structure commonly used to represent the solution of these problems. The objective function (or at least part of it) is given by the sum of lengths for the edges used in the solution.

Edges are ideal GLS features. A solution contains either an edge or not. Furthermore, each edge has a cost equal to its length. We can define a feature for each possible edge and assign a cost to it equal to the edge length. GLS quickly identifies and penalizes long and costly edges guiding local search to high-quality solutions, which use as much as possible the short edges available.

### 11.9.2 Assignment Problems

In assignment problems, a set of items has to be assigned to another set of items (e.g. airplanes to flights, locations to facilities people to work). Each assignment of item  $i$  to item  $j$  usually carries a cost and depending on the problem, a number of constraints are required to be satisfied (e.g. capacity or compatibility constraints). The assignment of item  $i$  to item  $j$  can be seen as a solution property which is either present in the solution or not. Since each assignment also carries a cost, this is another good example of a feature to be used in a GLS implementation.

In some variations of the problem such as the Quadratic Assignment Problem, the cost function is more complicated and assignments have an indirect impact on the cost. Even in these cases, we found that GLS can generate good results by assigning the same feature costs to all features (e.g. equal to 1 or some other arbitrary value). Although, GLS is not guiding the improvement method to good solutions (since this information is difficult to extract from the objective function), it can still diversify the search because of the penalty memory incorporated and it is capable of producing results comparable to popular heuristic methods.

### 11.9.3 Resource Allocation Problems

Assignment problems can be used to model resource allocation applications. A special but important case in resource allocation is when the resources available are not sufficient to service all requests. Usually, the objective function will contain a sum of costs for the unallocated requests, which is to be minimized. The cost incurred when a request is unallocated will reflect the importance of the request or the revenue lost in the particular scenario.

A possible feature to consider for these problems is whether a request is unallocated or not. If the request is unallocated then a cost is incurred in the objective function, which we can use as the feature cost to guide local search. The number of features in a problem is equal to the number of requests that may be left unallocated, one for each request. There may be hard constraints which state that certain requests

should always be allocated a resource, in which case there is no need to define a feature for them. Problems in this category include the Path Assignment Problem [1], Maximum Channel Assignment Problem [67] and Workforce Scheduling Problem [2].

### ***11.9.4 Constrained Optimization Problems***

Constraints are very important in capturing processes and systems in the real world. A number of combinatorial optimization problems deals with finding a solution, which satisfies a set of constraints or, if that is not possible, minimizes the number of constraint violations (relaxations). Constraint violations may have costs (weights) associated with them, in which case the sum of constraint violation costs is to be minimized.

Local search usually considers the number of constraint violations (or their weighted sum) as the objective function even in cases where the goal is to find a solution which satisfies all the constraints. Constraints by their nature can be easily used as features. They can be modelled by indicator functions and they also incur a cost (i.e. when violated/relaxed), which can be used as their feature cost. Problems which can benefit from this modelling include the Constraint Satisfaction and Partial Constraint Satisfaction Problem, the famous SAT and its MAX-SAT variant, Graph Colouring and various Frequency Assignment Problems [58].

The features exposed in the past sections will be used in the following case problems. In particular, we examine the application of GLS to the following problems:

- Travelling Salesman Problem (Routing/Scheduling category),
- Quadratic Assignment Problem (Assignment Problem category),
- Workforce Scheduling Problem (Resource Allocation category),
- Radio Link Frequency Assignment Problem (Constrained Optimization category).

For each case problem, we provide a short problem description along with guidelines on how to build a basic local search procedure, implement GLS and also GFLS when applicable.

## **11.10 Travelling Salesman Problem (TSP)**

### ***11.10.1 Problem Description***

There are many variations of the Travelling Salesman Problem (TSP). Here, we examine the classic symmetric TSP. The problem is defined by  $N$  cities and a symmetric distance matrix  $D = [d_{ij}]$  which gives the distance between any two cities

$i$  and  $j$ . The goal is to find a tour (i.e. closed path), which visits each city exactly once and is of minimum length. A tour can be represented as a cyclic permutation  $\pi$  on the  $N$  cities if we interpret  $\pi(i)$  to be the city visited after city  $i$ ,  $i = 1, \dots, N$ . The cost of a permutation is defined as

$$g(\pi) = \sum_{i=1}^N d_{i\pi(i)} \quad (11.6)$$

and gives the cost function of the TSP.

## 11.10.2 Local Search

### 11.10.2.1 Solution Representation

The solution representation usually adopted for the TSP is that of a vector which contains the order of the cities in the tour. For example, the  $i$ th element of the vector will contain an identifier for the  $i$ th city to be visited. Since the solution of the TSP is a closed path there is an edge implied from the last city in the vector to the first one in order to close the tour. The solution space of the problem is made of all possible permutations of the cities as represented by the vector.

### 11.10.2.2 Construction Method

A simple construction method is to generate a random tour. If the above solution representation is adopted then all that is required is a simple procedure, which generates a random permutation of the identifiers of the cities. More advanced TSP heuristics can be used if we require a higher quality starting solution to be generated [62]. This is useful in real-time/online applications where a good tour may be needed very early in the search process in case the user interrupts the algorithm. If there are no such concerns, then a random tour generator suffices since the GLS metaheuristic tends to be relatively insensitive to the starting solution and capable of finding high-quality solutions even if it runs for a relatively short time.

### 11.10.2.3 Improvement Method

Most improvement methods for the TSP are based on the  $k$ -Opt moves. Using  $k$ -Opt moves, neighbouring solutions can be obtained by deleting  $k$  edges from the current tour and reconnecting the resulting paths using  $k$  new edges. The  $k$ -Opt moves are the basis of the three most famous local search heuristics for the TSP, namely *2-Opt* [13], *3-Opt* [49] and *Lin–Kernighan (LK)* [50].

The reader can consider using the simple 2-Opt method, which in addition to its simplicity is very effective when combined with GLS. With 2-Opt, a neighbouring

solution is obtained from the current solution by deleting two edges, reversing one of the resulting paths and reconnecting the tour. In practical terms, this means reversing the order of the cities in a contiguous section of the vector or its remainder depending on which one is the shortest in length.

Computing incrementally the change in solution cost by a 2-Opt move is relatively simple. Let us assume that edges  $e_1$  and  $e_2$  are removed and edges  $e_3$  and  $e_4$  are added with lengths  $d_1, d_2, d_3, d_4$ , respectively. The change in cost is the following:

$$d_3 + d_4 - d_1 - d_2. \quad (11.7)$$

When we discuss the features used in the TSP, we will explain how this evaluation mechanism is revised to account for penalty changes in the augmented objective function.

### 11.10.3 Guided Local Search

For the TSP, a tour includes a number of edges and the solution cost (tour length) is given by the sum of the lengths of the edges in the tour (see Equation (11.6)). As mentioned in Section 11.9.1, edges are ideal features for routing problems such as the TSP. First, a tour either includes an edge or not and second, each edge incurs a cost in the objective function which is equal to the edge length, as given by the distance matrix  $D = [d_{ij}]$  of the problem. A set of features can be defined by considering all possible undirected edges  $e_{ij}$  ( $i = 1 \dots N, j = i + 1 \dots N, i \neq j$ ) that may appear in a tour with feature costs given by the edge lengths  $d_{ij}$ . With each edge  $e_{ij}$  connecting cities  $i$  and  $j$  is attached a penalty  $p_{ij}$  initially set to 0 which is increased by GLS during the search. When implementing the GLS algorithm for the TSP, the edge penalties can be arranged in a symmetric penalty matrix  $P = [p_{ij}]$ . As mentioned in Section 11.3, penalties have to be combined with the problem's objective function to form the augmented objective function which is minimized by local search. We therefore need to consider the auxiliary distance matrix:

$$D' = D + \lambda \cdot P = [d_{ij} + \lambda \cdot p_{ij}]. \quad (11.8)$$

Local search must use  $D'$  instead of  $D$  in move evaluations. GLS modifies  $P$  and (through that)  $D'$  whenever the local search reaches a local minimum.

In order to implement this, we revise the incremental move evaluation formula (11.7) to take into account the edge penalties and also parameter  $\lambda$ . If  $p_1, p_2, p_3, p_4$  are the penalties associated with edges  $e_1, e_2, e_3$ , and  $e_4$ , respectively, the revised version of Equation (11.7) is as follows:

$$(d_3 + d_4 - d_1 - d_2) + \lambda * (p_3 + p_4 - p_1 - p_2). \quad (11.9)$$

Similarly, we can implement GLS for higher order k-Opt moves.

The edges penalized in a local minimum are selected according to the utility function (11.3), which for the TSP takes the form

$$\text{util}(\text{tour}, e_{ij}) = I_{e_{ij}}(\text{tour}) \cdot \frac{d_{ij}}{1 + p_{ij}}, \quad (11.10)$$

where

$$I_{e_{ij}}(\text{tour}) = \begin{cases} 1, & e_{ij} \in \text{tour} \\ 0, & e_{ij} \notin \text{tour}. \end{cases} \quad (11.11)$$

The only parameter of GLS that requires tuning is parameter  $\lambda$ . Alternatively, we can tune the parameter  $\alpha$  parameter which is defined in Section 11.4.2 and is relatively instance independent. Experimenting with  $\alpha$  on the TSP, we found that there is an inverse relation between  $\alpha$  and local search effectiveness. Not so effective local search heuristics such as 2-Opt require higher  $\alpha$  values compared to more effective heuristics such as 3-Opt and LK. This is probably because the amount of penalty needed to escape from local minima decreases as the effectiveness of the heuristic increases explaining why lower values for  $\alpha$  (and consequently for  $\lambda$  which is a function of  $\alpha$ ) work better with 3-Opt and LK. For 2-Opt, the following range for  $\alpha$  generates high-quality solutions for instances in the TSPLIB [61]:

$$1/8 \leq \alpha \leq 1/2. \quad (11.12)$$

The reader may refer to [85] for more details on the experimentation procedure and the full set of results.

#### 11.10.4 Guided Fast Local Search

We can exploit the way local search works on the TSP to partition the neighbourhood in sub-neighbourhoods as required by Guided Fast Local Search. Each city in the problem may be seen as defining a sub-neighbourhood, which contains all 2-Opt edge exchanges removing one of the edges adjacent to the city. For a problem with  $N$  cities, the neighbourhood is partitioned into  $N$  sub-neighbourhoods, one for each city in the instance.

The sub-neighbourhoods to be activated after a move is executed are those of the cities at the ends of the edges removed or added by the move.

Finally, the sub-neighbourhoods activated after penalization are those defined by the cities at the ends of the edge(s) penalized. There is a good chance that these sub-neighbourhoods will include moves that remove one or more of the penalized edges.

## 11.11 Quadratic Assignment Problem (QAP)

### 11.11.1 Problem Description

The Quadratic Assignment Problem (QAP) is one of the most difficult problems in combinatorial optimization. The problem can model a variety of applications but it is mainly known for its use in facility location problems. In the following, we describe the QAP in its simplest form.

Given a set  $N = \{1, 2, \dots, n\}$  and  $n \times n$  matrices  $A = [a_{ij}]$  and  $B = [b_{kl}]$ , the QAP can be stated as follows:

$$\min_{p \in \Pi_N} \sum_{i=1}^n \sum_{j=1}^n a_{ij} \cdot b_{p(i)p(j)}, \quad (11.13)$$

where  $p$  is a permutation of  $N$  and  $\Pi_N$  is the set of all possible permutations. There are several other equivalent formulations of the problem. In the facility location context, each permutation represents an assignment of  $n$  facilities to  $n$  locations. More specifically, each position  $i$  in the permutation represents a location and its contents  $p(i)$  the facility assigned to that location. The matrix  $A$  is called the distance matrix and gives the distance between any two of the locations. The matrix  $B$  is called the flow matrix and gives the flow of materials between any two of the facilities. For simplicity, we only consider the Symmetric QAP case for which both the distance and flow matrices are symmetric.

### 11.11.2 Local Search

QAP solutions can be represented by permutations to satisfy the constraint that each facility is assigned to exactly one location. A move commonly used for the problem is simply to exchange the contents of two permutation positions (i.e. swap the facilities assigned to a pair of locations). A best improvement local search procedure starts with a random permutation. In each iteration, all possible moves (i.e. swaps) are evaluated and the best one is selected and performed. The algorithm reaches a local minimum when there is no move, which improves further the cost of the current permutation.

An efficient update scheme can be used in the QAP which allows evaluation of moves in constant time. The scheme works only with best improvement local search. Move values of the first neighbourhood search are stored and updated each time a new neighbourhood search is performed to take into account changes from the move last executed, see [71] for details. Move values do not need to be evaluated from scratch and thus the neighbourhood can be fully searched in roughly  $O(n^2)$

time instead of  $O(n^3)$ <sup>1</sup>. To evaluate moves in constant time, we have to examine all possible moves in each iteration and have their values updated. Because of that, the scheme cannot be easily combined with Fast Local Search, which examines only a number of moves in each iteration therefore preventing the problem to benefit substantially from GLS.

### 11.11.3 Guided Local Search

A set of features that can be used in the QAP is the set of all possible assignments of facilities to locations (i.e. location–facility pairs). This kind of feature is general and can be used in a variety of assignment problems as explained in Section 11.9.2. In the QAP, there are  $n^2$  possible location–facility combinations. Because of the structure of the objective function, it is not possible to estimate easily the impact of features and assign to them appropriate feature costs. In particular, the contribution in the objective function of a facility assignment to a location depends also on the placement of the other facilities with a non-zero flow to that facility.

Experimenting with the problem, we found that if all features are assigned the same cost (e.g. 1), the algorithm is still capable of generating high-quality solutions. This is due to the ability of GLS to diversify search using the penalty memory. Since features are considered of equal cost, the algorithm is distributing search efforts uniformly across the feature set. Comparative tests we conducted between GLS and the Tabu Search of [70] indicate that both algorithms are performing equally well when applied to the QAPLIB instances [8] with no clear winner across the instance set. GLS, although not using feature costs in this problem, is still very competitive to state-of-the-art techniques such as Tabu Search.

To determine  $\lambda$  in the QAP, one may use the formula below, which was derived experimentally:

$$\lambda = \alpha * n * (\text{mean flow}) * (\text{mean distance}), \quad (11.14)$$

where  $n$  is the size of the problem and the flow and distance means are computed over the distance and flow matrices, respectively (including any possible 0 entries which are common in QAP instances). Experimenting with QAPLIB instances, we found that optimal performance is achieved for  $\alpha = 0.75$ .

---

<sup>1</sup> To evaluate the change in the cost function (11.13) caused by a move normally requires  $O(n)$  time. Since there are  $O(n^2)$  moves to be evaluated, the search of the neighbourhood without the update scheme requires  $O(n^3)$  time.

## 11.12 Workforce Scheduling Problem

### 11.12.1 Problem Description

We now look at how GLS can be applied to a real-word resource allocation problem with unallocated requests called the Workforce Scheduling problem (WSP), see [77] for more details. The problem is to schedule a number of engineers to a set of jobs, minimizing the total cost according to a function, which is to be explained below. Each job is described by a triple:

$$(Loc, Dur, Type), \quad (11.15)$$

where Loc is the location of the job (depicted by its  $x$  and  $y$  co-ordinates), Dur is the standard duration of the job and Type indicates whether this job must be done in the morning, in the afternoon, as the first job of the day, as the last job of the day or “don’t care”.

Each engineer is described by a 5-tuple:

$$(Base, ST, ET, OT\_limit, Skill), \quad (11.16)$$

where Base is the  $x$  and  $y$  co-ordinates of the engineer location, ST and ET are this engineer’s starting and ending time, OT\_limit is his/her overtime limit and Skill is a skill factor between 0 and 1 which indicates the fraction of the standard duration that this engineer needs to accomplish a job. The cost function to be minimized is defined as follows:

$$\text{TotalCost} = \sum_{i=1}^{\text{NoT}} \text{TC}_i + \sum_{i=1}^{\text{NoT}} \text{OT}_i^2 + \sum_{j=1}^{\text{NoJ}} (\text{Dur}_j + \text{Penalty}) \times \text{UF}_j, \quad (11.17)$$

where

NoT = number of engineers,

NoJ = number of jobs,

$\text{TC}_i$  = Travelling Cost of engineer  $i$ ,

$\text{OT}_i$  = Overtime of engineer  $i$ ,

$\text{Dur}_j$  = Standard duration of job  $j$ ,

$\text{UF}_j = 1$  if job  $j$  is unallocated; 0 otherwise,

Penalty = constant (which is set to 60 in the tests).

The travelling cost between  $(x_1, y_1)$  and  $(x_2, y_2)$  is defined as follows:

$$TC((x_1, y_1), (x_2, y_2)) = \begin{cases} \frac{\Delta_x}{2} + \Delta_y \\ \frac{8}{\Delta_x + \Delta_y}, & \Delta_x > \Delta_y \\ \frac{2}{8}, & \Delta_y \geq \Delta_x \end{cases}. \quad (11.18)$$

Here  $\Delta_x$  is the absolute difference between  $x_1$  and  $x_2$ , and  $\Delta_y$  is the absolute difference between  $y_1$  and  $y_2$ . The greater of the  $x$  and  $y$  differences is halved before summing. The formula above was specifically designed for the benchmark used in [77] to convert distances into approximate travel times as observed in realistic trips conducted by engineers. Engineers are required to start from and return to their base everyday. An engineer may be assigned more jobs than he/she can finish.

## 11.12.2 Local Search

### 11.12.2.1 Solution Representation

We represent a candidate solution (i.e. a possible schedule) by a permutation of the jobs. Each permutation is mapped into a schedule using the deterministic algorithm described below:

procedure **Evaluation** (input: one particular permutation of jobs)

1. For each job, order the qualified engineers in ascending order of the distances between their bases and the job (such orderings only need to be computed once and recorded for evaluating other permutations).
2. Process one job at a time, following their ordering in the input permutation. For each job  $x$ , try to allocate it to an engineer according to the ordered list of qualified engineers:
  - 2.1. to check if engineer  $g$  can do job  $x$ , make  $x$  the first job of  $g$ ; if that fails to satisfy any of the constraints, make it the second job of  $g$ , and so on;
  - 2.2. if job  $x$  can fit into engineer  $g$ 's current tour, then try to improve  $g$ 's new tour (now with  $x$  in it): the improvement is done by a simple 2-opt algorithm (see Section 11.10), modified in a such a way that only better tours which satisfy the relevant constraints will be accepted;
  - 2.3. if job  $x$  cannot fit into engineer  $g$ 's current tour, then consider the next engineer in the ordered list of qualified engineers for  $x$ ; the job is unallocated if it cannot fit into any engineer's current tour.
3. The cost of the input permutation, which is the cost of the schedule thus created, is returned.

### 11.12.2.2 Construction Method

The starting point of local search is generated heuristically and deterministically: the jobs are ordered by the number of qualified engineers for them. Jobs that can be served by the fewest number of qualified engineers are placed earlier in the permutation.

### 11.12.2.3 Improvement Method

Given a permutation, local search is performed in a simple way: the pairs of jobs are examined one at a time. Two jobs are swapped to generate a new permutation if the new permutation is evaluated (using the Evaluation procedure above) to a lower cost than the original permutation. Note here that since the problem is also close to the Vehicle Routing Problem (VRP), one may follow a totally different approach considering VRP move operators such as insertions and swaps. In this case, the solution representation and construction methods need to be revised. The reader may refer to other works (e.g. [3]) for more information on the application of GLS to the VRP.

### 11.12.3 Guided Local Search

In the workforce scheduling problem, we use the feature type recommended for resource allocation problems in Section 11.9.3. In particular, the inability to serve jobs incurs a cost, which plays the most important part in the objective function. Therefore, we intend to bias local search to serve jobs of high importance. To do so, we define a feature for each job in the problem:

$$I_{\text{job}_j}(\text{schedule}) = \begin{cases} 1, & \text{job}_j \text{ is unallocated in schedule} \\ 0, & \text{job}_j \text{ is allocated in schedule.} \end{cases} \quad (11.19)$$

The cost of this feature is given by  $(\text{Dur}_j + \text{Penalty})$  which is equal to the cost incurred in the cost function (11.17) when a job is unallocated.

The jobs penalized in a local minimum are selected according to the utility function (11.3) which for workforce scheduling takes the form

$$\text{util}(\text{schedule}, \text{job}_j) = I_{\text{job}_j}(\text{schedule}) \cdot \frac{(\text{Dur}_j + \text{Penalty})}{1 + p_j}. \quad (11.20)$$

WSP exhibits properties found in resource allocation problems (i.e. unallocated job costs) and also in routing problems (i.e. travel costs). In addition to the above feature type and for better performance, we may consider introducing a second feature type based on edges as suggested in Section 11.9.1 for routing problems and explained in Section 11.10.3 for the TSP. This feature set can help to aggressively optimize the travel costs also incorporated in the objective function (11.17). Furthermore, one or both feature sets can be used in conjunction with a VRP-based local search method.

### 11.12.4 Guided Fast Local Search

To apply Guided Fast Local Search to workforce scheduling, each job permutation position defines a separate sub-neighbourhood. The activation bits are manipulated according to the general FLS algorithm of Section 11.5. In particular

1. all the activation bits are set to 1 (or “on”) when GFLS starts;
2. the bit for job permutation position  $x$  will be switched to 0 (or “off”) if every possible swap between the job at position  $x$  and the other jobs under the current permutation has been considered, but no better permutation has been found;
3. the bit for job permutation position  $x$  will be switched to 1 whenever  $x$  is involved in a swap which has been accepted.

Mapping penalized jobs to sub-neighbourhoods is straightforward. We simply activate the sub-neighbourhoods corresponding to the permutation positions of the penalized jobs. This essentially forces Fast Local Search to examine moves, which swap the penalized jobs.

## 11.13 Radio Link Frequency Assignment Problem

### 11.13.1 Problem Description

The *Radio Link Frequency Assignment Problem* (RLFAP) [58, 75] is abstracted from the real-life application of assigning frequencies to radio links. The problem belongs to the class of constraint optimization problems mentioned in Section 11.9.4. In brief, the interference level between the frequencies assigned to the different links has to be acceptable; otherwise communication will be distorted. The frequency assignments have to comply with certain regulations and physical characteristics of the transmitters. Moreover, the number of frequencies is to be minimized, because each frequency used in the network has to be reserved at a certain cost. In certain cases, some of the links may have pre-assigned frequencies which may be respected or preferred by the frequency assignment algorithm. Here, we examine a simplified version of the problem considering only the interference constraints. Information on the application of GLS to the full problem can be found in [83]. A definition of the simplified problem is the following.

We are given a set  $L$  of links. For each link  $i$ , a frequency  $f_i$  has to be chosen from a given domain  $D_i$ . Constraints are defined on pairs of links that limit the choice of frequencies for these pairs. For a pair of links  $\{i, j\}$  these constraints are either of type

$$|f_i - f_j| > d_{ij}, \quad (11.21)$$

or of type

$$|f_i - f_j| = d_{ij}, \quad (11.22)$$

for a given distance  $d_{ij} \geq 0$ . Two links  $i$  and  $j$  involved in a constraint of type Equation (11.21) are called *interfering* links, and the corresponding  $d_{ij}$  is the interfering distance. Two links bound by a constraint of type Equation (11.22) are referred to as a pair of *parallel* links; every link belongs to exactly one such pair.

Some of the constraints may be violated at a certain cost. Such restrictions are called *soft*, in contrast to the *hard* constraints, which may not be violated. The

constraints of type Equation (11.22) are always hard. Interference costs  $c_{ij}$  for violating soft constraints of type Equation (11.21) are given. An assignment of frequencies is complete if every link in  $L$  has a frequency assigned to it. We denote by  $C$  the set of all soft *interference* constraints.

The goal is to find a complete assignment that satisfies all hard constraints and is of minimum cost:

$$\min \sum_C c_{ij} \cdot \delta(|f_i - f_j| \leq d_{ij}) \quad (11.23)$$

subject to hard constraints:

- $|f_i - f_j| > d_{ij}$  : for all pairs of links  $\{i, j\}$  involved in the hard constraints,
- $|f_i - f_j| = d_{ij}$  : for all pairs of parallel links  $\{i, j\}$ ,
- $f_i \in D_i$  : for all links  $i \in L$ ,

where  $\delta(\cdot)$  is 1 if the condition within brackets is true and 0 otherwise.

We look next at a local search procedure for the problem.

### 11.13.2 Local Search

#### 11.13.2.1 Using an Alternate Objective Function

When using heuristic search to solve a combinatorial optimization problem, it is not always necessary to use the objective function as dictated in the problem formulation. Objective functions based on the original one can be devised which result in smoother landscapes. These objective functions can sometimes generate solutions of higher quality (with respect to the original objective function) than if the original one is used.

In the RLFAP, we can define and use a simple objective function  $g$ , which is given by the sum of all constraint violation costs in the solution with all the constraints contributing equally to the sum instead of using weights as in Equation (11.23). This objective function is as follows for a given solution  $s$ :

$$g(s) = \sum_{C \cup C^{\text{Hard}}} \delta(|f_i(s) - f_j(s)| \leq d_{ij}), \quad (11.24)$$

subject to hard constraints:

$$f_i(s) \in D'_i : \text{for all links } i \in L,$$

where  $\delta(\cdot)$  is 1 if the condition within brackets is true and 0 otherwise,  $f_i(s)$  is the frequency assigned to link  $i$  in solution  $s$ ,  $C^{\text{Hard}}$  is the set of hard inequality

constraints,  $C$  is the set of soft inequality constraints and  $D'_i$  is the reduced domain for link  $i$  containing only frequencies which satisfy the hard equality constraints.

A solution  $s$  with cost 0 with respect to  $g$  is satisfying all hard and soft constraints of the problem.

The motivation to use an objective function such as Equation (11.24) is closely related to the rugged landscapes formed in RLFAP, if the original cost function is used. In particular, high and very low violation costs are defined for some of the soft constraints. This leads to even higher violation costs to be defined for hard constraints. The landscape is not smooth but full of deep local minima mainly due to the hard and soft constraints of high cost. Soft constraints of low cost are buried under these high costs.

A similar objective function replacement approach has been used successfully by [54] in the MAX-SAT problem suggesting the universal appeal of the idea in constrained optimization problems.

### 11.13.2.2 Solution Representation

An efficient solution representation for the problem takes into account the fact that each link in RLFAP is connected to exactly one other link via a hard constraint of type Equation (11.22). In particular, we can define a decision variable for each pair of parallel links bound by an equality constraint Equation (11.22). The domain of this variable is defined as the set of all pairs of frequencies from the original domains of the parallel links that satisfy the hard equality constraint.

### 11.13.2.3 Construction Method

A construction method can be implemented by assigning to each decision variable (which assigns values to a pair of links) a random value from its domain. In large problem instances, it is beneficial to consider a domain pre-processing and reduction phase. Sophisticated techniques based on Arc-Consistency can be utilized during that phase to reduce the domain based on the problem's hard constraints. These domains can then be used instead of the original ones for the random solution generation and also by the improvement method.

### 11.13.2.4 Improvement Method

An improvement method can be based on the min-conflicts heuristic of Minton et al. [56] for Constraint Satisfaction Problems. A 1-optimal type move is used which changes the value of one variable at a time. Starting from a random and complete assignment of values to variables, variables are examined in an arbitrary static order. Each time a variable is examined, the current value of the variable changes to the value (in the variable's domain) which yields the minimum value for the objective

function. Ties are randomly resolved allowing moves to solutions with equal cost. These moves are called *sideways moves* [65] and enable the local search to examine plateaus of solutions that occur in the landscapes of many constrained optimization problems.

### 11.13.3 Guided Local Search

The most important cost factor in the RLFAP is the constraint violation costs defined for soft inequality constraints. Inequality constraints can be used to define a basic feature set for the RLFAP. Each inequality constraint is interpreted as a feature with the feature cost given by the constraint violation cost  $c_{ij}$  as defined in the problem's original cost function (11.23).

Hard inequality constraints are also modelled as features by assigning to them an infinite cost. This results in their utility to be penalized to also tend to infinity. To implement this in the code, hard constraints are simply given priority over soft constraints when penalties are applied. This basically forces local search to return back to a feasible region where penalizing soft constraints can resume.

GLS is especially suited to use the alternate objective function (11.24) because of the definition of the feature costs described above. The application of penalties can still force the local search toward solutions which satisfy constraints with high violation costs while the algorithm is benefiting from the smoother landscape introduced by (11.24).

The  $\lambda$  parameter can be set to 1 provided that we use (11.24) as the objective function. The same value for  $\lambda$  has also been used in MAX-SAT problems in [54] where the same approach is followed with respect to smoothing the landscape.

A variation of the GLS method which seems to significantly improve performance in certain RLFAP instances is to decrease penalties and not only increase them [83]. More specifically, the variant uses a circular list to retract the effects of penalty increases made earlier in the search process, in a way that very much resembles a tabu list. In particular, increased penalties are decreased after a certain number of increases. The scheme uses an array of size  $t$  where the  $t$  most recent features penalized are recorded. The array is treated as a circular list, adding elements in sequence in positions 1 through  $t$  and then starting over at position 1. Each time the penalty of a feature is increased (by one unit), the feature is inserted in the array and the penalty of the feature previously stored in the same position is decreased (by one unit). The rationale behind the strategy is to allow GLS to return to regions of the search visited earlier in the search process, so introducing a search intensification mechanism.

### 11.13.4 Guided Fast Local Search

Best improvement local search for the RLFAP as used in the context of Tabu Search (for an example, see [31]), evaluates all possible 1-optimal moves over all variables

before selecting and performing the best move. Given the large number of links in real-world instances, greedy local search is a computationally expensive option. This is especially the case for the RLFAP where we cannot easily devise an incremental move update mechanism (such as the one for the QAP) for all the problem's variations. The local search procedure described in Section 11.13.2 is already a faster alternative than best improvement. Using Guided Fast Local Search, things can be improved further.

To apply Guided Fast Local Search to RLFAP, each decision variable defines a sub-neighbourhood and has a bit associated with it. Whenever a variable is examined and its value is changed (i.e. the variable's parallel links are assigned to another pair of frequencies because of an improving or sideway move) the activation bit of the variable remains to 1 otherwise it turns to 0 and the variable is excluded in future iterations of the improvement loop. Additionally, if a move is performed, activation spreads to other variables which have their bits set to 1. In particular, we set to 1 the bit of variables for which improving moves may occur as a result of the move just performed. They are the variables for which one of their links is connected via a constraint to one of the links of the variable with a modified value. There are five potential schemes for propagating activation after changing the value of a variable. They are the following:

1. Activate all variables connected via a constraint to the variable with a modified value.
2. Activate only variables that are connected via a constraint which is violated. This resembles CSP local search methods where only variables in conflict have their neighbourhood searched.
3. Activate only variables that are connected via a constraint which has become violated as a result of the move (subset of condition 2 and also condition 4).
4. Activate only variables that are connected via a constraint that changed from violated to satisfied or from satisfied to violated, as a result of the move (superset of condition 3).
5. Activate variables that fall under either condition 2 or 4.

Experimentation suggests that scheme 5 tends to produce better results for the real-world instances of RLFAP available in the literature. Fast local search stops when all the variables are inactive or when a local minimum is detected by other means (i.e. a number of sideway moves is performed without an improving move found).

Finally, when a constraint is penalized we activate the variables connected via the constraint in an effort to find 1-Opt moves which will satisfy the constraint.

## 11.14 Summary and Conclusions

For many years, general heuristics for combinatorial optimization problems, with prominent examples such as Simulated Annealing and Genetic Algorithms, heavily relied on randomness to generate good approximate solutions to difficult NP-Hard

problems. The introduction and acceptance of Tabu Search [25] by the Operations Research community initiated an important new era for heuristic methods where deterministic algorithms exploiting historical information started to appear and to be used in real-world applications.

Guided local search described in this chapter follows this trend. While Tabu search is a class of algorithms (where a lot of freedom is given to the management of the tabu list), GLS is more prescriptive (the procedures are more concretely defined). GLS heavily exploits information (not only the search history) to distribute the search effort in the various regions of the search space. Important structural properties of solutions are captured by solution features. Solution features are assigned costs and local search is biased to spend its efforts according to these costs. Penalties on features are utilized for that purpose.

When local search settles in a local minimum, the penalties are increased for selected features present in the local minimum. By penalizing features appearing in local minima, GLS not only escapes the local minima visited (exploiting historical information) but also diversifies the choices, with regard to the various structural properties of solutions, as captured by the solution features. Features of high costs are penalized more often than features of low cost: the diversification process is directed and deterministic rather than undirected and random.

In general, several penalty cycles may be required before a move is executed out of a local minimum. This should not be viewed as an undesirable situation. It is caused by the uncertainty in the information as captured by the feature costs which forces the GLS to test its decisions against the landscape of the problem.

The penalization scheme of GLS is ideally combined with FLS which limits the neighbourhood search to particular parts of the overall solution leading to the GFLS algorithm. GFLS significantly reduces the computation times required to explore the area around a local minimum to find the best escape route allowing many more penalty modification cycles to be performed in a given amount of running time.

The GLS and GFLS methods are still in their early stages and future research is required to develop them further. The use of incentives implemented as negative penalties, which encourage the use of specific solution features, is one promising direction to be explored. Other interesting directions include *fuzzy features* with indicator functions returning real values in the  $[0, 1]$  interval, automated tuning of the  $\lambda$  or  $\alpha$  parameters, definition of effective termination criteria, alternative utility functions for selecting the features to be penalized and also studies about the convergence properties of GLS.

It is relatively easy to adapt GLS and GFLS to the different problems examined in this chapter. Although local search is problem dependent, the other structures of GLS and also GFLS are problem independent. Moreover, a mechanical, step-by-step procedure is usually followed when GLS or GFLS is applied to a new problem (i.e. implement a local search procedure, identify features, assign costs and define sub-neighbourhoods). This makes GLS and GFLS easier to use by non-specialist software engineers.

## References

1. Anderson, C.A., Fraughnaugh, K., Parker, M., Ryan, J.: Path assignment for call routing: An application of tabu search. *Ann. Oper. Res.* **41**, 301–312 (1993)
2. Azarmi, N. and Abdul-Hameed, W.: Workforce scheduling with constraint logic programming. *BT Technol. J.* **13**:1, 81–94 (1995)
3. Backer, B.D., Furnon, V., Shaw, P., Kilby, P., Prosser, P.: Solving vehicle routing problems using constraint programming and metaheuristics. *J. Heuristics* **6**:4, 501–523 (2000)
4. Basharu, M., Arana, I., Ahriz, H.: Distributed guided local search for solving binary DisCSPs. *Proceedings of FLAIRS 2005*, Florida, USA, AAAI Press, pp. 660–665 (2005)
5. Bentley, J.J.: Fast algorithms for geometric traveling salesman problems. *ORSA J. Comput.* **4**, 387–411 (1992)
6. Beullens, P., Muyldermans, L., Cattrysse, D., Van Oudheusden, D.: A guided local search heuristic for the capacitated arc routing problem. *Eur. J. Oper. Res.* **147**:3, 629–643 (2003)
7. Bouju, A., Boyce, J.F., Dimitropoulos, C.H.D., vom Scheidt, G., Taylor, J.G.: Intelligent search for the radio link frequency assignment problem. *Proceedings of the International Conference on Digital Signal Processing*, Cyprus (1995)
8. Burkard, R.E., Karisch, S.E., Rendl F.: QAPLIB - A Quadratic assignment problem library. *J. Global Optim.* **10**, 391–403 (1997)
9. Chalmers, A.G.: A minimum path parallel processing environment. *Research Monographs in Computer Science*, Alpha Books (1994)
10. Chiarandini, M. and Stutzle, T.: Stochastic local search algorithms for graph set T-colouring and frequency assignment. *Constraints* **12**(3), 371–403 (2007)
11. Chu, P., Beasley, J.E.: A genetic algorithm for the generalized assignment problem. *Comput. Oper. Res.* **24**, 17–23 (1997)
12. Congram, R.K., Potts, C.N.: Dynasearch Algorithms for the traveling salesman problem. Presentation at the Travelling Salesman Workshop, CORMSIS, University of Southampton, Southampton, UK (1999)
13. Croes, A.: A method for solving traveling-salesman problems. *Oper. Res.* **5**, 791–812 (1958)
14. Daum, M., Menzel, W.: Parsing natural language using guided local search. *Proceedings of 15th European Conference on Artificial Intelligence (ECAI-2002)*, pp. 435–439 (2002)
15. Davenport, A., Tsang, E.P.K., Wang, C.J., Zhu, K.: GENET: a connectionist architecture for solving constraint satisfaction problems by iterative improvement. *Proceedings of 12th National Conference for Artificial Intelligence (AAAI)*, 325–330, Seattle, WA, USA (1994)
16. Dorne, R., Mills, P., Voudouris, C.: Solving vehicle routing using iOpt. In: Doerner, K.F. et al. (eds.) *Metaheuristics: Progress in Complex Systems Optimization*. Operations Research/Computer Science Interfaces Series, vol. **39**, pp. 389–408, Springer, New York (2007)
17. Dorne, R., Voudouris, C., Liret, A., Ladde, C., Lesaint, D.: iSchedule an optimisation toolkit based on heuristic search to solve BT scheduling problems. *BT Technol. J.* **21**:4, 50–58 (2003)
18. Egeblad, J., Nielsen, B., Odgaard, A.: Fast neighbourhood search for two- and three-dimensional nesting problems. *Eur. J. Oper. Res.* **183**(3), 1249–1266 (2007)
19. Faroe, O., Pisinger, D., Zachariasen, M.: Guided local search for the three-dimensional bin packing problem. *Tech. Rep. 99-13*, Department of Computer Science, University of Copenhagen. (1999)
20. Faroe, O., Pisinger, D., Zachariasen, M.: Guided local search for final placement in VLSI design. *J. Heuristics* **9**, 269–295 (2003)
21. Flood, M.M.: The traveling-salesman problem. *Oper. Res.* **4**, 61–75 (1956)
22. Flores Lucio, G., Reed, M., Henning, I.: Guided local search as a network planning algorithm that incorporates uncertain traffic demands. *Comput. Net.* **51**(11), 3172–3196 (2007)
23. Freisleben, B., Merz, P.: A genetic local search algorithm for solving symmetric and asymmetric travelling salesman problems. *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, IEEE Press, pp. 616–621, Nayoya University, Japan (1996)

24. Gent, I.P., van Maaren, H., Walsh, T.: SAT2000, Highlights of satisfiability research in the year 2000. *Frontiers in Artificial Intelligence and Applications*, IOS Press. (2000)
25. Glover, F., Laguna, M.: *Tabu Search*. Kluwer Academic Publishers, Boston (1997)
26. GLS Demos: <http://cswww.essex.ac.uk/CSP/glsdemo.html> (2008)
27. Goldberg, D.E.: *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Pub. Co., Inc., Reading, MA (1989)
28. Gomes, N., Vale, Z., Ramos, C.: Hybrid Constraint algorithm for the maintenance scheduling of electric power units. *Proceedings of International Conference on Intelligent Systems Application to Power Systems (ISAP 2003)*, Lemnos, Greece (2003)
29. Hani, Y., Amodeo, L., Yalaoui, F., Chen, H.: Ant colony optimization for solving an industrial layout problem. *Eur. J. Oper. Res.* **183**(2), 633–642 (2007)
30. Hansen, P., Mladenovic, N.: An introduction to variable neighbourhood search. *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, In: Voss, S., Martello, S., Osman, I.H., Roucairol, C. (eds.) pp. 433–458. Kluwer, Boston (1999)
31. Hao J.-K., Dorne, R., Galinier, P.: Tabu search for frequency assignment in mobile radio networks. *J. Heuristics* **4**(1), 47–62 (1998)
32. Hifi, M., Michrafy, M., Sbihi, A.: Heuristic algorithms for the multiple-choice multidimensional knapsack problem. *J. Oper. Res. Soc.* **55**, 1323–1332 (2004)
33. Holland, J.H.: *Adaptation in natural and artificial systems*. University of Michigan Press, Ann Arbor, MI (1975)
34. Holstein, D., Moscato, P.: Memetic algorithms using guided local search: a case study. Corne, D., Glover, F., Dorigo, M., (eds.) *New Ideas in Optimisation*, pp. 235–244, McGraw-Hill, London (1999)
35. Hoos, H., Tsang, E.P.K.: Local search for constraint satisfaction, Chapter 5. Rossi, F., van Beek P., Walsh T. (eds.), *Handbook of Constraint Programming*, pp. 245–277 Elsevier, Amsterdam, The Netherlands (2006)
36. Johnson, D.: Local optimization and the traveling salesman problem. *Proceedings of the 17th Colloquium on Automata Languages and Programming, Lecture Notes in Computer Science*, vol **443**, pp. 446–461, Springer, Berlin (1990)
37. Jose, R., Boyce, J.: Application of connectionist local search to line management rail traffic control. *Proceedings of International Conf. on Practical Applications of Constraint Technology (PACT'97)*, London (1997)
38. Kilby, P., Prosser, P., Shaw, P.: Guided local search for the vehicle routing problem with time windows. In: Voss, S., Martello, S., Osman, I.H., Roucairol, C. (eds.), *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, pp. 473–486 Kluwer Academic Publishers, (1999)
39. Kilby, P., Prosser, P., Shaw, P.: A comparison of traditional and constraint-based heuristic methods on vehicle routing problems with side constraints. *Constraints* **5**(4), 389–414 (2000)
40. Knox, J.: Tabu search performance on the symmetric traveling salesman problem. *Comput. Oper. Res.* **21**(8), 867–876 (1994)
41. Koopman, B.O.: The theory of search, part III, the optimum distribution of search effort. *Oper. Res.* **5**, 613–626 (1957)
42. Kytjoki, J., Nuortio, T., Brysy, O., Gendreau, M.: An efficient variable neighbourhood search heuristic for very large scale vehicle routing problems. *Comput. Oper. Res.* **34**:9, 2743–2757 (2007)
43. Langer, Y., Bay, M., Crama, Y., Bair, F., Caprace, J.D., Rigo, P.: Optimization of surface utilization using heuristic approaches. *Proceedings of the International Conference COMPIT'05*, pp. 419–425 (2005)
44. Lau, T.L.: Guided Genetic Algorithm. PhD Thesis, Department of Computer Science, University of Essex, Colchester, UK. (1999)
45. Lau, T.L., Tsang, E.P.K.: Solving the processor configuration problem with a mutation-based genetic algorithm. *Int. J. Artif. Intell. Tools (IJAIT)* **6**(4), 567–585 (1997)
46. Lau, T.L., Tsang, E.P.K.: Guided genetic algorithm and its application to the radio link frequency allocation problem. *Proceedings of NATO symposium on Frequency Assignment*,

- Sharing and Conservation in Systems (AEROSPACE), AGARD, RTO-MP-13, paper No. 14b. (1998)
- 47. Lau, T.L., Tsang, E.P.K.: The guided genetic algorithm and its application to the general assignment problem. IEEE 10th International Conference on Tools with Artificial Intelligence (ICTAI'98), Taiwan, 336–343 (1998)
  - 48. Lee, J.H.M., Tam, V.W.L.: A framework for integrating artificial neural networks and logic programming. *Int. J. Artif. Intell. Tools* **4**, 3–32 (1995)
  - 49. Lin, S.: Computer solutions of the traveling-salesman problem. *Bell Syst. Tech. J.* **44**, 2245–2269 (1965)
  - 50. Lin, S., Kernighan, B.W.: An effective heuristic algorithm for the traveling salesman problem. *Oper. Res.* **21**, 498–516 (1973)
  - 51. Martin, O., Otto, S.W.: Combining simulated annealing with local search heuristics. Laporte G., Osman I.H. (eds.), *Metaheuristics in Combinatorial Optimization*, Ann. Oper. Res. 63 (1996)
  - 52. Mester, D., Brysy, O.: Active guided evolution strategies for large-scale vehicle routing problems with time windows. *Comput. Oper. Res.* **32**(6), 1593–1614 (2005)
  - 53. Mester, D.I., Ronin, Y. I., Nevo, E., Korol, A.B.: Fast and high precision algorithms for optimization in large-scale genomic problems. *Comput. Biol. Chem.* **28**(4), 281–290 (2004)
  - 54. Mills, P., Tsang, E.P.K.: Guided local search for solving SAT and weighted MAX-SAT problems. *J. Automat. Reas.* **24**, 205–223 (2000)
  - 55. Mills P., Tsang E., Ford J.: Applying an extended guided local search to the quadratic assignment problem. *Ann. Oper. Res.* **118**(1–4), 121–135 (2003)
  - 56. Minton S., Johnston, M.D., Philips A.B., Laird, P.: Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. *Artif. Intell.* **58**(1–3), (Special Volume on Constraint Based Reasoning), 161–205 (1992)
  - 57. Moghrabi, I.: Guided local search for query reformulation using weight propagation. *Int. J. Appl. Mathe. Comput. Sci. (AMCS)* **16**(4), 537–549 (2006)
  - 58. Murphrey, R.A., Pardalos, P.M., Resende, M.G.C.: Frequency assignment problems. Du D.-Z., Pardalos, P. (eds.) *Handbook of Combinatorial Optimization* vol. 4, Kluwer Academic Publishers, Dordrecht, The Netherlands (1999)
  - 59. Padron, V., Balaguer, C.: New methodology to solve the RPP by means of isolated edge. 2000 Cambridge Conference Tutorial Papers. In: Tuson, A. (ed.) *Young OR 11*, UK Operational Research Society (2000)
  - 60. Pesant, G., Gendreau, M.: A constraint programming framework for local search methods. *J. Heuristics* **5**(3), 255–279 (1999)
  - 61. Reinelt, G.: A traveling salesman problem library. *ORSA J. Comput.* **3**, 376–384 (1991)
  - 62. Reinelt, G.: *The Traveling Salesman: Computational Solutions for TSP Applications*. Lecture Notes in Computer Science 840, Springer, Berlin (1995)
  - 63. Resende, M.G.C., Feo, T.A.: A GRASP for satisfiability. Cliques, coloring, and satisfiability: Second DIMACS implementation challenge. In: Johnson D.S., Trick, M.A. (eds.) *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, vol. 26, pp. 499–520. American Mathematical Society, (1996)
  - 64. Selman, B., Kautz, H.: Domain-independent extensions to GSAT: solving large structured satisfiability problems. Proceedings of 13th International Joint Conference on AI, pp. 290–295, Chambery, France (1993)
  - 65. Selman, B., Levesque, H.J., Mitchell, D.G.: A new method for solving hard satisfiability problems. Proceedings of AAAI-92, pp. 440–446, San Jose, CA, USA (1992)
  - 66. Shang, Y., Wah, B.W.: A discrete lagrangian-based global-search method for solving satisfiability problems. *J. Global Optim.* **12**(1), 61–99 (1998)
  - 67. Simon, H.U.: Approximation algorithms for channel assignment in cellular radio networks. Proceedings of 7th International Symposium on Fundamentals of Computation Theory, Lecture Notes in Computer Science 380, pp. 405–416, Springer, Berlin (1989)
  - 68. Stone, L.D.: The process of search planning: current approaches and continuing problems. *Oper. Res.* **31**, 207–233 (1983)

69. Stuckey, P., Tam, V.: Semantics for using stochastic constraint solvers in constraint logic programming. *J. Funct. Logic Programming* **2** (1998)
70. Taillard, E.: Robust taboo search for the QAP. *Parallel Comput.* **17**, 443–455 (1991)
71. Taillard, E.: Comparison of iterative searches for the quadratic assignment problem. *Location Sci.* **3**, 87–105 (1995)
72. Tamura, H., Zhang, Z., Tang, Z., Ishii, M.: Objective function adjustment algorithm for combinatorial optimization problems. *IEICE Trans. Fundamentals of Electronics, Communications and Comput. Sci.* **E89-A**:9, 2441–2444 (2006)
73. Tarantilis, C.D., Zachariadis, E.E., Kiranoudis, C.T.: A guided tabu search for the heterogeneous vehicle routeing problem. *J. Oper. Res. Soc.* **59**, 1659–1673 (2008)
74. Tarantilis, C.D., Zachariadis, E.E., Kiranoudis, C.T.: A hybrid guided local search for the vehicle-routing problem with intermediate replenishment facilities. *INFORMS J. Comput.* **20**(1), 154–168 (2008)
75. Tiourine, S., Hurkens, C., Lenstra, J.K.: An overview of algorithmic approaches to frequency assignment problems. *EUCLID CALMA Project Overview Report*, Delft University of Technology, The Netherlands (1995)
76. Tsang, E.P.K.: Foundations of constraint satisfaction, Academic Press, London (1993)
77. Tsang, E.P.K., Voudouris, C.: Fast local search and guided local search and their application to British Telecom's workforce scheduling problem. *Oper. Res. Lett.* **20**(3), 119–127 (1997)
78. Tsang, E.P.K., Wang, C.J.: A generic neural network approach for constraint satisfaction problems. Taylor, J.G. (ed.) *Neural Network Applications*, pp. 12–22 Springer, Berlin (1992)
79. Tsang, E.P.K., Wang, C.J., Davenport, A., Voudouris, C., Lau, T.L.: A family of stochastic methods for constrain satisfaction and optimisation. *Proceedings of the First International Conference on The Practical Application of Constraint Technologies and Logic Programming (PACLP)*, London, pp. 359–383 (1999)
80. Vansteenwegen, P., Souffriau, W., Berghe, G., Oudheusden, D.: A guided local search meta-heuristic for the team orienteering problem. *Eur. J. Oper. Res.* **196**(1), 118–127 (2009)
81. Voudouris, C.: Guided Local Search for Combinatorial Optimisation Problems. PhD Thesis, Department of Computer Science, University of Essex, Colchester, UK (1997)
82. Voudouris, C.: Guided local search an illustrative example in function optimisation. *BT Technol. J.* **16**(3), 46–50 (1998)
83. Voudouris, C., Tsang, E.: Solving the radio link frequency assignment problems using guided local search. *Proceedings of NATO symposium on Frequency Assignment, Sharing and Conservation in Systems (AEROSPACE)*, AGARD, RTO-MP-13, paper No. 14a (1998)
84. Voudouris, C., Tsang, E.P.K.: Partial constraint satisfaction problems and guided local search. *Proceedings of PACT'96*, London, pp. 337–356 (1996)
85. Voudouris, C., Tsang, E.P.K.: Guided local search and its application to the travelling salesman problem. *Eur. J. Oper. Res.* **113**(2), 469–499 (1999)
86. Voudouris, C., Dorne, R., Lesaint, D., Liret, A.: iOpt: A Software Toolkit for Heuristic Search Methods. *Principles and Practice of Constraint Programming – CP 2001* In: Walsh, T. (ed.) *Lecture Notes in Computer Science*, vol. **2239**, pp. 716–729 Springer, Heidelberg (2001)
87. Wang, C.J., Tsang, E.P.K.: Solving constraint satisfaction problems using neural-networks. *Proceedings of the IEE Second International Conference on Artificial Neural Networks*, pp. 295–299, Bournemouth, UK (1991)
88. Wang, C.J., Tsang, E.P.K.: A cascadable VLSI design for GENET. In: *VLSI for Neural Networks and Artificial Intelligence*, Delgado-Frias J.G., Moore, W.R. (eds.) pp. 187–196 Plenum Press, New York (1994)
89. Xiaohu, T., Haubrich, H.-J.: A hybrid metaheuristic method for the planning of medium-voltage distribution networks. *Proceedings of 15th Power Systems Computation Conference (PSCC 2005)*, Liege, Belgium (2005)
90. Zachariadis, E., Tarantilis, C., Kiranoudis, C.: A Guided Tabu Search for the Vehicle Routing Problem with two-dimensional loading constraints. *Eur. J. Oper. Res.* **195**(3), 729–743 (2009)
91. Zachariadis, E., Tarantilis, C., Kiranoudis, C.: A hybrid metaheuristic algorithm for the vehicle routing problem with simultaneous delivery and pick-up service. *Exp. Syst. Appl.* **36**(2), 1070–1081 (2009)

92. Zhang, Q., Sun, J., Tsang, E.P.K, Ford, J.: Combination of guided local search and estimation of distribution algorithm for solving quadratic assignment problem. Bird of a Feather Workshops, Genetic and Evolutionary Computation Conference. (2003)
93. Zhong, Y., Cole, M. H.: A vehicle routing problem with backhauls and time windows: a guided local search solution. Transport Res. E: Logistics Transport Rev. **41**(2), 131–144 (2005)



# Chapter 12

## Iterated Local Search: Framework and Applications

Helena R. Lourenço, Olivier C. Martin and Thomas Stützle

**Abstract** The key idea underlying iterated local search is to focus the search not on the full space of all candidate solutions but on the solutions that are returned by some underlying algorithm, typically a local search heuristic. The resulting search behavior can be characterized as iteratively building a chain of solutions of this embedded algorithm. The result is also a conceptually simple metaheuristic that nevertheless has led to state-of-the-art algorithms for many computationally hard problems. In fact, very good performance is often already obtained by rather straightforward implementations of the metaheuristic. In addition, the modular architecture of iterated local search makes it very suitable for an algorithm engineering approach where, progressively, the algorithms' performance can be further optimized. Our purpose here is to give an accessible description of the underlying principles of iterated local search and a discussion of the main aspects that need to be taken into account for a successful application of it. In addition, we review the most important applications of this method and discuss its relationship to other metaheuristics.

### 12.1 Introduction

The importance of high-performance algorithms for tackling difficult optimization problems cannot be understated, and in many cases the most effective methods are metaheuristics. When designing a metaheuristic, simplicity should be favored, both

---

Helena R. Lourenço,  
Universitat Pompeu Fabra, Barcelona, Spain  
e-mail: helena.ramalhinho@upf.edu

Olivier C. Martin  
Université Paris-Sud, Orsay, France  
e-mail: olivier.martin@u-psud.fr

Thomas Stützle,  
IRIDIA, Université Libre de Bruxelles (ULB), Brussels, Belgium  
e-mail: stuetzle@ulb.ac.be

conceptually and in practice. Naturally, it must also lead to effective algorithms, and, if possible, general-purpose ones. If we think of a metaheuristic as simply a construction for guiding (problem-specific) heuristics, the ideal case is when the metaheuristic can be used without *any* problem-dependent knowledge.

As metaheuristics have become more and more sophisticated, this ideal case has been pushed aside in the quest for greater performance. As a consequence, problem-specific knowledge (in addition to that built into the heuristic being guided) must now be incorporated into metaheuristics in order to reach state-of-the-art level. Unfortunately, this makes the boundary between heuristics and *metaheuristics* fuzzy, and we run the risk of loosing both simplicity and generality. To counter this, we appeal to modularity and try to decompose a metaheuristic algorithm into a few parts, each with its own specificity. In particular, we would like to have a totally general-purpose part, so that any problem-specific knowledge built into the metaheuristic would be restricted to another part. Finally, to the extent possible, we prefer to leave untouched the embedded heuristic (which is to be “guided”) because of its potential complexity. One can also consider the case where this heuristic is only available through an object module, the source code being proprietary; it is then necessary to be able to treat it as a “black-box” routine. Iterated local search provides a simple way to satisfy all these requirements.

The essence of iterated local search can be given in a nut-shell: one *iteratively* builds a sequence of solutions generated by the embedded heuristic, leading to far better solutions than if one were to use repeated random trials of that heuristic. This simple idea [9] has a long history, and its rediscovery by many authors has led to many different names for iterated local search like *iterated descent* [7, 8], *large-step Markov chains* [62], *iterated Lin-Kernighan* [46], *chained local optimization* [61], and combinations of these [1]. Readers interested in these historical developments should consult the review in [47]. For us, there are two main points that make an algorithm an iterated local search: (i) there must be a single chain that is being followed (this then excludes population-based algorithms); (ii) the search for better solutions occurs in a reduced space defined by the output of a black-box heuristic. In practice, local search has been the most frequently used embedded heuristic, but in fact any optimizer can be used, be it deterministic or not.

The purpose of this review is to give a detailed description of iterated local search and to show where it stands in terms of performance. So far, in spite of its conceptual simplicity, it has led to a number of state-of-the-art results without the use of too much problem-specific knowledge; perhaps this is because iterated local search is very malleable, many implementation choices being left to the developer.

We have organized this chapter as follows. First we give a high-level presentation of iterated local search in Section 12.2. Then we discuss the importance of the different parts of the metaheuristic in Section 12.3, especially the subtleties associated with perturbing the solutions. In Section 12.4 we go over past work aimed at testing iterated local search in practice, while in Section 12.5 we discuss similarities and differences between iterated local search and other metaheuristics. This chapter closes with a summary of what has been achieved so far and an outlook on what the near future may look like.

## 12.2 Iterating a Local Search

### 12.2.1 General Framework

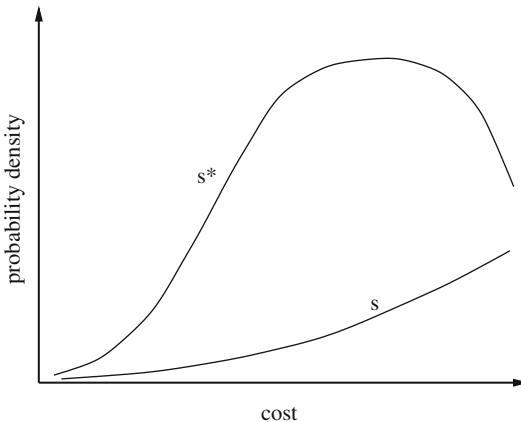
We assume that we have been given a problem-specific heuristic optimization algorithm that from now on we shall refer to as a local search (even if it is not a true local search). This algorithm is implemented via a computer routine that we call `LocalSearch`. The question we ask is “Can such an algorithm be improved by the use of iteration?” Our answer is “YES,” and in fact the improvements obtained in practice are usually significant. Only in rather pathological cases where the iteration method is “incompatible” with the local search will the improvement be minimal. In the same vein, in order to have the *largest* possible improvement, it is necessary to have some understanding of the way the `LocalSearch` works. However, to keep this presentation as simple as possible, we shall ignore for the time being these complications; the additional subtleties associated with tuning the iteration to the local search procedure will be discussed in Section 12.3. Furthermore, all issues associated with the actual speed of the algorithm are omitted in this first section as we wish to focus solely on the high-level architecture of iterated local search.

Let  $\mathcal{C}$  be the cost function of our combinatorial optimization problem;  $\mathcal{C}$  is to be *minimized*. We label candidate solutions or simply “solutions” by  $s$  and denote by  $\mathcal{S}$  the set of all  $s$  (for simplicity  $\mathcal{S}$  is taken to be finite, but it does not matter much). Finally, for the purposes of this high-level presentation, we assume that the local search procedure is deterministic and memoryless<sup>1</sup>: for a given input  $s$ , it always outputs the same solution  $s^*$  whose cost is less than or equal to  $\mathcal{C}(s)$ . `LocalSearch` then defines a many to one mapping from the set  $\mathcal{S}$  to the smaller set  $\mathcal{S}^* = \{s^*\}$  of locally optimal solutions. To have a pictorial view of this, we introduce the “basin of attraction” of a local minimum  $s^*$  as the set of solutions  $s$  that are mapped to  $s^*$  under the local search routine. `LocalSearch` then takes an  $s \in \mathcal{S}$  as a starting solution and produces a local optimum  $s^* \in \mathcal{S}^*$  at the bottom of the corresponding basin of attraction.

Now take an  $s$  or an  $s^*$  at random. Typically, the cost distribution has a very rapidly rising part at the lowest values. In Figure 12.1 we show the kind of distributions found in practice for combinatorial optimization problems having a finite solution space. The distribution of costs is bell shaped, with a mean and variance that is significantly smaller for solutions in  $\mathcal{S}^*$  than for those in  $\mathcal{S}$ . As a consequence, it is much better to use local search than to sample randomly in  $\mathcal{S}$  if one seeks low-cost solutions. The essential ingredient necessary for local search is a neighborhood structure. This means that  $\mathcal{S}$  is a “space” with some topological structure, not just a set. Having such a space allows one to move from one solution  $s$  to a better one in an intelligent way, something that would not be possible if  $\mathcal{S}$  were just a set.

---

<sup>1</sup> The reader can check that very little of what we say really uses this property, and in practice, many successful implementations of iterated local search have non-deterministic local searches or include memory.



**Fig. 12.1** Probability densities of costs. The curve labeled  $s$  indicates the left tail of the cost density function for all solutions, while the curve labeled  $s^*$  indicates the cost density function for the solutions that are local optima.

Now the question is how to go beyond this use of LocalSearch. More precisely, given the mapping from  $\mathcal{S}$  to  $\mathcal{S}^*$ , how can one further reduce the costs found without opening up and modifying LocalSearch, leaving it as a “black box” routine?

### 12.2.2 Random Restart

The simplest possibility to improve upon a cost found by LocalSearch is to repeat the search from another starting point. Every  $s^*$  generated is then independent, and the use of multiple trials allows one to reach the lower part of the distribution. Although such a “random restart” approach with independent samplings is sometimes a useful strategy (in particular when all other options fail), it breaks down as the instance size grows because in the limit, the tail of the cost distribution collapses. Indeed, empirical studies [47] and general arguments [79] indicate that local search algorithms on large generic instances lead to costs that (i) have a mean that is a fixed percentage above the optimum cost; (ii) have a *distribution* that becomes arbitrarily peaked around the mean when the instance size goes to infinity. This second property makes it impossible in practice to find an  $s^*$  whose cost is even a little bit lower percentage-wise than the typical cost. Note, however, that there do exist many solutions of significantly lower cost, it is just that *random* sampling has a lower and lower probability of finding them as the instance size increases. To reach those configurations, a biased sampling is necessary; this is precisely what is accomplished by a stochastic search.

### 12.2.3 Searching in $\mathcal{S}^*$

To overcome the problem just mentioned associated with large instance sizes, reconsider what local search does: it takes one solution from  $\mathcal{S}$  where  $\mathcal{C}$  has a large

mean to a solution in  $\mathcal{S}^*$  where  $\mathcal{C}$  has a smaller mean. It is then natural to invoke recursion: use local search to go from  $\mathcal{S}^*$  to a smaller space  $\mathcal{S}^{**}$  where the mean cost is even lower! That would correspond to an algorithm with one local search nested inside another. Such a construction could be iterated for as many levels as desired, leading to a hierarchy of nested local searches. But upon closer scrutiny, we see that the problem is precisely how to formulate local search beyond the lowest level of the hierarchy: local search requires a neighborhood structure and this is not a priori given. The fundamental difficulty is to define neighbors in  $\mathcal{S}^*$  so that they can be enumerated and accessed efficiently. Furthermore, it is desirable for neighbors in  $\mathcal{S}^*$  to be relatively close according to the distance metric defined in space  $\mathcal{S}$ ; if this were not the case, a stochastic search on  $\mathcal{S}^*$  would have little chance of being effective.

Upon further thought, it transpires that one can introduce a good neighborhood structure on  $\mathcal{S}^*$  as follows. First, one recalls that a neighborhood structure on set  $\mathcal{S}$  directly induces a neighborhood structure on *subsets* of  $\mathcal{S}$ : two subsets are neighbors simply if they contain solutions that are neighbors. Second, take these subsets to be the basins of attraction of the solutions in  $\mathcal{S}^*$ ; this leads us to associate any  $s^* \in \mathcal{S}^*$  with its basin of attraction. Then, this immediately provides the “canonical” notion of neighborhood on  $\mathcal{S}^*$ , which can be stated in a simple way:  $s_1^*$  and  $s_2^*$  are neighbors in  $\mathcal{S}^*$  if their basins of attraction intersect (i.e., they contain neighbor solutions in  $\mathcal{S}$ ). Unfortunately this definition has the major drawback that one cannot in practice list the neighbors of  $s^*$  because there is no computationally efficient method for finding all solutions  $s$  in the basin of attraction of  $s^*$ . Nevertheless, we can *stochastically* generate neighbors as follows. Starting from  $s^*$ , create a randomized path in  $\mathcal{S}$ ,  $s_1, s_2, \dots, s_i$ , where  $s_{j+1}$  is a neighbor of  $s_j$ . Determine the first  $s_j$  in this path that belongs to a different basin of attraction so that applying local search to  $s_j$  leads to  $s^{*'} \neq s^*$ . Then  $s^{*'}$  is a neighbor of  $s^*$ .

Given this procedure, we can in principle perform a local search<sup>2</sup> in  $\mathcal{S}^*$ . Extending the argument recursively, we see that it would be possible to have an algorithm implementing nested searches, performing local search on  $\mathcal{S}$ ,  $\mathcal{S}^*$ ,  $\mathcal{S}^{**}$ , and so on, in a hierarchical way. Unfortunately, the implementation of a neighbor search at the level of  $\mathcal{S}^*$  is too costly computationally because of the number of times one has to execute LocalSearch. Thus we are led to abandon the (stochastic) search for neighbors in  $\mathcal{S}^*$ ; instead we use a weaker notion of closeness which then allows for a fast stochastic search in  $\mathcal{S}^*$ . Our construction leads to a (biased) sampling of  $\mathcal{S}^*$ . Such a sampling will be better than a random one if it is possible to find appropriate computational ways to go from one  $s^*$  to another. Finally, one last advantage of this modified notion of closeness is that it does not require basins of attraction to be defined; the local search can then incorporate memory or be non-deterministic, making the method far more general.

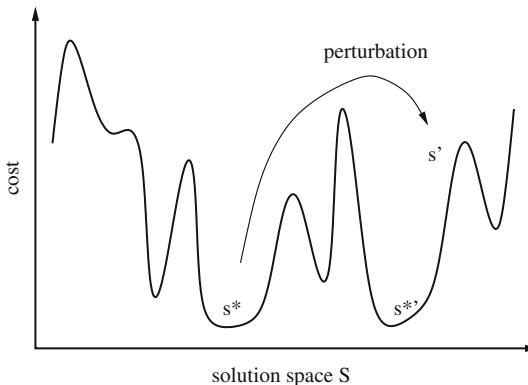
---

<sup>2</sup> Note that the local search finds neighbors stochastically; generally there is no efficient way to ensure that one has tested *all* the neighbors of any given  $s^*$ .

### 12.2.4 Iterated Local Search

We want to explore  $\mathcal{S}^*$  using a walk that steps from one  $s^*$  to a “nearby” one, without the constraint of using only neighbors as defined above. Iterated local search (ILS) achieves this heuristically as follows. Given the current  $s^*$ , we first apply a change or perturbation that leads to an intermediate state  $s'$ , which belongs to  $\mathcal{S}$ . Then LocalSearch is applied to  $s'$  and we reach a solution  $s^{* \prime}$  in  $\mathcal{S}^*$ . If  $s^{* \prime}$  passes an acceptance test, it becomes the next element of the walk in  $\mathcal{S}^*$ ; otherwise, we return to  $s^*$ . The resulting walk is a case of a stochastic search in  $\mathcal{S}^*$ , but where neighborhoods are never explicitly introduced. This iterated local search procedure should lead to good biased sampling as long as the perturbations are neither too small nor too large. If they are too small, one will often fall back to  $s^*$  and few new solutions of  $\mathcal{S}^*$  will be explored. If on the contrary the perturbations are too large,  $s'$  will be random, there will be no bias in the sampling, and we will recover a random restart-type algorithm.

The overall ILS procedure is pictorially illustrated in Figure 12.2. To be complete, let us note that generally the iterated local search walk will not be reversible; in particular one may sometimes be able to step from  $s_1^*$  to  $s_2^*$  but not from  $s_2^*$  to  $s_1^*$ . However, this “unfortunate” aspect of the procedure does not prevent ILS from being very effective in practice.



**Fig. 12.2** Pictorial representation of iterated local search. Starting with a local minimum  $s^*$ , we apply a perturbation leading to a solution  $s'$ . After applying LocalSearch, we find a new local minimum  $s^{* \prime}$  that may be better than  $s^*$ .

Since deterministic perturbations may lead to short cycles (for instance of length two), one should randomize the perturbations or make them adaptive to avoid this kind of cycling. If the perturbations depend on any of the previous  $s^*$ , one has a walk in  $\mathcal{S}^*$  with *memory*. Now the reader may have noticed that aside from the issue of perturbations (which use the structure on  $\mathcal{S}$ ), our formalism reduces the problem to that of a stochastic search on  $\mathcal{S}^*$ . Then all bells and whistles (diversification, intensification, tabu, adaptive perturbations, and acceptance criteria, etc.)

---

**Algorithm 1** Iterated local search

---

```

1:  $s_0 = \text{GenerateInitialSolution}$ 
2:  $s^* = \text{LocalSearch}(s_0)$ 
3: repeat
4:    $s' = \text{Perturbation}(s^*, history)$ 
5:    $s^{*'} = \text{LocalSearch}(s')$ 
6:    $s^* = \text{AcceptanceCriterion}(s^*, s^{*'}, history)$ 
7: until termination condition met

```

---

that are commonly used in that context may be applied here. This leads us to define iterated local search as a metaheuristic having the high-level architecture given by Algorithm 1.

In practice, much of the potential complexity of ILS is hidden in the history dependence. If there happens to be no such dependence, the walk has no memory:<sup>3</sup> the perturbation and acceptance criterion do not depend on any of the solutions visited previously during the walk, and one accepts or not  $s^{*'}$  with a fixed rule. This leads to random walk dynamics on  $\mathcal{S}^*$  that are “Markovian,” i.e., the probability of making a particular step from  $s_1^*$  to  $s_2^*$  depends only on  $s_1^*$  and  $s_2^*$ . Most of the work using ILS has been of this type, though studies show that incorporating memory enhances performance [85].

Staying within Markovian walks, the most basic acceptance criteria will use only the difference in the costs of  $s^*$  and  $s^{*'}$ ; this type of dynamics for the walk is then very similar in spirit to what occurs in simulated annealing. A limiting case of this is to accept only improving moves, as happens in simulated annealing at zero temperature; the algorithm then does stochastic descent in  $\mathcal{S}^*$ . If we add to such a method a termination criterion, the resulting algorithm pretty much has two nested local searches; to be precise, it has a local search operating on  $\mathcal{S}$  embedded in a stochastic search operating on  $\mathcal{S}^*$ . More generally, one can extend this type of algorithm to more levels of nesting, having a different stochastic search algorithm for  $\mathcal{S}^*$ ,  $\mathcal{S}^{**}$ , and so on. Each level would be characterized by its own type of perturbation and stopping rule; to our knowledge, such a construction has never been attempted.

We can summarize this section by saying that the potential power of iterated local search lies in its *biased* sampling of the set of local optima. The efficiency of this sampling depends both on the kinds of perturbations and on the acceptance criteria. Interestingly, even with the most naïve implementations of these components, iterated local search is much better than random restart. But still much better results can be obtained if the iterated local search modules are optimized. First, the acceptance criteria can be adjusted empirically as in simulated annealing without knowing anything about the problem being optimized. This kind of optimization will be familiar to any user of metaheuristics, though the questions of memory may become quite complex. Second, the perturbation can incorporate as much problem-specific information as the developer is willing to put into it. In practice, a rule of thumb can

---

<sup>3</sup> Recall that to simplify this section’s presentation, the local search is assumed to have no memory.

be used as a guide: “a good perturbation transforms one excellent solution into an excellent starting point for a local search.” Together, these different aspects show that iterated local search algorithms can have a wide range of complexity, but complexity may be added progressively and in a modular way. (Recall in particular that all of the fine-tuning that resides in the embedded local search can be ignored if one wants, and it does not appear in the metaheuristic *per se*.) This makes iterated local search an appealing metaheuristic for both academic and industrial applications. The cherry on the cake is speed: as we shall see soon, one can perform  $k$  local searches embedded within an iterated local search *much* faster than if the  $k$  local searches are run with random restart.

## 12.3 Getting High Performance

Given all these advantages, we hope the reader is now motivated to go on and consider the more nitty-gritty details that arise when developing an ILS algorithm for a new application. In this section, we will illustrate the main issues that need to be tackled when optimizing an ILS algorithm in order to achieve high performance.

There are four components to consider: `GenerateInitialSolution`, `LocalSearch`, `Perturbation`, and `AcceptanceCriterion`. Before attempting to develop a state-of-the-art algorithm, it is relatively straightforward to develop a more basic version of ILS. Indeed, (i) one can start with a random solution or one returned by some greedy construction heuristic; (ii) for most problems a local search algorithm is readily available; (iii) for the perturbation, a random move in a neighborhood of higher order than the one used by the local search algorithm can be surprisingly effective; and (iv) a reasonable first guess for the acceptance criterion is to force the cost to decrease, corresponding to a stochastic first-improvement algorithm in  $\mathcal{S}^*$ . Basic ILS implementations of this type usually lead to much better performance than random restart approaches. The developer can then run this basic ILS to build his intuition and try to improve the overall algorithm performance by improving each of the four modules. This should be particularly effective if it is possible to take into account the specificities of the combinatorial optimization problem under consideration. In practice, this tuning is easier for ILS than for other, less modular metaheuristics. The reason may be that the complexity of ILS is reduced by its modularity, the function of each component being relatively easy to understand. Finally, the last task to consider is the overall optimization of the ILS algorithm; indeed, the different components affect one another and so it is necessary to understand their interactions. However, because these interactions are so problem dependent, we wait till the end of this section before discussing that kind of “global” optimization.

Perhaps the main message here is that the developer can choose the level of optimization he wants. In the absence of any optimizations, ILS is a simple, easy to implement, and quite effective metaheuristic. But with further work on its four components, ILS can often be turned into a very competitive or even state-of-the-art algorithm.

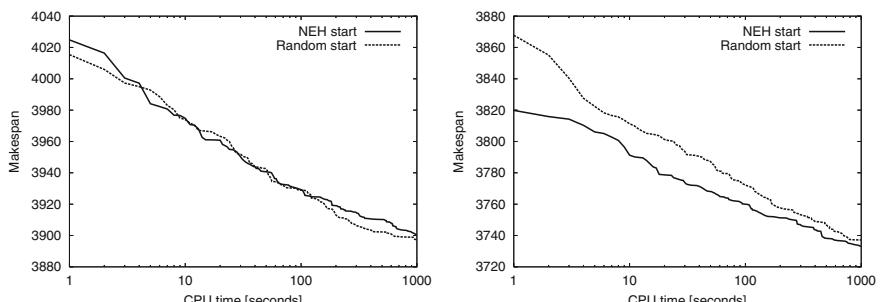
### 12.3.1 Initial Solution

Local search applied to the initial solution  $s_0$  gives the starting point  $s_0^*$  of the walk in  $\mathcal{S}^*$ . Starting with a good  $s_0^*$  can be important if high-quality solutions are to be reached *as fast as possible*.

Standard choices for  $s_0$  are either a random initial solution or a solution returned by a greedy construction heuristic. A greedy initial solution  $s_0$  has two main advantages over random starting solutions: (i) when combined with local search, greedy initial solutions often result in better quality solutions  $s_0^*$ ; (ii) a local search from greedy solutions takes, on average, less improvement steps and therefore the local search requires less CPU time.<sup>4</sup>

The question of an appropriate initial solution for (random restart) local search carries over to ILS because of the dependence of the walk in  $\mathcal{S}^*$  on  $s_0^*$ . Indeed, when starting with a random  $s_0$ , ILS may take several iterations to catch up in quality with runs using an  $s_0^*$  obtained by a greedy initial solution. Hence, for short computation times the initial solution is certainly important to achieve the highest possible solution quality. For larger computation times, the dependence on  $s_0$  of the final solution returned by ILS reflects just how fast, if at all, the memory of the initial solution is lost when performing the walk in  $\mathcal{S}^*$ .

Let us illustrate the trade-offs between random and greedy initial solutions when using an ILS algorithm for the permutation flow shop problem (PFSP) [82]. That ILS algorithm uses a straightforward local search implementation and random perturbations and always applies the perturbation to the best solution found so far. In Figure 12.3 we show how the average solution cost (makespan) evolves with the



**Fig. 12.3** The plots show the average solution cost (makespan on the y-axis) as a function of CPU time (given on the x-axis) for an ILS algorithm applied to the PFSP on instances ta051 and ta056.

<sup>4</sup> Note that the best possible greedy initial solution need not be the best choice when combined with a local search. For example, in [47], it is shown that the combination of the Clarke–Wright starting tour (one of the best performing construction heuristics for the travelling salesman problem) with local search resulted in worse local optima than starting from random initial solutions when using 3-opt. Additionally, greedy algorithms that generate very high-quality initial solutions can be quite time consuming.

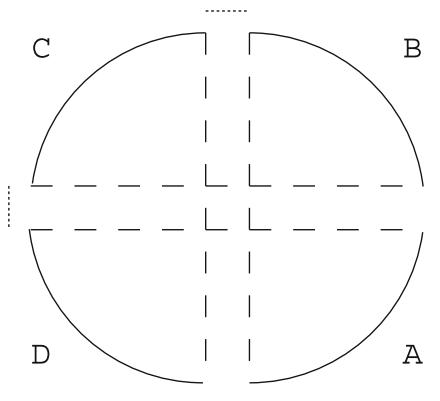
number of iterations for two instances. The averages are for 10 independent runs when starting from random initial solutions or from initial solutions returned by the NEH heuristic [71]. (NEH is one of the best performing constructive heuristics for the PFSP.) For short runs, the curve for the instance on the right shows that the NEH initial solutions lead to better average solution cost than random initial solutions. But, for longer times, the picture is not so clear. Sometimes, random initial solutions lead to better average results as observed on the instance on the left. This kind of test was also performed for ILS applied to the travelling salesman problem (TSP) [1]. Again it was observed that the initial solution had a significant influence on quality for short-to-medium sized runs.

In general, there will not always be a clear-cut answer regarding the best choice of an initial solution, but greedy initial solutions appear to be recommendable when one needs low-cost solutions quickly. For much longer runs, the initial solution seems to be less relevant, so the user can choose the initial solution, which is the easiest to implement. If, however, one has an application where the influence of the initial solution does persist for long times, the ILS walk is probably having difficulty in exploring  $\mathcal{S}^*$  and so other perturbations or acceptance criteria should be considered.

### 12.3.2 Perturbation

The main drawback of iterative improvement is that it gets trapped in local optima that are significantly worse than the global optimum. Much like simulated annealing, ILS escapes from local optima by applying perturbations to the current local minimum. We will refer to the *strength* of a perturbation as the number of solution components that are modified. For instance for the TSP, it is the number of edges that are modified in the tour, while in the flow shop problem, it is the number of jobs that are moved by the perturbation. Generally, the local search should not be able to undo the perturbation, otherwise one will fall back into the local optimum just visited. Surprisingly, a *random* move in a neighborhood of higher order than the one used by the local search algorithm can often achieve this and will lead to a satisfactory algorithm. Still better results can be obtained if the perturbations take into account properties of the problem and are well matched to the local search algorithm.

By how much should the perturbation change the current solution? If the perturbation is too strong, ILS may behave like a random restart, so better solutions will only be found with a very low probability. On the other hand, if the perturbation is too small, the local search will often fall back into the local optimum just visited and the diversification of the search will be very limited. An example of a simple but effective perturbation for the TSP is the *double-bridge move*. This perturbation cuts four edges (and is thus of “strength” four) and introduces four new ones as shown in Figure 12.4. Notice that each bridge is a two-change, but neither of the two-changes individually keeps the tour connected. Nearly all ILS studies of the TSP have incorporated this kind of perturbation, and it has been found to be effective for all instance



**Fig. 12.4** Schematic representation of the double-bridge move. The four *dotted edges* are removed and the remaining parts A, B, C, D are reconnected by the *dashed edges*.

sizes. This is almost certainly because it changes the topology of the tour and can operate on quadruples of very distant cities, whereas local search always modifies the tour among nearby cities. In effect, the double-bridge perturbation cannot be undone easily, neither by simple local search algorithms such as 2-opt or 3-opt nor by most local search algorithms based on the Lin–Kernighan heuristic [55], which is currently the champion local search algorithm for the TSP. (Only very few local searches include such double-bridge changes in the search, the best known being the Lin–Kernighan implementation of Helsgaun [40].) Furthermore, this perturbation does not increase much the tour length, so even if the current solution is very good, one is almost sure the next one will be good, too. These two properties of the perturbation—its small strength and its fundamentally different nature from the changes used in local search—make the TSP the perfect application for ILS.

We will now consider optimizing the perturbation assuming the other modules to be fixed. In problems like the TSP, one can hope to have a satisfactory ILS when using perturbations of fixed size (independent of the instance size). On the contrary, for more difficult problems, fixed-strength perturbations may lead to poor performance. Of course, the strength of the perturbations used is not the whole story; their nature is almost always very important and will also be discussed. Finally we will close by pointing out that the perturbation strength has an effect on the speed of the local search: weak perturbations usually lead to faster execution of LocalSearch. All these different aspects need to be considered when optimizing this module.

### 12.3.2.1 Perturbation Strength

For some problems, an appropriate perturbation strength is very small and seems to be rather independent of the instance size. This is the case for both the TSP and the PFSP, and, interestingly, ILS for these problems is very competitive with today's best metaheuristic methods. We can also consider other problems where one is driven instead to large perturbation sizes. Consider the example of an ILS

algorithm for the quadratic assignment problem (QAP). We use an embedded 2-opt local search algorithm, the perturbation is a random exchange of the location of  $k$  items, where  $k$  is an adjustable parameter, and the perturbation always modifies the best solution found so far. We applied this ILS algorithm to QAPLIB instances<sup>5</sup> from four different classes of QAP instances [86]; computational results are given in Table 12.1. A first observation is that the best perturbation size is strongly dependent on the particular instance. For two of the instances, the best performance was achieved when as many as 75% of the solution components were altered by the perturbation. Additionally, when the perturbation strength is too small, the ILS performed worse than random restart (corresponding to the perturbation strength  $n$ ). However, the fact that random restart for the QAP may perform—on average—better than a basic ILS algorithm is a bit misleading: in the next section we will show that by modifying the acceptance criterion, ILS becomes far better than random restart. Thus, one should keep in mind that the optimization of an ILS algorithm may require more than the optimization of the individual components.

**Table 12.1** The first column gives the identifier of the QAP instance; the number in the identifier gives its size  $n$ . The successive columns are for perturbation sizes  $3, n/12, \dots, n$ . A perturbation of size  $n$  corresponds to random restart. The table shows the average solution cost measured across 10 independent runs for each instance. The CPU time for each trial is 30 s. for kra30a, 60 s. for tai60a and sko64, and 120 s. for tai60b on a Pentium III 500 MHz PC.

Instance	3	$n/12$	$n/6$	$n/4$	$n/3$	$n/2$	$3n/4$	$n$
kra30a	2.51	2.51	2.04	1.06	0.83	0.42	0.0	0.77
sko64	0.65	1.04	0.50	0.37	0.29	0.29	0.82	0.93
tai60a	2.31	2.24	1.91	1.71	1.86	2.94	3.13	3.18
tai60b	2.44	0.97	0.67	0.96	0.82	0.50	0.14	0.43

### 12.3.2.2 Adaptive Perturbations

The behavior of ILS for the QAP and also for other combinatorial optimization problems [41, 82] shows that there is no a priori single best size for the perturbation. This observation motivates the possibility of modifying the perturbation strength and adapting it *during* the run.

To this end, one approach is to exploit the search history. For the development of such schemes, inspiration can be taken from what is done in the context of reactive search [5, 6]. In particular, Battiti and Protasi proposed a reactive search algorithm for MAX-SAT, which fits perfectly into the ILS framework [5]. They perform a perturbation scheme which is implemented by a tabu search algorithm and after each perturbation they apply a standard local improvement algorithm.

Another way of adapting the perturbation is to change its strength during the search according to an a priori defined scheme. One particular example is employed

---

<sup>5</sup> QAPLIB is accessible at <http://www.seas.upenn.edu/qaplib>.

in *basic variable neighborhood search* (basic VNS) [38, 68]; we refer to Section 12.5 for some explanations on VNS. Other examples arise in the context of tabu search [36]. In particular, ideas such as strategic oscillations may be useful to derive more effective perturbations.

### 12.3.2.3 More Complex Perturbation Schemes

Perturbations can be more complex than random changes in a higher order neighborhood. One rather general procedure to generate  $s'$  from the current  $s^*$  is as follows. First, gently modify the definition of the instance, e.g., via the parameters defining the various costs. Second, for this modified instance, run LocalSearch using  $s^*$  as input; the output is the perturbed solution  $s'$ . Interestingly, this is the method proposed in the oldest ILS work we are aware of: Baxter tested this approach with success on a location problem [9]. This idea seems to have been rediscovered later by Codenotti et al. in the context of the TSP [18]. They first change slightly the city coordinates. Then they apply the local search to  $s^*$  using the perturbed city locations, obtaining the new tour  $s'$ . Finally, running LocalSearch on  $s'$  using the *unperturbed* city coordinates, they obtain the new candidate tour  $s^{* \prime}$ .

Other sophisticated ways to generate good perturbations consist in optimizing a sub-part of the problem. Such an approach was proposed by Lourenço [56] in the context of the job shop scheduling problem (JSP). Her perturbation schemes are based on defining one- or two-machine sub-problems by fixing a number of variables in the current solution and solving these sub-problems, either heuristically [57] or to optimality using for instance Carlier's exact algorithm [15] or the early-late algorithm [57]. These schemes work well because (i) local search is unable to undo the perturbations; (ii) after the perturbation, the solutions tend to be very good and also have “new” parts that are optimized. More recently, evolutionary algorithms have been used to generate perturbations for ILS algorithms [59]. The idea in this approach is to generate a small initial population of solutions by perturbing the best-so-far solution, to perform a short run of a GA with this population and then to use the best solution found in this process as a new starting solution for the local search.

### 12.3.2.4 Speed

In the context of “easy” problems where ILS can work very well with weak (fixed size) perturbations, there is another reason why that metaheuristic can perform much better than random restart: *Speed*. Indeed, LocalSearch will usually execute much faster on a solution obtained by applying a small perturbation to a local optimum than on a random solution. As a consequence, iterated local search can run many more local searches than random restart for the same CPU time. As a qualitative example, consider again Euclidean TSPs.  $\mathcal{O}(n)$  local changes have to be applied by the local search to reach a local optimum from a random starting solution, whereas empirically a nearly constant number is necessary in ILS when using the  $s'$  obtained with the double-bridge perturbation. Hence, in a given amount of CPU time, ILS

can sample many more local optima than random restart can. This *speed factor* can give ILS a considerable advantage over other restart schemes.

Let us illustrate this speed factor quantitatively. We compared the number of local searches performed in a given amount of CPU time for the TSP by (i) random restart; (ii) ILS using a double-bridge move; (iii) ILS using five simultaneous double-bridge moves. (For both ILS implementations, we used random starting solutions and the routine AcceptanceCriterion accepted only shorter tours.) For our numerical tests we used a 3-opt implementation with standard speed-up techniques. In particular, it used a fixed radius nearest-neighbor search restricted to candidate lists with the 40 nearest neighbors of each city and “don’t look” bits [11, 47, 62]. Initially, all don’t look bits were turned off (set to 0). If no improving move was found for a given node, its don’t look bit was turned on (set to 1) and the node was not considered as a starting node for finding an improving move in the next iteration. When an arc incident to a node was changed by a move, the node’s don’t look bit was turned off again. In addition, after a perturbation we only turned off the don’t look bits of the 25 cities around each of the four breakpoints in the current tour. All three algorithms were run for 120 s on a 266 MHz Pentium II processor on a set of TSPLIB<sup>6</sup> instances ranging from 100 to 5915 cities. Results are given in Table 12.2. For the smallest instances, we see that iterated local search ran between 2 and 10 times as many local searches as random restart. This advantage of ILS grows fast with increasing instance size: for the largest instance, the first ILS algorithm ran approximately 260 times as many local searches as random restart in our allotted time. Obviously, this speed advantage of ILS over random restart is strongly dependent on the strength of the applied perturbation. The larger the perturbation size, the more the solution

**Table 12.2** The first column gives the identifier of the TSP instance, where the number in the identifier specifies the number of cities. The next columns give the number of local searches performed when using (i) random restart (#LS<sub>RR</sub>); (ii) ILS with a single double-bridge perturbation (#LS<sub>1-DB</sub>); (iii) ILS with a five double-bridge perturbation (#LS<sub>5-DB</sub>). All algorithms were run for 120 s on a PC with a 266 MHz Pentium processor.

Instance	#LS <sub>RR</sub>	#LS <sub>1-DB</sub>	#LS <sub>5-DB</sub>
kroA100	17,507	56,186	34,451
d198	7715	36,849	16,454
lin318	4271	25,540	9430
pcb442	4394	40,509	12,880
rat783	1340	21,937	4631
pr1002	910	17,894	3345
pcb1173	712	18,999	3229
d1291	835	23,842	4312
f11577	742	22,438	3915
pr2392	216	15,324	1777
pcb3038	121	13,323	1232
f13795	134	14,478	1773
r15915	34	8820	556

<sup>6</sup> TSPLIB is accessible at [www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95](http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95).

is modified and generally the longer the subsequent local search takes. This fact is intuitively obvious and it is confirmed in Table 12.2.

In summary, the optimization of the perturbation depends on many factors, and problem-specific characteristics play a central role. Finally, it is important to keep in mind that the perturbation also interacts with the other components of ILS. We will discuss these interactions in Section 12.3.5.

### 12.3.3 Acceptance Criterion

ILS does a randomized walk in  $\mathcal{S}^*$ , the space of local minima. The perturbation mechanism together with the local search defines the possible transitions between a current solution  $s^*$  in  $\mathcal{S}^*$  and a “neighboring” solution  $s^{* \prime}$  also in  $\mathcal{S}^*$ . The procedure `AcceptanceCriterion` then determines whether  $s^{* \prime}$  is accepted or not as the new current solution. `AcceptanceCriterion` has a strong influence on the nature and effectiveness of the walk in  $\mathcal{S}^*$ . Roughly, it can be used to control the balance between intensification and diversification of that search. A simple way to illustrate this is to consider a Markovian acceptance criterion. A very strong intensification is achieved if only better solutions are accepted. We call this acceptance criterion `Better` and it is defined for minimization problems as

$$\text{Better}(s^*, s^{* \prime}, \text{history}) = \begin{cases} s^{* \prime} & \text{if } \mathcal{C}(s^{* \prime}) < \mathcal{C}(s^*) \\ s^* & \text{otherwise.} \end{cases} \quad (12.1)$$

At the opposite extreme is the random walk acceptance criterion (denoted by `RW`) which always applies the perturbation to the most recently visited local optimum, irrespective of its cost:

$$\text{RW}(s^*, s^{* \prime}, \text{history}) = s^{* \prime}. \quad (12.2)$$

This criterion clearly favors diversification over intensification.

Many intermediate choices between these two extreme cases are possible. In one of the first ILS algorithms, the large-step Markov chain algorithm proposed by Martin et al. [62, 63], a simulated annealing type acceptance criterion was applied. We call it `LSCM`( $s^*, s^{* \prime}, \text{history}$ ). In particular,  $s^{* \prime}$  is always accepted if it is better than  $s^*$ . Otherwise, if  $s^{* \prime}$  is worse than  $s^*$ ,  $s^{* \prime}$  is accepted with probability  $\exp\{(\mathcal{C}(s^*) - \mathcal{C}(s^{* \prime}))/T\}$  where  $T$  is a parameter called temperature, which is usually lowered during the run as in simulated annealing. Note that `LSCM` approaches the `RW` acceptance criterion if  $T$  is very high, while at very low temperatures `LSCM` is similar to the `Better` acceptance criterion. An interesting possibility for `LSCM` is to allow non-monotonic temperature schedules as proposed for simulated annealing [43] or tabu thresholding [34]. This can be most effective if it is done using memory: when further intensification no longer seems useful, increase the temperature to do diversification for a limited time, then resume intensification. Of course, just as in tabu search, it is desirable to do this in an automated and self-regulating manner [36].

A very limited usage of memory in the acceptance criterion is to restart the ILS algorithm when the intensification seems to become ineffective. (Of course, this is a rather extreme way to switch from intensification to diversification). For instance one can restart the ILS algorithm from a new initial solution if no improved solution has been found for a given number of iterations. The restart of the algorithm can easily be modeled by the acceptance criterion  $\text{Restart}(s^*, s^{*'}, \text{history})$ . Let  $i_{\text{last}}$  be the last iteration where a better solution has been found and  $i$  be the iteration counter. Then  $\text{Restart}(s^*, s^{*'}, \text{history})$  is defined as

$$\text{Restart}(s^*, s^{*'}, \text{history}) = \begin{cases} s^{*'} & \text{if } \mathcal{C}(s^{*'}) < \mathcal{C}(s^*) \\ s & \text{if } \mathcal{C}(s^{*'}) \geq \mathcal{C}(s^*) \text{ and } i - i_{\text{last}} > i_r \\ s^* & \text{otherwise.} \end{cases} \quad (12.3)$$

where  $i_r$  is a parameter that indicates that the algorithm should be restarted if no improved solution was found for  $i_r$  iterations;  $s$  can be generated in different ways. The simplest strategy is to generate a new solution randomly or by a greedy randomized heuristic. Clearly many other ways to incorporate memory may and should be considered, the overall efficiency of ILS being quite sensitive to the acceptance criterion applied. We now illustrate this with two examples.

### 12.3.3.1 Example 1: TSP

Let us consider the effect of the two acceptance criteria `RW` and `Better`. We performed our tests on the TSP as summarized in Table 12.3. We give the average percentage over the known optimal solutions when using 10 independent runs on

**Table 12.3** Influence of the acceptance criterion for various TSP instances. The first column gives the identifier of the TSP instance, where the number in the identifier specifies the number of cities. The next columns give the average percentage over the optimal tour length obtained using: random restart (RR), iterated local search with `RW`, and iterated local search with `Better`. The results are averaged over 10 independent runs. All algorithms were run for 120 s on a PC with a 266 MHz Pentium processor.

Instance	$\Delta_{\text{avg}}(\text{RR})$	$\Delta_{\text{avg}}(\text{RW})$	$\Delta_{\text{avg}}(\text{Better})$
kroA100	0.0	0.0	0.0
d198	0.003	0.0	0.0
lin318	0.66	0.30	0.12
pcb442	0.83	0.42	0.11
rat783	2.46	1.37	0.12
pr1002	2.72	1.55	0.14
pcb1173	3.12	1.63	0.40
d1291	2.21	0.59	0.28
f11577	10.3	1.20	0.33
pr2392	4.38	2.29	0.54
pcb3038	4.21	2.62	0.47
f13795	38.8	1.87	0.58
r15915	6.90	2.13	0.66

our set of benchmark instances. In addition, we also give this number for the random restart 3-opt algorithm. First, we observe that both ILS algorithms lead to a significantly better average solution quality than random restart using the same local search. This is particularly true for the largest instances, confirming the claims made in Section 12.2. Second, given that one expects good solutions for the TSP to cluster (see Section 12.3.5), a good strategy should incorporate intensification. It is thus not surprising to see that the Better criterion leads to shorter tours than the RW criterion.

The runs given in this example are rather short. For much longer runs, the Better strategy comes to a point where it no longer finds improved tours. In fact, an analysis of ILS algorithms based on the run-time distribution methodology [42] has shown that such stagnation situations effectively occur and that the performance of the ILS algorithm can be considerably improved by additional diversification mechanisms [84], an occasional restart of the ILS algorithm being the conceptually simplest case.

### 12.3.3.2 Example 2: QAP

Let us come back to ILS for the QAP. For this problem we found that the acceptance criterion Better together with a poor choice of the perturbation strength could result in worse performance than random restart. In Table 12.4 we give results for the same ILS algorithm except that we now also consider the use of the RW and Restart acceptance criteria. We see that the performance of the ILS algorithms using these acceptance criteria is much better than random restart, the only exception being for the ILS algorithm with RW for a small perturbation strength on tai60b.

This example shows that there are strong interdependencies between the perturbation strength and the acceptance criterion. This dependency is rarely completely

**Table 12.4** Further tests on the QAP benchmark instances using the same perturbations and CPU times than for Table 12.1; given is the average solution cost measured across 10 independent runs for each instance. Here we consider three different choices for the acceptance criterion. Clearly, the inclusion of diversification significantly lowers the average cost found.

Instance	Acceptance	3	$n/12$	$n/6$	$n/4$	$n/3$	$n/2$	$3n/4$	$n$
kra30a	Better	2.51	2.51	2.04	1.06	0.83	0.42	0.0	0.77
kra30a	RW	0.0	0.0	0.0	0.0	0.0	0.02	0.47	0.77
kra30a	Restart	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.77
sko64	Better	0.65	1.04	0.50	0.37	0.29	0.29	0.82	0.93
sko64	RW	0.11	0.14	0.17	0.24	0.44	0.62	0.88	0.93
sko64	Restart	0.37	0.31	0.14	0.14	0.15	0.41	0.79	0.93
tai60a	Better	2.31	2.24	1.91	1.71	1.86	2.94	3.13	3.18
tai60a	RW	1.36	1.44	2.08	2.63	2.81	3.02	3.14	3.18
tai60a	Restart	1.83	1.74	1.45	1.73	2.29	3.01	3.10	3.18
tai60b	Better	2.44	0.97	0.67	0.96	0.82	0.50	0.14	0.43
tai60b	RW	0.79	0.80	0.52	0.21	0.08	0.14	0.28	0.43
tai60b	Restart	0.08	0.08	0.005	0.02	0.03	0.07	0.17	0.43

understood. But, as a general rule of thumb, when it is necessary to allow for diversification, we believe it is best to do so by accepting numerous small perturbations rather than by accepting one large perturbation.

Most of the acceptance criteria applied so far in ILS algorithms either are fully Markovian or make use of the search history in a very limited way. We expect that there will be many more ILS applications in the future making strong use of the search history; in particular, alternating between intensification and diversification is likely to be an essential feature in these applications.

### 12.3.4 Local Search

So far we have treated the local search algorithm as a black box, which is called many times by ILS. Since the behavior and performance of the overall ILS algorithm is quite sensitive to the choice of the embedded heuristic, one should optimize this choice whenever possible. In practice, there may be many quite different algorithms that can be used for the embedded heuristic. (As mentioned at the beginning of the chapter, the heuristic needs not even be a local search.) One might think that the better the local search, the better the corresponding ILS. Often this is true. For instance in the context of the TSP, Lin–Kernighan [55] is a better local search than 3-opt, which itself is better than 2-opt [47]. Using a fixed type of perturbation such as the double-bridge move, one finds that iterated Lin–Kernighan gives better solutions than iterated 3-opt which itself gives better solutions than iterated 2-opt [47, 84]. But if we assume that the total computation time is fixed, it might be better to apply more frequently a faster but less effective local search algorithm than a slower and more powerful one. Clearly, which choice is best depends on just how much more time is needed to run the better heuristic. If the speed difference is not large, for instance, if it is independent of the instance size, then it usually worth using the better heuristic. This is the most frequent case; in the TSP, for instance, 3-opt is a bit slower than 2-opt, but the improvement in quality of the tours is well worth the extra CPU time, be it using random restart or iterated local search. The same comparison applies to using Lin–Kernighan rather than 3-opt. However, there are other cases where the increase in CPU time is so large compared to the improvement in solution quality that it is best not to use the “better” local search. For example, again in the context of the TSP, it is known that 4-opt gives slightly better solutions than 3-opt, but in standard implementations it is  $O(n)$  times slower ( $n$  being the number of cities). It is then better not to use 4-opt as the local search embedded in ILS.

There are also other aspects that should be considered when selecting a local search. Clearly, there is not much point in having an excellent local search if it will systematically undo the perturbation; however, this issue is one of globally optimizing iterated local search, so it will be postponed till the next subsection. Another important aspect is whether one can really get the speed-ups that were mentioned in subsection 12.3.2. There we saw that a standard speed-up for local search was to introduce don’t look bits. These give a large gain in speed if the bits can also be reset after the application of the perturbation. This requires that the

developer is able to access the source code of LocalSearch. A state-of-the art ILS will take advantage of all possible speed-up tricks, and thus the LocalSearch most likely will not be a true black box.

Finally, there may be some advantages in allowing LocalSearch to sometimes generate worse solutions. For instance, if we replace the local search heuristic by tabu search or short simulated annealing runs, the corresponding ILS may perform better. This seems most promising when standard iterative improvement algorithms perform poorly. This is indeed the case in the job shop scheduling problem: the use of tabu search as the embedded heuristic gives rise to a very effective iterated local search [58].

### 12.3.5 Global Optimization of ILS

So far, we have considered representative issues arising when optimizing separately each of the four components of an iterated local search. In particular, when illustrating various important characteristics of one component, we kept the other components fixed. But clearly the optimization of one component depends on the choices made for the others; as an example, we made it clear that a good perturbation must have the property that it cannot be easily undone by the local search. Thus, at least in principle, one should tackle the *global* optimization of an ILS. Since at present there is no theory for analyzing a metaheuristic such as iterated local search, we will just give a rough idea of how such a global optimization can be approached in practice.

If we reconsider the subsection on the effect of the initial solution, we see that GenerateInitialSolution is to a large extent irrelevant when the ILS performs well and rapidly loses the memory of its starting point. Hereafter we assume that this is the case; then the optimization of GenerateInitialSolution can be ignored and we are left with the joint optimization of the three other components. Clearly the best choice of Perturbation depends on the choice of LocalSearch while the best choice of AcceptanceCriterion depends on the choices of LocalSearch and Perturbation. In practice, we can approximate this global optimization problem by successively optimizing each component, assuming the others are fixed until no improvements are found for any of the components [26]. Thus the only difference with what has been presented in the previous subsections is that the optimization has to be iterative. This does not guarantee global optimization of the ILS, but it should lead to an adequate optimization of the overall algorithm.

Given these approximations, we should be more precise about what we want to optimize. For most users, it will be the mean (over starting solutions) of the best cost found during a run of a given length. Then the “best” choice for the different components is a well-defined problem, though it is intractable without further restrictions. Furthermore, in general, the detailed instance that will be considered by the user is not known ahead of time, so it is important that the resulting ILS algorithm be robust. Thus it is preferable not to optimize it to the point where it is sensitive to the details of the instance. This robustness seems to be achieved in practice:

researchers implement versions of iterated local search with a reasonable level of global optimization and then test with some degree of success the performance on standard benchmarks.

At the risk of repeating ourselves, let us highlight the main dependencies of the components:

1. The perturbation should not be easily undone by the local search; if the local search has obvious short-comings, a good perturbation should compensate for them.
2. The combination Perturbation–AcceptanceCriterion determines the relative balance of intensification and diversification; large perturbations are only useful if they can be accepted, which occurs only if the acceptance criterion is not too biased toward better solutions.

As a general guideline, LocalSearch should be as powerful as possible as long as it is not too costly in CPU time. Given such a choice, find a well-adapted perturbation following the discussion in Section 12.3.2; to the extent possible, taking advantage of the structure of the problem. Finally, set the AcceptanceCriterion routine so that  $\mathcal{S}^*$  is sampled adequately. With this point of view, the overall optimization of the ILS is nearly a bottom-up process, but with iteration. Perhaps the core issue is what to put into Perturbation. In particular, is it possible to consider only weak perturbations? From a theoretical point of view, the answer to this question depends on whether the best solutions “cluster” in  $\mathcal{S}^*$ . In some problems (and the TSP is one of them), there is a strong correlation between the cost of a solution and its “distance” to the optimum: in effect, the best solutions cluster together, i.e., have many similar components. This has been referred to in many different ways: “Massif Central” phenomenon [31], principle of proximate optimality [36], and replica symmetry [66]. If the problem under consideration has this property, it is not unreasonable to hope to find the true optimum using a biased sampling of  $\mathcal{S}^*$ . In particular, it is clear that it is useful to use intensification to improve the probability of hitting the global optimum.

There are, however, other types of problems where the clustering is incomplete, i.e., where very distant solutions can be nearly as good as the optimum. Examples of combinatorial optimization problems in this category are QAP, graph bi-section, and MAX-SAT. When the space of solutions has this property, new strategies have to be used. Clearly, it is still necessary to use intensification to get the best solution in one’s current neighborhood, but generally this will not lead to the optimum. After an intensification phase, one must explore other regions of  $\mathcal{S}^*$ . This can be attempted by using “large” perturbations whose strength grows with the instance. Other possibilities are to restart the algorithm from scratch and repeat another intensification phase or by oscillating the acceptance criterion between intensification and diversification phases. Additional ideas on the trade-offs between intensification and diversification are well discussed in the context of tabu search (see, for example, [36]). Clearly, finding an appropriate balance of intensification vs. diversification is very important but still a challenging problem.

## 12.4 Selected Applications of ILS

ILS algorithms have been applied successfully to a variety of combinatorial optimization problems. In some cases, these algorithms achieve extremely high performance and even constitute the current state-of-the-art algorithms, while in other cases the ILS approach is merely competitive with other metaheuristics. In this section, we give an overview of interesting ILS applications, presenting the core ideas of these algorithms to illustrate possible uses of ILS. We put a particular emphasis on the TSP, given its central role in the development of ILS algorithms.

### 12.4.1 ILS for the TSP

The TSP is probably the best-known combinatorial optimization problem. *De facto*, it is a standard test-bed for the development of new algorithmic ideas: a good performance on the TSP is taken as evidence of the value of such ideas. Like many other metaheuristic algorithms, some of the first ILS algorithms were introduced and tested on the TSP, the oldest case of this being due to Baum [7, 8]. He coined his method *iterated descent*; his tests used 2-opt as the embedded heuristic, random 3-changes as the perturbations, and imposed the tour length to decrease (thus the name of the method). His results were not impressive, in part because some algorithm components were probably not the most appropriate and also because he tackled non-Euclidean TSPs.

A major improvement in the performance of ILS algorithms came from the *large-step Markov chain* (LSMC) algorithm proposed by Martin et al. [62]. They used a simulated annealing-like acceptance criterion (LSMC) from which the algorithm's name is derived. They considered both the application of 3-opt local search and the Lin–Kernighan heuristic, which is the best performing local search algorithm for the TSP. But probably the key ingredient of their work is the introduction of the double-bridge move for the perturbation. This choice made the approach very powerful for the Euclidean TSP and encouraged much more work along these lines. In particular, Johnson [46, 47] coined the term “iterated Lin–Kernighan” (ILK) for his implementation of ILS using Lin–Kernighan as the local search. The main differences with the LSMC implementation are (i) double-bridge moves are random rather than biased; (ii) the costs are improving (only better tours are accepted, corresponding to the choice `Better` in our notation). Since these initial studies, other ILS variants have been proposed; Johnson and McGeoch [47] give a summary of the situation as of 1997 and several additional ILS variants are covered in a 2002 book chapter, which summarizes early results from the 8th DIMACS implementation challenge on the TSP [48].

A high-performing ILS algorithm is offered as part of the Concorde software package and it is available for download at <http://www.tsp.gatech.edu/concorde/>. This chained Lin–Kernighan code has been developed by Applegate, Bixby, Chvatal, and Cook and a detailed description of the code is given in their recent book on the TSP [2]; this book also contains details on an extensive computational study of this

code. Noteworthy is also the experimental study by Applegate et al. [1] who performed tests on very large TSP instances with up to 25 million cities. Recently, a new ILS variant has been proposed that further illustrates the impressive performance of ILS algorithms on very large TSP instances. Currently, the iterated Lin-Kernighan variant of Merz and Huhse [65] appears to be the best performing algorithm for very large TSP instances with several millions of cities when the computation times are relatively short (in the range of a few hours on a modern PC as of 2008).

A major leap in TSP solving stems from Helsgaun's Lin–Kernighan implementation and its iterated version [40]. The main novelty of Helsgaun's algorithm lies on the local search side: the Lin–Kernighan variant developed is based on more complex basic moves than previous implementations. His iterated version of the Lin–Kernighan heuristic is not really an ILS algorithm like the ones presented in this chapter since the generation of new starting solutions is through a solution construction method. However, the constructive mechanism is very strongly biased toward the incumbent solution, which makes this approach somehow similar to an ILS algorithm. The most recent version of this algorithm, along with an accompanying technical report describing the recent developments, is available for download at <http://www.akira.ruc.dk/~keld/research/LKH/>.

There are a number of other ILS algorithms for the TSP that not necessarily offer the ultimate state-of-the-art performance but that illustrate various ideas that may be useful in ILS algorithms. One algorithm, which has already been mentioned before, is the one by Codenotti et al. [18]. It gives an example of a complex perturbation scheme, which is based on the modification of the instance data. Various perturbation sizes as well as population-based extensions of ILS algorithms for the TSP have been studied by Hong et al. [41]. The perturbation mechanism is also the focus of the work by Katayama and Narisha [49]. They introduce a new perturbation mechanism, which they called *genetic transformation*. The genetic transformation mechanism uses two tours, the best found so far,  $s_{\text{best}}^*$ , and a second, current local optimum,  $s^*$ . First a random 4-opt move is performed on  $s_{\text{best}}^*$ , resulting in  $s^{* \prime}$ . Then the subtours that are shared among  $s^{* \prime}$  and  $s^*$  are kept and the resulting parts are reconnected with a greedy algorithm. Computational experiments with an iterated Lin–Kernighan algorithm using the genetic transformation method instead of the standard double-bridge move have shown that the approach is effective.

An analysis of the run-time behavior of various ILS algorithms for the TSP is done by Stützle and Hoos [84, 85]; this analysis clearly shows that ILS algorithms with the Better acceptance criterion show a type of stagnation behavior for long run-times. To avoid such stagnation, restarts and a particular acceptance criterion to diversify the search were proposed. The goal of this latter strategy is to force the search, once search stagnation is detected, to continue from a high-quality solution that is beyond a certain minimal distance from the current one [84]. As shown in [42], current state-of-the-art algorithms such as Helsgaun's iterated Lin–Kernighan can also suffer from stagnation behavior and, hence, their performance can be further improved by similar ideas.

Finally, let us mention that ILS algorithms have been used as components of more complex algorithms. A clear example is the tour merging approach [2, 21].

The central idea is to generate a set  $G$  of high-quality tours by using ILS and then to post-process these solutions further. In particular, in tour merging, the optimal tour (or, if this is not feasible in reasonable computation time, the best possible tour) is produced from fragments of tours occurring in  $G$ .

### 12.4.2 ILS for Other Problems

ILS algorithms have been applied to a large number of other problems, where often they achieve state-of-the-art performance or are very close to it.

**Single machine total weighted tardiness problem.** Congram, Potts and van de Velde have presented an ILS algorithm for the single machine total weighted tardiness problem (SMTWTP) [20] based on a dynasearch local search. The perturbation mechanism in their ILS algorithm applies a series of random interchange moves and additionally exploits specific properties of the SMTWTP. In the acceptance criterion, Congram et al. introduce a *backtrack step*: after  $\beta$  iterations in which every new local optimum is accepted, the algorithm restarts from the best solution found so far. In our notation, the backtrack step is a particular choice for the history dependence incorporated into the acceptance criterion. The performance of this ILS algorithm was excellent, solving almost all available benchmark instances in a few seconds on the available hardware. A further improvement over this algorithm, mainly based on an enlarged neighborhood being searched within the dynasearch local search, was presented by Grosso et al. [37]. This approach outperformed the first iterated dynasearch algorithm, hence, defining the current state-of-the-art for solving the SMTWTP.

**Single and parallel-machine scheduling.** Brucker et al. [12, 13] apply the principles of ILS to a number of one-machine and parallel-machine scheduling problems. They introduce a local search method which is based on two types of neighborhoods. At each step one goes from one feasible solution to a neighboring one with respect to the secondary neighborhood. The main difference with standard local search methods is that this secondary neighborhood is defined on the set of locally optimal solutions of the first neighborhood. Thus, this is an ILS with two nested neighborhoods; searching in the primary neighborhood corresponds to our local search phase; searching in the secondary neighborhood is like our perturbation phase. The authors also note that the second neighborhood is problem specific; this is what is observed in ILS where the perturbation should be adapted to the problem. The search at a higher level reduces the search space size and at the same time leads to better results.

**Flow shop scheduling.** Stützle [82] applied ILS to the permutation flow shop problem (PFSP) under the makespan criterion. The algorithm is based on a straight forward first-improvement local search using the insert neighborhood while the perturbation is composed of swap moves, which exchange the positions of two adjacent jobs, and interchange moves, which have no adjacency constraint. Experimentally, it was found that perturbations with just a few swap and interchange moves

were sufficient to obtain very good results. Several acceptance criteria have been compared; the best performing was ConstTemp, which corresponds to choosing a constant temperature in the L<sub>SMC</sub> criterion. This ILS algorithm was shown to be among the top performing metaheuristic algorithms for the PFSP [76]; an adaptation of this ILS algorithm has also shown very good performance on the flow shop problem with flowtime objective [27]. The ILS algorithm has also been extended to an iterated greedy (IG) algorithm [77]. The essential idea in IG, and also a few other algorithms [45, 80], is to perturb the current solution by a destruction/construction mechanism. In the solution destruction phase, a complete solution is reduced to a partial solution  $s_p$  by removing solution components; in the following construction phase, a complete solution is reconstructed starting from  $s_p$  by a greedy construction heuristic. Despite the simplicity of the underlying idea, this IG algorithm is a state-of-the-art algorithm for the PFSP [77].

ILS has also been used to solve a flow shop problem with several stages in series, where at each stage a number of machines is available for processing the jobs. Yang et al. [91] presented such a method; at each stage, instead of a single machine, there is a group of identical parallel machines. Their metaheuristic has two phases that are repeated iteratively. In the first phase, the operations are assigned to the machines and an initial sequence is constructed. The second phase uses an ILS to find better schedules for each machine at each stage by modifying the sequence of operations on each machine. Yang et al. also proposed a “hybrid” metaheuristic: they first apply a decomposition procedure to solve a series of single stage sub-problems; then they follow with their ILS. The process is repeated until a satisfactory solution is obtained.

**Job shop scheduling.** Lourenço [56] and Lourenço and Zwijnenburg [58] used ILS to tackle the job shop scheduling problem under the makespan criterion. They performed extensive computational tests, comparing different ways to generate initial solutions, various local search algorithms, different perturbations, and three acceptance criteria. While they found that the initial solution had only a very limited influence, the other components turned out to be very important. Perhaps the heart of their work is the way they perform the perturbations, which has already been described in Section 12.3.2.

Balas and Vazacopoulos [4] presented a variable depth search heuristic, which they called guided local search (GLS). GLS is based on the concept of neighborhood trees, proposed by the authors, where each node corresponds to a solution and the child nodes are obtained by performing an interchange on some critical arc. They developed ILS algorithms by embedding GLS within the shifting bottleneck (SB) procedure and by replacing the reoptimization cycle of SB with a number of cycles of the GLS procedure. They call this procedure SB-GLS1. The later SB-GLS2 variant works as follows. Once all machines have been sequenced, they iteratively remove one machine and apply GLS to a smaller instance defined by the remaining machines. Then again GLS is applied on the initial instance containing *all* machines. Hence, both heuristics are similar in spirit to the one proposed by Lourenço [56] in the sense that they are based on re-optimizing a part of the instance and then reapplying local search to the full one.

Kreipl applied ILS to the total weighted tardiness job shop scheduling problem [53]. His ILS algorithm uses a RW acceptance criterion and the local search consists in reversing critical arcs and arcs adjacent to them. One original aspect of this ILS is the perturbation step: Kreipl applies a few steps of a simulated annealing-like algorithm with constant temperature; in the perturbation phase a smaller neighborhood than the one used in the local search phase is applied. The number of iterations performed during the perturbation phase depends on how good the incumbent solution is. In promising regions, only a few steps are applied to stay near good solutions, otherwise, a “large” perturbation is applied to escape from a poor region. Computational results with the ILS algorithm on a set of benchmark instances have shown a very promising performance. In fact, the algorithm performance is roughly similar to a later, much more complex algorithm proposed by Essafi et al. [29]. Interestingly, this latter approach integrates an ILS algorithm as a local search operator into an evolutionary algorithm, illustrating the fact that ILS can also be used as an improvement method inside other metaheuristics.

**Graph bipartitioning.** The graph bipartitioning problem is among the early ILS applications. Martin and Otto [60, 61] introduced an ILS for this problem following their earlier work on the TSP. For the local search, they used the Kernighan–Lin variable depth local search algorithm [51] which is the analog for this problem of the Lin–Kernighan algorithm. When considering possible perturbations, they noticed a particular weakness of the Kernighan–Lin local search: it frequently generates partitions with many “islands,” i.e., the two sets  $A$  and  $B$  are typically highly fragmented (disconnected). Thus they introduced perturbations that exchanged vertices between these islands rather than between the whole sets  $A$  and  $B$ . Finally, for the acceptance criterion, Martin and Otto used the Better acceptance criterion. The overall algorithm significantly improved over the embedded local search (random restart of the Kernighan–Lin local search); it also improved over simulated annealing when the acceptance criterion was optimized.

**MAX-SAT.** Battiti and Protasi present an application of *reactive search* to the MAX-SAT problem [5]. Their algorithm consists of two phases: a local search phase and a diversification (perturbation) phase. Because of this, their approach fits perfectly into the ILS framework. Their perturbation is obtained by running a tabu search on the current local minimum to guarantee that the modified solution  $s'$  is sufficiently different from the current solution  $s^*$ . Their measure of difference is just the Hamming distance; the minimum distance is set by the length of a tabu list that is adjusted during the algorithm execution. For LocalSearch, they use a standard iterative improvement algorithm appropriate for the MAX-SAT problem. Depending on the distance between  $s'^*$  and  $s^*$ , the tabu list length for the perturbation phase is dynamically adjusted. The next perturbation phase is then started based on solution  $s'^*$ —corresponding to the RW acceptance criterion. This work illustrates very nicely how one can adjust dynamically the perturbation strength in an ILS run. We conjecture that similar schemes will be useful to adapt the perturbation size while running an ILS algorithm. In later work, Smyth et al. [81] have developed an ILS algorithm based on a robust tabu search algorithm that is used in both the local search phase

and the perturbation phase. The main difference between the two phases is that the length of the tabu list is strongly increased in the perturbation to drive the search away from the current solution. Extensive computational tests showed that this algorithm reaches state-of-the-art performance for a number of MAX-SAT instance classes [42, 81]. Noteworthy is also the ILS algorithm of Yagiura and Ibaraki, which is based on large neighborhoods for MAX-SAT that are used in the local search phase [90].

**Quadratic assignment problem.** ILS algorithms have also reached remarkable performance on the QAP [83]. Based on the insights gained through an analysis of the run-time behavior of a basic ILS algorithm with the Better acceptance criterion, Stützle has proposed a number of different ILS algorithms. Population-based extensions of ILS that use restart-type criteria and additional criteria for maintaining solution diversity have been the best performing variants. An extensive experimental campaign has identified this population-based ILS variant as state of the art for structured QAP instances.

**Other problems.** ILS has been applied to a number of other problems and we shortly mention here some of them without attempting to give an exhaustive enumeration. A number of ILS approaches for coloring graphs have been proposed [14, 17, 72]; these approaches generally reach very high-quality colorings and perform particularly well on some structured graphs. ILS algorithms have also been proposed for various vehicle routing problems (VRPs), including time-dependent VRPs [39], VRPs with time penalty functions [44], the prize-collecting VRP [87], and a multiple depot vehicle scheduling problem [54]. ILS algorithms have also been successfully applied to the car sequencing problem proposed in the 2005 ROADEF challenge, as illustrated in [23, 73]. ILS is used as a local search procedure within a GRASP approach by Ribeiro and Urrutia for tackling the mirrored traveling tournament problem [74]. Very high-performing ILS algorithms have also been proposed for problems such as maximum clique [50], image registration [24], some loop layout problems [10], linear ordering [19, 78], logistic network design problems [22], a capacitated hub location problem [75], Bayesian networks structure learning [25], and minimum sum-of-squares clustering [64].

### 12.4.3 Summary

The examples we have chosen in this section stress several points that have already been mentioned. First, the choice of the local search algorithm is usually quite critical if one is to obtain peak performance. In most applications, the best performing ILS algorithms apply much more sophisticated local search algorithms than simple best or first-improvement methods. Second, the other components of an ILS also need to be optimized if state-of-the-art results are to be achieved. This optimization should be global and should involve the use of problem-specific properties. Examples of this last point were given in scheduling applications where good perturbations were not simply random, but rather involved re-optimization of significant parts of the instance (c.f. the job shop case).

The final picture is one where (i) ILS is a versatile metaheuristic, which can be easily adapted to different combinatorial optimization problems and (ii) sophisticated perturbation schemes and search diversification are essential ingredients to achieve the best possible ILS performance.

## 12.5 Relation to Other Metaheuristics

In this section, we highlight the similarities and differences between ILS and other well-known metaheuristics. We shall distinguish metaheuristics that are essentially variants of local search and those that generate solutions using a mechanism that is not necessarily based on an explicit neighborhood structure. Among the first class, which we call *neighborhood-based metaheuristics*, are methods like simulated annealing (SA) [16, 52], tabu search (TS) [36], or guided local search (GLS) [89]. The second class comprises metaheuristics like GRASP [30], ant colony optimization (ACO) [28], evolutionary and memetic algorithms [3, 67, 69], scatter search [35], variable neighborhood search (VNS) [38, 68], and ILS. Some metaheuristics of this second class, like evolutionary algorithms and ant colony optimization, do not necessarily make use of local search algorithms; however, a local search can be embedded in them, in which case the performance is usually enhanced [28, 69, 70]. The other metaheuristics in this class explicitly use embedded local search algorithms as an essential part of their structure. For simplicity, we will assume in what follows that all the metaheuristics of this second class do incorporate local search algorithms. In this case, such metaheuristics generate iteratively input solutions that are passed to a local search; they can thus be interpreted as multi-start algorithms in the most general meaning of that term. This is why we call them here *multi-start-based metaheuristics*.

### 12.5.1 Neighborhood-Based Metaheuristics

Neighborhood-based metaheuristics are extensions of iterative improvement algorithms. They avoid getting stuck in locally optimal solutions by allowing moves to worse solutions in the neighborhood of the current solution. Metaheuristics in this class differ mainly by their move strategies. In the case of SA, the neighborhood is sampled randomly and worse solutions are accepted with a probability, which depends on a temperature parameter and the degree of deterioration incurred; better neighboring solutions are usually accepted while much worse neighboring solutions are accepted with a low probability. In the case of (simple) TS strategies, the neighborhood is explored in an aggressive way and cycles are avoided by declaring tabu attributes of visited solutions. Finally, in the case of GLS, the evaluation function is dynamically modified by penalizing certain solution components. This allows the search to escape from a solution that is a local optimum of the original objective function.

Obviously, any of these neighborhood-based metaheuristics can be used as the local search procedure in ILS. In general, however, these metaheuristics do not halt, so it is necessary to limit their run time if they are to be embedded in ILS. One particular advantage of combining neighborhood-based metaheuristics with ILS is that they often obtain much better solutions than iterative improvement algorithms. But this advantage usually comes at the cost of larger computation times. Since these metaheuristics allow one to obtain better solutions at the expense of greater computation times, we are confronted with the following optimization problem when using them within an ILS:<sup>7</sup> “For how long should one run the embedded search in order to achieve the best trade-off between computation time and solution quality?” This is analogous to the question of whether it is best to have a fast but not so effective local search or a slower but a more powerful one. The answer depends of course on the total computation time available and on how the costs improve with time.

A different type of connection between ILS, SA, and TS arises from certain similarities in the algorithms. For example, SA can be seen as an ILS without a local search phase (SA samples the original space  $\mathcal{S}$  and not the reduced space  $\mathcal{S}^*$ ) and where the acceptance criteria are  $\text{LSMC}(s^*, s^{*\prime}, \text{history})$ . While SA does not employ memory, the use of memory is the main feature of TS, which makes a strong use of historical information at multiple levels. Given its effectiveness, we expect that the integration of memories will become widespread in future ILS applications.<sup>8</sup> Furthermore, since TS is a prototype for memory intensive search procedures, it can be a valuable source of inspiration for deriving ILS variants with a more direct usage of memory; this can lead to a better balance between intensification and diversification in the search.<sup>9</sup> Similarly, TS strategies may also be improved by features of ILS algorithms and by some insights gained from the research on ILS.

### 12.5.2 Multi-start-Based Metaheuristics

Multi-start-based metaheuristics can be classified into *constructive* metaheuristics and *perturbation-based* metaheuristics.

Well-known examples of constructive metaheuristics are ACO and GRASP, which both use a probabilistic solution construction phase. An important difference between ACO and GRASP is that ACO has an indirect memory of the search process, which is used to bias the construction process, whereas GRASP does not

<sup>7</sup> This question is not specific to ILS; it arises for all multi-start-based metaheuristics.

<sup>8</sup> In early TS publications, proposals similar to the use of perturbations were put forward under the name *random shakeup* [32]. These procedures were characterized as a “randomized series of moves that leads the heuristic (away) from its customary path” [32]. The relationship to perturbations in ILS is obvious.

<sup>9</sup> Indeed, in [33], Glover uses “strategic oscillation” whereby one cycles over these procedures: the simplest moves are used till there is no more improvement, and then progressively more advanced moves are used.

use that kind of memory. An obvious difference between ILS and constructive metaheuristics is that ILS does not construct solutions. However, both generate a sequence of solutions, and if the constructive metaheuristic uses an embedded local search, both go from one local minimum to another. So it might be said that the perturbation phase of an ILS is replaced by a (memory-dependent) construction phase in these constructive metaheuristics. But another connection can be made: ILS can be used instead of the embedded “local search” in ACO or GRASP. (This is exactly what is done, for example, in [74].) This is one way to generalize ILS, but it is not specific to these kinds of metaheuristics: whenever one has an embedded local search, one can try to replace it by an iterated local search.

Perturbation-based metaheuristics differ in the techniques they use to actually perturb solutions. Before going into details, let us introduce one additional feature for classifying metaheuristics: we will distinguish between population-based algorithms and those that use a single current solution (a population is of size one). For example, evolutionary algorithms, memetic algorithms, scatter search, and ACO are population-based, while ILS uses a single solution at each step. Whether or not a metaheuristics is population-based is important for the type of perturbation that can be applied. If no population is used, new solutions are generated by applying perturbations to single solutions; this is what happens for ILS and VNS. If a population is present, one can also use the possibility of recombining several solutions into a new one. Such combinations of solutions are implemented by “crossover” operators in evolutionary algorithms or in the recombination of multiple solutions in scatter search.

In general, population-based metaheuristics are more complex to use than those following a single solution: they require mechanisms to manage a population of solutions and more importantly it is necessary to find effective operators for the combination of solutions. Most often, this last task is a real challenge. The complexity of population-based local search methods can be justified if they lead to better performance than non population-based methods. Therefore, one question of interest is whether using a population of solutions is really useful. Clearly, for some problems such as the TSP with high cost–distance correlations, the use of a single element in the population leads to good results, so the advantage of population-based methods is small or may become only noticeable if very high computation times are invested. However, for other problems, the use of a population can be an appealing way to achieve search diversification. Thus, population-based methods may be desirable if their complexity is not overwhelming. Because of this, population-based extensions of ILS are promising approaches.

To date, several population-based extensions of ILS have been proposed [41, 83, 85, 88]. The approaches proposed in [41, 85] keep the simplicity of ILS algorithms by maintaining unchanged the perturbations: one parent is perturbed to give one child. More complex population-based ILS extensions with mechanisms for maintaining diversity in the population are considered in [83]. A population of solutions is used in [88] to restrict the perturbation to explore only parts of solutions where pairs of solutions differ (similar in spirit to the genetic transformations [49]) and to reduce the size of the neighborhood in the local search.

Finally, let us discuss VNS, which is the metaheuristic closest to ILS. VNS begins by observing that the concept of local optimality is conditional on the neighborhood structure used in a local search. Then VNS systemizes the idea of changing the neighborhood during the search to avoid getting stuck in poor quality solutions. Several VNS variants have been proposed. The most widely used one, *basic VNS*, can be seen as an ILS algorithm, which uses the Better acceptance criterion and a systematic way of varying the perturbation strength. To do so, basic VNS orders neighborhoods as  $\mathcal{N}_1, \dots, \mathcal{N}_m$  where the order is chosen according to the neighborhood size. Let  $k$  be a counter variable,  $k = 1, 2, \dots, m$ , and initially set  $k = 1$ . If the perturbation and the subsequent local search lead to a new best solution, then  $k$  is reset to 1, otherwise  $k$  is increased by one. We refer to [38] for a description of other VNS variants.

A major difference between ILS and VNS is the philosophy underlying the two metaheuristics: ILS has the explicit goal of building a walk in the set of locally optimal solutions, while VNS algorithms are derived from the idea of systematically changing neighborhoods during the search.

Clearly, there are major points in common between most of today's high-performance metaheuristics. How can one summarize how ILS differs from the others? We shall proceed by enumeration as the diversity of today's metaheuristics seems to forbid any simpler approach. When compared to ACO and GRASP, we see that ILS uses perturbations to create new solutions; this is quite different in principle and in practice from using construction. When compared to evolutionary algorithms, memetic algorithms, and scatter search, we see that ILS, as we defined it, has a population of size one; therefore no recombination operators need be defined. We could continue like this, but we cannot expect the boundaries between all metaheuristics to be clear-cut. Not only are hybrid methods very often the way to go, but most often one can smoothly go from one metaheuristic to another. In addition, as mentioned at the beginning of this chapter, the distinction between heuristic and metaheuristic is rarely unambiguous. So our point of view is not that ILS has essential features that are absent in other metaheuristics; rather, when considering the basic structure of ILS, some simple yet powerful ideas transpire, and these can be of use in most metaheuristics, being close or not in spirit to ILS.

## 12.6 Conclusions

ILS has many of the desirable features of a metaheuristic: it is simple, easy to implement, robust, and highly effective. The essential idea of ILS lies in focusing the search not on the full space of solutions but on a smaller subspace defined by the solutions that are locally optimal for a given optimization engine. The success of ILS lies in the *biased* sampling of this set of local optima. How effective this approach turns out to be depends mainly on the choice of the local search, the perturbation, and the acceptance criterion. Interestingly, even when using the most naive implementations of these components, ILS can do much better than random restart. But, with further work to carefully adapt the components to the problem at hand, ILS

can often become a competitive or even state-of-the-art algorithm. This dichotomy is important because the optimization of the algorithm can be done progressively, and so ILS can be kept at any desired level of simplicity. This, plus the modular nature of ILS, leads to short development times and gives ILS an edge over more complex metaheuristics in the world of industrial applications. As an example of this, recall that ILS essentially treats the embedded heuristic as a black box; then upgrading an ILS to take advantage of a new and better local search algorithm is nearly immediate. Because of all these features, we believe that ILS is a promising and powerful algorithm to solve real complex problems in industry and services, in areas ranging from finance to production management and logistics. Finally, let us note that even if this review was presented in the context of tackling combinatorial optimization problems, in reality much of what we covered can be extended in a straightforward manner to continuous optimization problems.

Looking ahead toward future research directions, we expect ILS to be applied to new kinds of problems. Some challenging examples are (i) problems where the constraints are so restrictive that most metaheuristics fail; (ii) multi-objective problems that bring us closer to real problems; and (iii) dynamic or real-time problems where the data about the instance are received or vary during the solution process.

The ideas and results presented in this chapter leave many questions unanswered. Clearly, more work needs to be done to better understand the interplay between the ILS modules `GenerateInitialSolution`, `Perturbation`, `LocalSearch`, and `AcceptanceCriterion`. Other directions for improving ILS performance are to consider the intelligent use of memory, explicit intensification and diversification strategies, and greater problem-specific tuning. The exploration of these issues will certainly lead to higher performance iterated local search algorithms.

**Acknowledgments** Olivier Martin acknowledges support from the Institut Universitaire de France, Helena Lourenço acknowledges support from the Ministerio de Educacion y Ciencia, Spain, MEC-SEJ2006-12291, and Thomas Stützle acknowledges support from the F.R.S.-FNRS, of which he is a Research Associate. This work was supported by the META-X project, an *Action de Recherche Concertée* funded by the Scientific Research Directorate of the French Community of Belgium.

## References

1. Applegate, D., Cook, W.J., Rohe, A.: Chained Lin-Kernighan for large traveling salesman problems. *INFORMS J. Compt.* **15**(1), 82–92 (2003)
2. Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J.: *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, Princeton, NJ (2006)
3. Bäck, T.: *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, Oxford, UK (1996)
4. Balas, E., Vazacopoulos, A.: Guided local search with shifting bottleneck for job shop scheduling. *Manag. Sci.* **44**(2), 262–275 (1998)
5. Battiti, R., Protasi, M.: Reactive search, a history-based heuristic for MAX-SAT. *ACM J. Exp. Algorithmics* 2 (1997)
6. Battiti, R., Tecchiolli, G.: The reactive tabu search. *ORSA J. Comput.* **6**(2), 126–140 (1994)
7. Baum, E.B.: Iterated descent: a better algorithm for local search in combinatorial optimization problems. Technical Report, Caltech, Pasadena, CA, (1986). manuscript

8. Baum, E.B.: Towards practical “neural” computation for combinatorial optimization problems. In: Denker, J. (ed.) *Neural Networks for Computing*, pp. 53–64, (1986). AIP conference proceedings, Snowbird, Utah
9. Baxter, J.: Local optima avoidance in depot location. *J. Oper. Res. Soc.* **32**, 815–819, (1981)
10. Bennell, J.A., Potts, C.N., Whitehead, J.D.: Local search algorithms for the min-max loop layout problem. *J. Oper. Res. Soc.* **53**(10), 1109–1117 (2002)
11. Bentley, J.L.: Fast algorithms for geometric traveling salesman problems. *ORSA J. Comput.*, **4**(4), 387–411 (1992)
12. Brucker, P., Hurink, J., and Werner, F. Improving local search heuristics for some scheduling problems—part I. *Discrete Appl. Math.* **65**(1–3), 97–122 (1996)
13. Brucker, P., Hurink, J., and Werner, F. Improving local search heuristics for some scheduling problems—part II. *Discrete Appl. Math.* **72**(1–2), 47–69 (1997)
14. Caramia, M., Dell’Olmo, P.: Coloring graphs by iterated local search traversing feasible and infeasible solutions. *Discrete Appl. Math.* **156**(2), 201–217 (2008)
15. Carlier, J.: The one-machine sequencing problem. *Eur. J. Oper. Res.* **11**:42–47 (1982)
16. Černý, V.: A thermodynamical approach to the traveling salesman problem. *J. Optim. Theory Appl.* **45**(1), 41–51 (1985)
17. Chiarandini, M., Stützle, T.: An application of iterated local search to the graph coloring problem. In: Mehrotra, A., Johnson, D.S., Trick, M. (eds.) *Proceedings of the Computational Symposium on Graph Coloring and its Generalizations*, pp. 112–125, Ithaca, NY (2002)
18. Codenotti, B., Manzini, G., Margara, L., Resta, G.: Perturbation: an efficient technique for the solution of very large instances of the Euclidean TSP. *INFORMS J. Comput.* **8**(2), 125–133 (1996)
19. Congram, R.K.: Polynomially Searchable Exponential Neighbourhoods for Sequencing Problems in Combinatorial Optimization. PhD thesis, Southampton University, Faculty of Mathematical Studies, Southampton, UK (2000)
20. Congram, R.K., Potts, C.N., van de Velde, S.: An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS J. Comput.* **14**(1), 52–67 (2002)
21. Cook, W.J., Seymour, P.: Tour merging via branch-decomposition. *INFORMS J. Comput.* **15**(3), 233–248 (2003)
22. Cordeau, J.-F., Laporte, G., Pasin, F.: An iterated local search heuristic for the logistics network design problem with single assignment. *Int. J. Production Econ.* **113**(2), 626–640 (2008)
23. Cordeau, J.-F., Laporte, G., Pasin, F.: Iterated tabu search for the car sequencing problem. *Eur. J. Oper. Res.* **191**(3), 945–956 (2008)
24. Cordón, O., Damas, S.: Image registration with iterated local search. *J. Heuristics* **12**(1–2), 73–94 (2006)
25. de Campos, L.M., Fernández-Luna, J.M., Miguel Puerta, J.: An iterated local search algorithm for learning Bayesian networks with restarts based on conditional independence tests. *Int. J. Intell. Syst.* **18**, 221–235, (2003)
26. den Besten, M.L., Stützle, T., Dorigo, M.: Design of iterated local search algorithms: An example application to the single machine total weighted tardiness problem. In: Boers, E.J.W. et al. (eds.) *Applications of Evolutionary Computing*. Lecture Notes in Computer Science, vol. 2037, pp. 441–452. Springer, Berlin (2001)
27. Dong, X., Huang, H., Chen, P.: An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion. *Comput. Oper. Res.* **36**(5), 1664–1669 (2009)
28. Dorigo, M., Stützle, T.: *Ant Colony Optimization*. MIT Press, Cambridge, MA (2004)
29. Essafi, I., Mati, Y., Dauzère-Péréz, S.: A genetic local search algorithm for minimizing total weighted tardiness in the job-shop scheduling problem. *Comput. Oper. Res.* **35**(8), 2599–2616 (2008)
30. Feo, T.A., Resende, M.G.C. Greedy randomized adaptive search procedures. *J. Global Optim.* **6**, 109–133 (1995)
31. Fonlupt, C., Robilliard, D., Preux, P., Talbi, E.-G.: Fitness landscape and performance of meta-heuristics. In: Voss, S., Martello, S., Osman, I.H., Roucairol, C. (eds.) *Meta-Heuristics:*

- Advances and Trends in Local Search Paradigms for Optimization, pp. 257–268. Kluwer, Boston, MA (1999)
- 32. Glover, F.: Future paths for integer programming and links to artificial intelligence. *Comput. Oper. Res.* **13**(5), 533–549 (1986)
  - 33. Glover, F.: Tabu search – part I. *ORSA J. Comput.* **1**(3), 190–206 (1989)
  - 34. Glover, F.: Tabu thresholding: improved search by nonmonotonic trajectories. *ORSA J. Comput.* **7**(4), 426–442 (1995)
  - 35. Glover, F.: Scatter search and path relinking. In: Corne, D., Dorigo, M., Glover, F. (eds.) *New Ideas in Optimization*, pp. 297–316. McGraw Hill, London, UK (1999)
  - 36. Glover, F., Laguna, M.: *Tabu Search*. Kluwer, Boston, MA (1997)
  - 37. Grosso, A., Della Croce, F., Tadei, R.: An enhanced dynasearch neighborhood for the single-machine total weighted tardiness scheduling problem. *Oper. Res. Lett.* **32**(1), 68–72 (2004)
  - 38. Hansen, P., Mladenović, N.: Variable neighborhood search: principles and applications. *Eur. J. Oper. Res.* **130**(3), 449–467 (2001)
  - 39. Hashimoto, H., Yagiura, M., Ibaraki, T.: An iterated local search algorithm for the time-dependent vehicle routing problem with time windows. *Discrete Optim.* **5**(2), 434–456 (2008)
  - 40. Helsgaun, K.: An effective implementation of the Lin-Kernighan traveling salesman heuristic. *Eur. J. Oper. Res.* **126**(1), 106–130 (2000)
  - 41. Hong, I., Kahng, A.B., Moon, B.R.: Improved large-step Markov chain variants for the symmetric TSP. *J. Heuristics* **3**(1), 63–81 (1997)
  - 42. Hoos, H.H., Stützle, T.: *Stochastic Local Search—Foundations and Applications*. Morgan Kaufmann, San Francisco, CA (2005)
  - 43. Hu, T.C., Kahng, A.B., Tsao, C.-W.A.: Old bachelor acceptance: a new class of non-monotone threshold accepting methods. *ORSA J. Comput.* **7**(4), 417–425 (1995)
  - 44. Ibaraki, T., Imahori, S., Nonobe, K., Sobue, K., Uno, T., Yagiura, M.: An iterated local search algorithm for the vehicle routing problem with convex time penalty functions. *Discrete Appl. Math.* **156**(11), 2050–2069 (2008)
  - 45. Jacobs, L.W., Brusco, M.J.: A local search heuristic for large set-covering problems. *Naval Res. Logistics* **42**(7), 1129–1140 (1995)
  - 46. Johnson, D.S.: Local optimization and the travelling salesman problem. In: Proceedings of the 17th Colloquium on Automata, Languages, and Programming. Lecture Notes in Computer Science, vol. 443, pp. 446–461. Springer, Berlin (1990)
  - 47. Johnson, D.S., McGeoch, L.A.: The travelling salesman problem: a case study in local optimization. In: Aarts, E.H.L., Lenstra, J.K. (eds.) *Local Search in Combinatorial Optimization*, pp. 215–310. Wiley, Chichester (1997)
  - 48. Johnson, D.S., McGeoch, L.A.: Experimental analysis of heuristics for the STSP. In: Gutin, G., Punnen, A. (eds.) *The Traveling Salesman Problem and its Variations*, pp. 369–443. Kluwer, Dordrecht (2002)
  - 49. Katayama, K., Narihisa, H.: Iterated local search approach using genetic transformation to the traveling salesman problem. In: Banzhaf, W., Daida, J., Eiben, A.E., Garzon, M.H., Honavar, V., Jakieła, M., Smith, R.E. (eds.) *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-1999)*, vol. 1, pp. 321–328. Morgan Kaufmann, San Francisco, CA (1999)
  - 50. Katayama, K., Sadamatsu, M., Narihisa, H.: Iterated  $k$ -opt local search for the maximum clique problem. In: Cotta, C., van Hemert, J. (eds.) *Evolutionary Computation in Combinatorial Optimization*. Lecture Notes in Computer Science, vol. 4446, pp. 84–95. Springer, Berlin (2007)
  - 51. Kernighan, B.W., Lin, S.: An efficient heuristic procedure for partitioning graphs. *Bell Syst. Technol. J.* **49**, 213–219 (1970)
  - 52. Kirkpatrick, S., Gelatt Jr, C.D., Vecchi, M.P.: Optimization by simulated annealing. *Science* **220**, 671–680 (1983)
  - 53. Kreipl, S.: A large step random walk for minimizing total weighted tardiness in a job shop. *J. Scheduling* **3**(3), 125–138 (2000)

54. Laurent, B., Hao, J.-K.: Iterated local search for the multiple depot vehicle scheduling problem. *Comput. Industrial Eng.* **57**(1), 277–286 (2009)
55. Lin, S., Kernighan, B.W.: An effective heuristic algorithm for the travelling salesman problem. *Oper. Res.* **21**, 498–516 (1973)
56. Lourenço, H.R.: Job-shop scheduling: computational study of local search and large-step optimization methods. *Eur. J. Oper. Res.* **83**(2), 347–364 (1995)
57. Lourenço, H.R.: A polynomial algorithm for a special case of the one-machine scheduling problem with time-lags. Technical Report Economic Working Papers Series, No. 339, Universitat Pompeu Fabra (1998)
58. Lourenço, H.R., Zwijnenburg, M.: Combining the large-step optimization with tabu-search: Application to the job-shop scheduling problem. In: Osman, I.H., Kelly, J.P. (eds.) *Meta-Heuristics: Theory & Applications*, pp. 219–236. Kluwer, Boston, MA (1996)
59. Lozano, M., García-Martínez, C.: An evolutionary ILS-perturbation technique. In: Blesa, M.J., Blum, C., Cotta, C., Fernández, A.J., Gallardo, J.E., Roli, A., Sampels, M. (eds.) *Hybrid Metaheuristics, 5th International Workshop, HM 2008. Lecture Notes in Computer Science*, vol. 5296, pp. 1–15. Springer, Berlin (2008)
60. Martin, O., Otto, S.W.: Partitioning of unstructured meshes for load balancing. *Concurrency: Pract. Exp.* **7**, 303–314 (1995)
61. Martin, O., Otto, S.W.: Combining simulated annealing with local search heuristics. *Ann. Oper. Res.* **63**, 57–75 (1996)
62. Martin, O., Otto, S.W., Felten, E.W.: Large-step Markov chains for the traveling salesman problem. *Complex Syst.* **5**(3), 299–326 (1991)
63. Martin, O., Otto, S.W., Felten, E.W.: Large-step Markov chains for the TSP incorporating local search heuristics. *Oper. Res. Lett.* **11**, 219–224 (1992)
64. Merz, P.: An iterated local search approach for minimum sum-of-squares clustering. In: Berthold, M.R., Lenz, H.-J., Bradley, E., Kruse, R., Borgelt, C. (eds.) *Advances in Intelligent Data Analysis V, IDA 2003. Lecture Notes in Computer Science*, vol. 2810 pp. 286–296. Springer, Berlin (2003)
65. Merz, P., Huhse, J.: An iterated local search approach for finding provably good solutions for very large TSP instances. In: Rudolph, G., Jansen, T., Lucas, S.M., Poloni, C., Beume, N. (eds.) *Parallel Problem Solving from Nature—PPSN X. Lecture Notes in Computer Science*, vol. 5199, pp. 929–939. Springer, Berlin (2008)
66. Mézard, M., Parisi, G., Virasoro, M.A.: *Spin-Glass Theory and Beyond*. Lecture Notes in Physics. vol. 9, World Scientific, Singapore (1987)
67. Michalewicz, Z., Fogel, D.B.: *How to Solve it: Modern Heuristics*. Springer, Berlin (2000)
68. Mladenović, N., Hansen, P.: Variable neighborhood search. *Comput. Oper. Res.* **24**, 1097–1100 (1997)
69. Moscato, P., Cotta, C.: Memetic algorithms. In: González, T.F. (ed.) *Handbook of Approximation Algorithms and Metaheuristics. Computer and Information Science Series*, Chapter 27. Chapman & Hall/CRC, Boca Raton, FL (2007)
70. Mühlenbein, H.: Evolution in time and space – the parallel genetic algorithm. In: Rawlings, G.J.E. (ed.) *Foundations of Genetic Algorithms*, pp. 316–337. Morgan Kaufmann, San Mateo, CA (1991)
71. Nawaz, M., Enscore Jr., E., Ham, I.: A heuristic algorithm for the  $m$ -machine,  $n$ -job flow-shop sequencing problem. *OMEGA* **11**(1), 91–95 (1983)
72. Paquete, L., Stützle, T.: An experimental investigation of iterated local search for coloring graphs. In: Cagnoni, S., Gottlieb, J., Hart, E., Middendorf, M., Raidl, G. (eds.) *Applications of Evolutionary Computing. Lecture Notes in Computer Science*, vol. 2279, pp. 122–131. Springer, Berlin (2002)
73. Ribeiro, C.C., Aloise, D., Noronha, T.F., Rocha, C., Urrutia, S.: A hybrid heuristic for a multi-objective real-life car sequencing problem with painting and assembly line constraints. *Eur. J. Oper. Res.* **191**(3), 981–992 (2008)
74. Ribeiro, C.C., Urrutia, S.: Heuristics for the mirrored traveling tournament problem. *Eur. J. Oper. Res.* **179**(3), 775–787 (2007)

75. Rodríguez-Martín, I., Salazar González, J.J.: Solving a capacitated hub location problem. *Eur. J. Oper. Res.* **184**(2), 468–479 (2008)
76. Ruiz, R., Maroto, C.: A comprehensive review and evaluation of permutation flowshop heuristics. *Eur. J. Oper. Res.* **165**(2), 479–494 (2005)
77. Ruiz, R., Stützle, T.: A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *Eur. J. Oper. Res.* **177**(3), 2033–2049 (2007)
78. Schiavonotto, T., Stützle, T.: The linear ordering problem: instances, search space analysis and algorithms. *J. Math. Modelling Algorithms* **3**(4), 367–402 (2004)
79. Schreiber, G.R., Martin, O.C.: Cut size statistics of graph bisection heuristics. *SIAM J. Optim.* **10**(1), 231–251 (1999)
80. Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H., Dueck, G.: Record breaking optimization results using the ruin and recreate principle. *J. Comput. Phys.* **159**(2), 139–171 (2000)
81. Smyth, K., Hoos, H.H., Stützle, T.: Iterated robust tabu search for MAX-SAT. In: Xiang, Y., Chaib-draa, B. (eds.) *Advances in Artificial Intelligence, 16th Conference of the Canadian Society for Computational Studies of Intelligence. Lecture Notes in Computer Science*, vol. 2671, pp. 129–144. Springer, Berlin (2003)
82. Stützle, T.: Applying iterated local search to the permutation flow shop problem. Technical Report AIDA-98-04, FG Intellektik, TU Darmstadt, Darmstadt, Germany, August (1998)
83. Stützle, T.: Iterated local search for the quadratic assignment problem. *Eur. J. Oper. Res.* **174**(3), 1519–1539 (2006)
84. Stützle, T., Hoos, H.H.: Analysing the run-time behaviour of iterated local search for the travelling salesman problem. In: Hansen, P., Ribeiro, C. (eds.) *Essays and Surveys on Meta-heuristics, Operations Research/Computer Science Interfaces Series*, pp. 589–611. Kluwer, Boston, MA (2001)
85. Stützle, T.: *Local Search Algorithms for Combinatorial Problems: Analysis, Improvements, and New Applications. Dissertations in Artificial Intelligence*, vol. 220. IOS Press, Amsterdam, The Netherlands (1999)
86. Taillard, É.D.: Comparison of iterative searches for the quadratic assignment problem. *Location Sci.* **3**, 87–105 (1995)
87. Tang, L., Wang, X.: Iterated local search algorithm based on a very large-scale neighborhood for prize-collecting vehicle routing problem. *Int. J. Adv. Manufact. Technol.* **29**(11–12), 1246–1258 (2006)
88. Thierens, D.: Population-based iterated local search: restricting the neighborhood search by crossover. In: Deb, K. et al. (eds.) *Genetic and Evolutionary Computation—GECCO 2004, Part II. Lecture Notes in Computer Science*, vol. 3102, pp. 234–245. Springer, Berlin (2004)
89. Voudouris, C., Tsang, E.: Guided local search. Technical Report Technical Report CSM-247, Department of Computer Science, University of Essex, Colchester, UK (1995)
90. Yagiura, M., Ibaraki, T.: Efficient 2 and 3-flip neighborhood search algorithms for the MAX SAT: experimental evaluation. *J. Heuristics* **7**(5), 423–442 (2001)
91. Yang, Y., Kreipl, S., Pinedo, M.: Heuristics for minimizing total weighted tardiness in flexible flow shops. *J. Scheduling* **3**(2), 89–108 (2000)



# Chapter 13

## Large Neighborhood Search

David Pisinger and Stefan Ropke

**Abstract** Heuristics based on *large neighborhood search* have recently shown outstanding results in solving various transportation and scheduling problems. Large neighborhood search methods explore a complex neighborhood by use of heuristics. Using large neighborhoods makes it possible to find better candidate solutions in each iteration and hence traverse a more promising search path. Starting from the large neighborhood search method, we give an overview of *very large scale neighborhood* search methods and discuss recent variants and extensions like *variable depth search* and *adaptive large neighborhood search*.

### 13.1 Introduction

The topic of this chapter is the metaheuristic *large neighborhood search* (LNS) proposed by Shaw [50]. In LNS, an initial solution is gradually improved by alternately destroying and repairing the solution. The LNS heuristic belongs to the class of heuristics known as *very large scale neighborhood search* (VLSN) algorithms [2]. All VLSN algorithms are based on the observation that searching a large neighborhood results in finding local optima of high quality, and hence overall a VLSN algorithm may return better solutions. However, searching a large neighborhood is time consuming; hence various filtering techniques are used to limit the search. In VLSN algorithms, the neighborhood is typically restricted to a subset of

---

David Pisinger

Department of Management Engineering, Technical University of Denmark, Produktionstorvet,  
Building 426, 2800 Kgs. Lyngby, Denmark  
e-mail: pisinger@man.dtu.dk

Stefan Ropke

Department of Transport, Technical University of Denmark, Bygningstorvet, Building 115, 2800  
Kgs. Lyngby, Denmark  
e-mail: sr@transport.dtu.dk

the solutions which can be searched efficiently. In LNS the neighborhood is implicitly defined by methods (often heuristics) which are used to destroy and repair an incumbent solution.

The two similar terms LNS and VLSN may cause confusion. We consistently use VLSN for the broad class of algorithms that searches very large neighborhoods and LNS for the particular metaheuristic, belonging to the class of VLSN algorithms, that is described in Section 13.2.

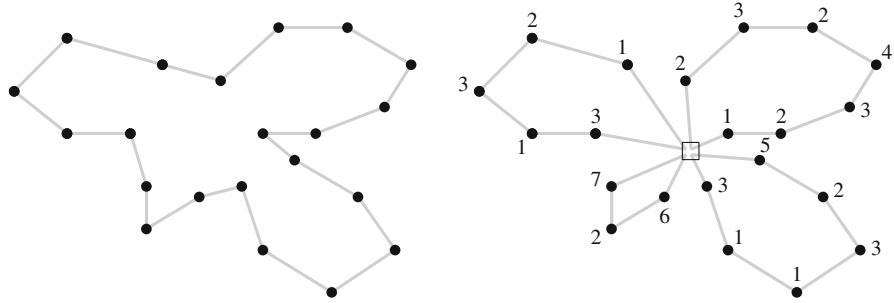
In the rest of the introduction we first define two example problems in Section 13.1.1. We then define neighborhood search algorithms in Section 13.1.2 and the class of VLSN algorithms in Section 13.1.3. In Sections 13.1.3.1–13.1.3.3 we describe three subclasses of VLSN algorithms. In Section 13.2 we describe the LNS heuristic and an extension called adaptive large neighborhood search. This is followed in Section 13.3 by a survey of LNS applications and the chapter is concluded in Section 13.4.

### 13.1.1 Example Problems

Throughout this chapter we will refer to two example problems: the *traveling salesman problem* (TSP) and the *capacitated vehicle routing problem* (CVRP). The TSP is probably the most studied and well-known combinatorial optimization problem. In the TSP a salesman has to visit a number of cities. The salesman must perform a tour through all the cities such that the salesman returns to his starting city at the end of the tour. More precisely we are given an undirected graph  $G = (V, E)$  in which each edge  $e \in E$  has an associated cost  $c_e$ . The goal of the TSP is to find a cyclic tour, such that each vertex is visited exactly once. The sum of the edge costs used in the tour must be minimized. We recommend [3] for more information about the TSP.

In the CVRP, which is a generalization of the TSP, one has to serve a set of customers using a fleet of homogeneous vehicles based at a common depot. Each customer has a certain demand for goods which initially are located at the depot. The task is to design vehicle routes starting and ending at the depot such that all customer demands are fulfilled.

The CVRP can be defined more precisely as follows. We are given an undirected graph  $G = (V, E)$  with vertices  $V = \{0, \dots, n\}$  where vertex 0 is the depot and the vertices  $N = \{1, \dots, n\}$  are customers. Each edge  $e \in E$  has an associated cost  $c_e$ . The demand of each customer  $i \in N$  is given by a positive quantity  $q_i$ . Moreover,  $m$  homogeneous vehicles are available at the depot and the capacity of each vehicle is equal to  $Q$ . The goal of the CVRP is to find exactly  $m$  routes, starting and ending at the depot, such that each customer is visited exactly once by a vehicle and such that the sum of demands of the customers on each route is less than or equal to  $Q$ . The sum of the edge costs used in the  $m$  routes must be minimized. We recommend [54] for further information about the CVRP and vehicle routing problems in general. An example of a TSP and a CVRP solution is shown in Figure 13.1.



**Fig. 13.1** Left: A TSP solution. Right: A CVRP solution. In the CVRP the depot is marked with a square and the customers  $i$  are marked with nodes each having a demand  $q_i$ .

### 13.1.2 Neighborhood Search

In this section we formally introduce the term neighborhood search. We are given an instance  $I$  of a combinatorial optimization problem where  $X$  is the set of feasible solutions for the instance (we write  $X(I)$  when we need to emphasize the connection between instance and solution set) and  $c : X \rightarrow \mathbb{R}$  is a function that maps from a solution to its *cost*.  $X$  is assumed to be finite, but is usually an extremely large set. We assume that the combinatorial optimization problem is a minimization problem, that is, we want to find a solution  $x^*$  such that  $c(x^*) \leq c(x) \forall x \in X$ .

We define a *neighborhood* of a solution  $x \in X$  as  $N(x) \subseteq X$ ; that is,  $N$  is a function that maps a solution to a set of solutions. A solution  $x$  is said to be *locally optimal* or a *local optimum* with respect to a neighborhood  $N$  if  $c(x) \leq c(x') \forall x' \in N(x)$ . With these definitions it is possible to define a neighborhood search algorithm. The algorithm takes an initial solution  $x$  as input. It computes  $x' = \arg \min_{x'' \in N(x)} \{c(x'')\}$ , that is, it finds the cheapest solution  $x'$  in the neighborhood of  $x$ . If  $c(x') < c(x)$  then the algorithm performs the update  $x = x'$ . The neighborhood of the new solution  $x$  is searched for an improving solution and this is repeated until a local optimum  $x$  is reached. When this happens the algorithm stops. The algorithm is denoted a *steepest descent* algorithm as it always chooses the best solution in the neighborhood.

A simple example of a neighborhood for the TSP is the *2-opt* neighborhood which can be traced back to [16]. The neighborhood of a solution  $x$  in the 2-opt neighborhood is the set of solutions that can be reached from  $x$  by deleting two edges in  $x$  and adding two other edges in order to reconnect the tour. A simple example of a neighborhood for the CVRP is the *relocate* neighborhood (see, e.g., [26]). In this neighborhood,  $N(x)$  is defined as the set of solutions that can be created from  $x$  by relocating a single customer. The customer can be moved to another position in its current route or to another route.

We define the size of the neighborhood  $N(\cdot)$  for a particular instance  $I$  as  $\max \{|N(x)| : x \in X(I)\}$ . Let  $\mathcal{I}(n)$  be the (possibly infinite) set of all instances of

size  $n$  of the problem under study. We can then define the size of a neighborhood as a function  $f(n)$  of the instance size  $n$ :  $f(n) = \max \{ |N(x)| : I \in \mathcal{I}(n), x \in X(I) \}$ .

The 2-opt neighborhood for the TSP and the relocate neighborhood for the CVRP have size  $f(n) = O(n^2)$  where  $n$  is the number of cities/customers.

### 13.1.3 Very Large-Scale Neighborhood Search

Ahuja et al. [2] define and survey the class of VLSN algorithms. A neighborhood search algorithm is considered as belonging to the class of VLSN algorithms if the neighborhood it searches grows exponentially with the instance size or if the neighborhood is simply too large to be searched explicitly in practice. Clearly, the class of VLSN algorithms is rather broad. Ahuja et al. [2] categorize VLSN into three classes: (1) variable-depth methods, (2) network flow-based improvement methods, and (3) methods based on restriction to subclasses solvable in polynomial time. We will describe the three classes in more detail in Sections 13.1.3.1, 13.1.3.2, and 13.1.3.3, respectively. Notice that although the concept of VLSN was not formalized until recently, algorithms based on similar principles have been used for decades.

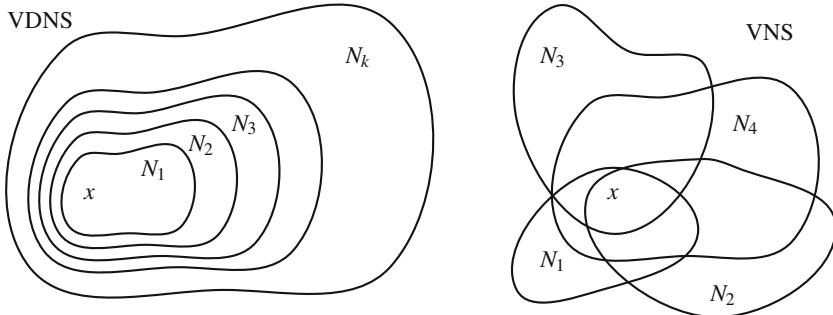
Intuitively, searching a very large neighborhood should lead to higher quality solutions than searching a small neighborhood. However, in practice, small neighborhoods can provide similar or superior solution quality if embedded in a meta-heuristic framework because they typically can be searched more quickly. Such behavior is reported in, e.g., [8] and [26]. This shows that VLSN algorithms are not “magic bullets.” However, for the right applications they provide excellent results.

As stated earlier, the focus of this chapter is a particular VLSN algorithm called LNS, which will be described in Section 13.2. The LNS heuristic does not fit well into any of the three categories defined in Ahuja et al. [2], but it definitely belongs to the class of VLSN algorithms as it searches a very large neighborhood.

#### 13.1.3.1 Variable-Depth Methods

Larger neighborhoods generally lead to local solutions of better quality, but the search is more time consuming. Hence, a natural idea is to gradually extend the size of the neighborhood, each time the search gets trapped in a local minimum.

*Variable-depth neighborhood search* (VDNS) methods search a parameterized family of still deeper neighborhoods  $N_1, N_2, \dots, N_k$  in a heuristic way. A typical example is the 1-exchange neighborhood  $N_1$  where one variable/position is changed. Similarly, the 2-exchange neighborhood  $N_2$  swaps the value of two variables/positions. In general the  $k$ -exchange neighborhood  $N_k$  changes  $k$  variables. Variable-depth search methods are techniques that search the  $k$ -exchange



**Fig. 13.2** Illustration of the neighborhoods used by VDNS and VNS. The current solution is marked  $x$ . VDNS typically operates on one type of neighborhood with variable depth, while VNS operates on structurally different neighborhoods  $N_1, \dots, N_k$ .

neighborhood partially, hence reducing the time used to search the neighborhood. See Figure 13.2 for an illustration of variable-depth neighborhoods.

One of the first applications of variable-depth search was the Lin–Kernighan heuristic [29] for solving the TSP. Briefly, the idea in the Lin–Kernighan heuristic is to replace as many as  $n$  edges when moving from a tour  $S$  to a tour  $T$ . In even steps of the algorithm an edge is inserted into the Hamiltonian path, while in odd steps an edge is deleted to restore a Hamiltonian path. From each Hamiltonian path a Hamiltonian cycle is implicitly constructed by joining the two end nodes. The choice for the edge to be added to the Hamiltonian path is made in a greedy way, maximizing the gain in the objective function. The Lin–Kernighan algorithm terminates when no improving tour can be constructed.

The basic idea in a VDNS heuristic is to make a sequence of local moves and to freeze all moved combinatorial objects to prevent the search from cycling. VDNS stops when no further local move is possible and returns a best found solution.

An extension of the Lin–Kernighan heuristic, called ejection chains, was proposed by Glover in [19]. An ejection chain is initiated by selecting a set of elements that will undergo a change of state. The result of this change leads to identifying a collection of other sets, with the property that the elements of at least one set must be “ejected from” their current states. State-change steps and ejection steps typically alternate. In some cases, a cascade of operations may be triggered leading to a domino effect.

Variable-depth and ejection-chain based algorithms have been applied to several problems, including the traveling salesman problem [17, 42], the vehicle routing problem with time windows [51], the generalized assignment problem [56], and nurse scheduling [14]. Ahuja et al. [2] give an excellent overview of earlier applications of the VDNS methods.

Frequently, VDNS methods are used in conjunction with other metaheuristic frameworks, like the filter-and-fan methods in Glover and Rego [20].

### 13.1.3.2 Network Flows Based Improvement Algorithms

This family of improvement algorithms use various network flow algorithms to search the neighborhood. In general they can be grouped in the following three, not necessarily distinct, categories: (i) minimum cost cycle methods, (ii) shortest path-based methods, and (iii) minimum cost assignment-based methods. In the following we give a short overview of the methods and refer to the survey of Ahuja et al. [2] for further details.

#### Neighborhoods Defined by Cycles

A *cyclic exchange neighborhood* consists of a sequence of elements being transferred among a family of subsets. Thompson [52] showed how to find an improving neighbor in the cyclic exchange neighborhood by finding a negative cost cycle in an hereto constructed improvement graph. Finding a negative cost subset-disjoint cycle in the improvement graph is NP-hard, but effective heuristics for searching the graph exist.

Thompson and Psaraftis [53] and Gendreau et al. [18] applied the cyclic neighborhood to solve the VRP. Ahuja et al. [1] used cyclic exchanges to solve the capacitated minimum spanning tree problem.

#### Neighborhoods Defined by Paths

*Path exchanges* is a generalization of the swap neighborhood. A large-scale neighborhood can be defined by aggregating an arbitrary number of so-called *independent* swap operations [2]. The best neighbor of a TSP tour for the compounded swap neighborhood can be found in  $O(n^2)$  time by solving a shortest path problem in an improvement graph constructed for this purpose.

For the one machine batching problem, Hurink [26] applies a special case of the compounded swap neighborhood where only adjacent pairs are allowed to switch. An improving neighbor can be found in  $O(n^2)$  time by solving a shortest path problem in the improvement graph.

Considering the single machine scheduling problem, Brueggemann and Hurink [7] presented an extension of the adjacent pairwise interchange neighborhood which can be searched in quadratic time by calculating a shortest path in an improvement graph.

#### Neighborhoods Defined by Assignments and Matching

The *assignment neighborhood* was first presented by Sarvanov and Doroshko [48] for the TSP. It is an exponential neighborhood structure defined by finding minimum cost assignments in an improvement graph.

For the TSP, the assignment neighborhood is based on the removal of  $k$  nodes, from which a bipartite graph is constructed. In this graph, the nodes on the left hand are the removed nodes, and the nodes on the right-hand side are the remaining nodes. The cost of each assignment is the cost of inserting a node between two existing nodes. Sarvanov and Doroshko [48] considered the case where  $k = n/2$  and  $n$  is even. Punnen [41] generalized this to arbitrary  $k$  and  $n$ .

Using the same concept, Franceschi et al. [13] obtained promising results for the distance-constrained CVRP, reporting 13 cases in which they were able to improve the best-known solution in the literature. The assignment problem, extended with additional capacity constraints, is solved as an *integer programming* (IP) problem. A further improvement is to identify removed nodes and insertion points in a clever way.

Brueggemann and Hurink [6] presented a neighborhood of exponential size for the problem of scheduling independent jobs on parallel machines minimizing the weighted average completion time. The neighborhood can be searched through matchings in a certain improvement neighborhood.

### 13.1.3.3 Efficiently Solvable Special Cases

Several NP-hard problems may be solved in polynomial time when restricting the problem topology or adding constraints to the original problem. Using these *special cases* as neighborhoods, one can often search exponentially large neighborhoods in polynomial time.

Ahuja et al. [2] describe a general method for turning a solution method for a restricted problem into a VLSN search technique. For each current solution  $x$  we create a well-structured instance of the problem which can be solved in polynomial time. The well-structured instance is solved, and a new solution  $x$  is found. Although the search method has a large potential, it is not always simple to construct an algorithm which turns  $x$  into a well-structured instance.

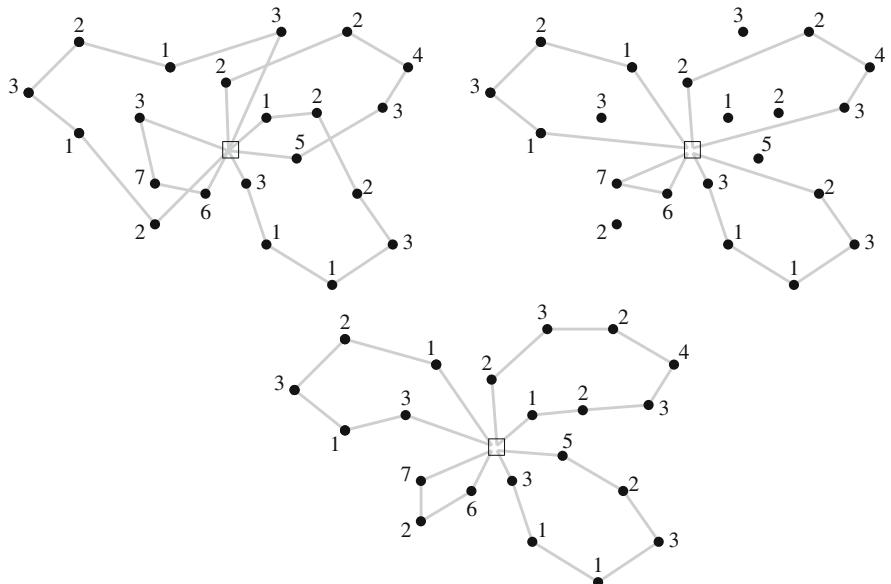
A *Halin graph* is a graph that may be obtained by considering a tree with no nodes of degree 2 in the plane and by joining the leaf nodes by a cycle so that the resulting graph is planar. A number of NP-hard problems can be solved efficiently (often in linear time) when restricted to a Halin graph. For instance, Cornuejols et al. [12] presented a linear-time algorithm for the TSP defined on a Halin graph. Phillips et al. [38] presented similar results for the bottleneck TSP, and Winter [55] for the Steiner problem.

Brueggemann and Hurink [7] also present a neighborhood for the single machine scheduling problem which is based on a dominance rule for sequences. A relaxation of the dominance rule can be solved in polynomial time by using a *shortest processing time first rule*.

## 13.2 Large Neighborhood Search

The *large neighborhood search* (LNS) metaheuristic was proposed by Shaw [50]. Most neighborhood search algorithms explicitly define the neighborhood like the relocate neighborhood described in Section 13.1.2. In the LNS metaheuristic the neighborhood is defined implicitly by a *destroy* and a *repair* method. A destroy method destracts part of the current solution while a repair method rebuilds the destroyed solution. The destroy method typically contains an element of stochasticity such that different parts of the solution are destroyed in every invocation of the method. The neighborhood  $N(x)$  of a solution  $x$  is then defined as the set of solutions that can be reached by first applying the destroy method and then the repair method.

To illustrate the destroy and repair concepts, consider the CVRP. A destroy method for the CVRP could remove, say, 15% of the customers in the current solution, shortcircuiting the routes where customers have been removed. A very simple destroy method would select the customers to remove at random. A repair method could rebuild the solution by inserting removed customers, using a greedy heuristic. Such a heuristic could simply scan all free customers, insert the one whose insertion cost is the lowest, and repeat inserting until all customers have been inserted. The destroy and repair step is illustrated in Figure 13.3.



**Fig. 13.3** Destroy and repair example. The *top left* figure shows a CVRP solution before the destroy operation. The *top right* figure shows the solution after a destroy operation that removed six customers (now disconnected from the routes). The *bottom* figure shows the solution after the repair operation has reinserted the customers.

**Algorithm 1** Large neighborhood search

---

```

1: input: a feasible solution  $x$ 
2:  $x^b = x$ ;
3: repeat
4:    $x' = r(d(x))$ ;
5:   if accept( $x'$ ,  $x$ ) then
6:      $x = x'$ ;
7:   end if
8:   if  $c(x') < c(x^b)$  then
9:      $x^b = x'$ ;
10:  end if
11: until stop criterion is met
12: return  $x^b$ 
```

---

Since the destroy method can destruct a large part of the solution, the neighborhood contains a large amount of solutions which explains the name of the heuristic. Consider, for example, a CVRP instance with 100 customers. There are  $C(100, 15) = 100!/(15! \times 85!) = 2.5 \times 10^{17}$  different ways to select the customers to be removed if the percentage or degree of destruction of the solution is 15%. For each removal choice there are many ways of repairing the solution, but different removal choices can of course result in the same solution after the repair.

We now present the LNS heuristic in more details. Pseudocode for the heuristic is shown in Algorithm 1. Three variables are maintained by the algorithm. The variable  $x^b$  is the best solution observed during the search,  $x$  is the current solution, and  $x'$  is a temporary solution that can be discarded or promoted to the status of current solution. The function  $d(\cdot)$  is the destroy method while  $r(\cdot)$  is the repair method. More specifically,  $d(x)$  returns a copy of  $x$  that is partly destroyed. Applying  $r(\cdot)$  to a partly destroyed solution repairs it, that is, it returns a feasible solution built from the destroyed one. In line 2 the global best solution is initialized. In line 4 the heuristic first applies the destroy method and then the repair method to obtain a new solution  $x'$ . In line 5 the new solution is evaluated, and the heuristic determines whether this solution should become the new current solution (line 6) or whether it should be rejected. The *accept* function can be implemented in different ways. The simplest choice is to only accept improving solutions. Line 8 checks whether the new solution is better than the best known solution. Here  $c(x)$  denotes the objective value of solution  $x$ . The best solution is updated in line 9 if necessary. In line 11 the termination condition is checked. It is up to the implementer to choose the termination criterion, but a limit on the number of iterations or a time limit would be typical choices. In line 12 the best solution found is returned. From the pseudocode it can be noticed that the LNS metaheuristic does not search the entire neighborhood of a solution, but merely samples this neighborhood.

The main idea behind the LNS heuristic is that the large neighborhood allows the heuristic to navigate in the solution space easily, even if the instance is tightly constrained. This is to be opposed to a small neighborhood which can make the navigation in the solution space much harder.

In the original LNS paper [50] the accept method only allowed improving solutions. Later papers like [44, 49] have used an acceptance criteria borrowed from simulated annealing. With such an acceptance criteria, the temporary solution  $x'$  is always accepted if  $c(x') \leq c(x)$ , and accepted with probability  $e^{-(c(x') - c(x))/T}$  if  $c(x) < c(x')$ . Here  $T > 0$  is the current *temperature*. The temperature is initialized at  $T_0 > 0$  and is decreased gradually, for example, by performing the update  $T_{\text{new}} = \alpha T_{\text{old}}$  at each iteration, where  $0 < \alpha < 1$  is a parameter. The idea is that  $T$  is relatively high initially, thus allowing deteriorating solutions to be accepted. As the search progresses  $T$  decreases and toward the end of the search only a few or no deteriorating solutions will be accepted. If such an acceptance criteria is employed, the LNS heuristic can be viewed as a standard simulated annealing heuristic with a complex neighborhood definition.

The destroy method is an important part of the LNS heuristic. The most important choice when implementing the destroy method is the *degree of destruction*: if only a small part of the solution is destroyed then the heuristic may have trouble exploring the search space as the effect of a large neighborhood is lost. If a very large part of the solution is destroyed then the LNS heuristic almost degrades into repeated re-optimization. This can be time consuming or yield poor-quality solutions dependent on how the partial solution is repaired. Shaw [50] proposed to gradually increase the degree of destruction, while Ropke and Pisinger [44] choose the degree of destruction randomly in each iteration by choosing the degree from a specific range dependent on the instance size. The destroy method must also be chosen such that the entire search space can be reached, or at least the interesting part of the search space where the global optimum is expected to be found. Therefore, it cannot focus on always destroying a particular component of the solution but must make it possible to destroy every part of the solution.

The implementer of an LNS heuristic has much freedom in choosing the repair method. A first decision is whether the repair method should be optimal in the sense that the best possible full solution is constructed from the partial solution, or whether it should be a heuristic assuming that one is satisfied with a good solution constructed from the partial solution. An optimal repair operation will be slower than a heuristic one, but may potentially lead to high-quality solutions in a few iterations. However, from a diversification point of view, an optimal repair operation may not be attractive: only improving or identical-cost solutions will be produced and it can be difficult to leave valleys in the search space unless a large part of the solution is destroyed in each iteration. The framework also enables the implementer to choose if the repair method should be hand-coded or if a general-purpose solver like a mixed integer programming (MIP) or constraint programming solver should be invoked.

It is worth observing that the LNS heuristic typically alternates between an infeasible solution and a feasible solution: the destroy operation creates an infeasible solution which is brought back into feasible form by the repair heuristic. Alternately the destroy and repair operations can be viewed as fix/optimize operations: the *fix* method (corresponding to the destroy method) fixes part of the solution at its current value while the rest remains free; the *optimize* method (corresponding to the repair

method) attempts to improve the current solution while respecting the fixed values. Such an interpretation of the heuristic may be more natural if the repair method is implemented using MIP or constraint programming solvers.

The concept of destroy and repair methods in large neighborhood lends itself best to problems which naturally can be decomposed into a master problem covering a number of tasks to be carried out and a set of subproblems which need to satisfy some constraints. In this case, the destroy methods remove some tasks from the current solution, and the repair methods reinsert the tasks. Hence, problems where Dantzig Wolfe decomposition has been used with success are good candidates for LNS heuristics.

Before closing this section, it should be mentioned that a framework, very similar to the LNS, has been proposed under the name *ruin and recreate* by Schrimpf et al. [49].

### 13.2.1 Adaptive Large Neighborhood Search

The adaptive large neighborhood search (ALNS) heuristic was proposed in [44] and extends the LNS heuristic by allowing multiple destroy and repair methods to be used within the same search. Each destroy/repair method is assigned a weight that controls how often the particular method is attempted during the search. The weights are adjusted dynamically as the search progresses so that the heuristic adapts to the instance at hand and to the state of the search.

Using neighborhood search terminology, one can say that the ALNS extends the LNS by allowing multiple neighborhoods within the same search. The choice of neighborhood to use is controlled dynamically using recorded performance of the neighborhoods.

---

#### Algorithm 2 Adaptive large neighborhood search

---

```

1: input: a feasible solution  $x$ 
2:  $x^b = x; \rho^- = (1, \dots, 1); \rho^+ = (1, \dots, 1);$ 
3: repeat
4:   select destroy and repair methods  $d \in \Omega^-$  and  $r \in \Omega^+$  using  $\rho^-$  and  $\rho^+$ ;
5:    $x' = r(d(x));$ 
6:   if accept( $x', x$ ) then
7:      $x = x';$ 
8:   end if
9:   if  $c(x') < c(x^b)$  then
10:     $x^b = x';$ 
11:   end if
12:   update  $\rho^-$  and  $\rho^+;$ 
13: until stop criterion is met
14: return  $x^b$ 

```

---

A pseudocode for the ALNS heuristic is shown in Algorithm 2. Compared to the LNS pseudocode in Algorithm 1, the following have changed. Lines 4 and 12 have

been added and line 2 has been modified. The sets of destroy and repair methods are denoted by  $\Omega^-$  and  $\Omega^+$ , respectively. Two new variables are introduced in line 2:  $\rho^- \in \mathbb{R}^{|\Omega^-|}$  and  $\rho^+ \in \mathbb{R}^{|\Omega^+|}$ , to store the weight of each destroy and repair method, respectively. Initially all methods have the same weight. In line 4 the weight vectors  $\rho^-$  and  $\rho^+$  are used to select the destroy and repair methods using a *roulette wheel principle*. The algorithm calculates the probability  $\phi_j^-$  of choosing the  $j$ th destroy method as follows:

$$\phi_j^- = \frac{\rho_j^-}{\sum_{k=1}^{|\Omega^-|} \rho_k^-},$$

and the probabilities for choosing the repair methods are determined in the same way.

The weights are adjusted dynamically, based on the recorded performance of each destroy and repair method. This takes place in line 12: when an iteration of the ALNS heuristic is completed, a score  $\psi$  for the destroy and repair method used in the iteration is computed using the formula

$$\psi = \max \begin{cases} \omega_1 & \text{if the new solution is a new global best,} \\ \omega_2 & \text{if the new solution is better than the current one,} \\ \omega_3 & \text{if the new solution is accepted,} \\ \omega_4 & \text{if the new solution is rejected,} \end{cases} \quad (13.1)$$

where  $\omega_1, \omega_2, \omega_3$ , and  $\omega_4$  are parameters. A high  $\psi$  value corresponds to a successful method. We would normally have  $\omega_1 \geq \omega_2 \geq \omega_3 \geq \omega_4 \geq 0$ .

Let  $a$  and  $b$  be the indices of the destroy and repair methods that were used in the last iteration of the algorithm, respectively. The components corresponding to the selected destroy and repair methods in the  $\rho^-$  and  $\rho^+$  vectors are updated using the equations

$$\rho_a^- = \lambda \rho_a^- + (1 - \lambda) \psi, \quad \rho_b^+ = \lambda \rho_b^+ + (1 - \lambda) \psi, \quad (13.2)$$

where  $\lambda \in [0, 1]$  is the *decay* parameter that controls how sensitive the weights are to changes in the performance of the destroy and repair methods. Note that the weights that are not used at the current iteration remain unchanged. The aim of the adaptive weight adjustment is to select weights that work well for the instance being solved. We encourage heuristics that bring the search forward; these are the ones rewarded with the  $\omega_1, \omega_2$ , and  $\omega_3$  parameters in Equation (13.1). We discourage heuristics that lead to many rejected solutions as an iteration resulting in a rejected solution is a wasted iteration, roughly speaking. This is achieved by assigning a low value to  $\omega_4$ .

The ALNS heuristic described so far is prone to favor complex repair methods that more often reach high-quality solutions compared to simpler repair methods. This is fine if the complex and simple repair methods are equally time consuming, but that may not be the case. If some methods are significantly slower than others, one may normalize the score  $\psi$  of a method with a measure of the time consumption of the corresponding heuristic. This ensures a proper trade-off between time consumption and solution quality.

### 13.2.2 Designing an ALNS Algorithm

The considerations for selecting destroy and repair methods in the LNS heuristic, mentioned earlier, also hold for an ALNS heuristic. However, the ALNS framework gives some extra freedom because multiple destroy/repair methods are allowed. In the pure LNS heuristic we have to select a destroy and repair method that is expected to work well for a wide range of instances. In an ALNS heuristic we can afford to include destroy/repair methods that only are suitable in some cases—the adaptive weight adjustment will ensure that these heuristics seldom are used in instances where they are ineffective. Therefore, the selection of destroy and repair methods can be turned into a search for methods that are good at either diversification or intensification.

Below we will discuss some typical destroy and repair methods. In the discussion we will assume that our solution is represented by a set of decision *variables*. The term variables should be understood in a rather abstract way.

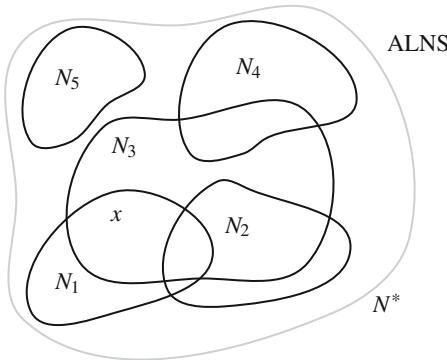
Diversification and intensification for the destroy methods can be accomplished as follows: to diversify the search, one may randomly select the parts of the solution that should be destroyed (*random destroy* method). To intensify the search one may try to remove  $q$  “critical” variables, i.e., variables having a large cost or variables spoiling the current structure of the solution (e.g., edges crossing each other in an Euclidean TSP). This is known as *worst destroy* or *critical destroy*.

One may also choose a number of related variables that are easy to interchange while maintaining feasibility of the solution. This *related destroy* neighborhood was introduced by Shaw [50]. For the CVRP one can define a relatedness measure between each pair of customers. The measure could simply be the distance between the customers and it could include customer demand as well (customers with similar demand are considered related). A related destroy would select a set of customers that have a high mutual relatedness measure. The idea is that it should be easy to exchange similar customers.

Finally, one may use *history-based destroy* where the  $q$  variables are chosen according to some historical information as presented in [39]. The historical information could, for example, count how often setting a given variable (or set of variables) to a specific value leads to a bad solution. One may then try to remove variables that are currently assigned to an improper value, based on the historical information.

The repair methods ( $\Omega^+$ ) are often based on concrete well-performing heuristics for the given problem. These heuristics can make use of variants of the greedy paradigm, e.g., performing the locally best choice in each step or performing the least bad choice in each step. The repair methods can also be based on approximation algorithms or exact algorithms. Exact algorithms can be relaxed to obtain faster solution times at the cost of solution quality. Some examples are presented in [4] and [50]. Time-consuming and fast repair methods can be mixed by penalizing the time-consuming methods as described earlier.

Figure 13.4 illustrates, in an abstract way, the many neighborhoods in an ALNS heuristic. Each neighborhood on the figure can be considered as a unique combination of a destroy and repair method.



**Fig. 13.4** Illustration of neighborhoods used by ALNS. The current solution is marked with  $x$ . ALNS operates on structurally different neighborhoods  $N_1, \dots, N_k$  defined by the corresponding search heuristics. All neighborhoods  $N_1, \dots, N_k$  in ALNS are a subset of the neighborhood  $N^*$  defined by modifying  $q$  variables, where  $q$  is a measure of the maximum degree of destruction.

In traditional local search heuristics the diversification is controlled implicitly by the local search paradigm (accept ratio, tabu list, etc.). For the (A)LNS heuristic this may not be enough. It can often be advantageous to use noising or randomization in both the destroy and repair methods to obtain a proper diversification. This helps avoiding stagnating search processes where the destroy and repair methods keep performing the same modifications to a solution.

Some optimization problems can be split into a number of subproblems, where each subproblem can be solved individually. Such problems include the *bin-packing problem* in which a number of bins are to be filled or the *vehicle routing problem* in which a number of routes are to be constructed. For such problems one should decide whether the subproblems should be solved one by one (*sequential heuristics*) or all subproblems should be solved at the same time (*parallel heuristics*). Sequential heuristics are easier to implement but may have the disadvantage that the last subproblem solved is left with variables that do not fit well together. This is to some extent avoided in parallel heuristics.

A natural extension to the ALNS framework is to have *coupled neighborhoods*. In principle, one may, for each destroy method  $d_i$ , define a subset  $K_i \subseteq \Omega^+$  of repair neighborhoods that can be used with  $d_i$ . The roulette wheel selection of repair neighborhoods will then only choose a neighborhood in  $K_i$  if  $d_i$  was chosen.

As a special case, one may have  $K_i = \emptyset$  meaning that the neighborhood  $d_i$  takes care of both the destroy and repair steps. One could use an ordinary local search heuristic to compete with the other destroy and repair neighborhoods, ensuring that

a thorough investigation of the solution space close to the current solution is made from time to time.

For some problems it may be sufficient to have a number of destroy and repair heuristics that are selected randomly with equal probability, that is without the adaptive layer. Such heuristics share the robustness of the ALNS heuristics, while having considerably fewer parameters to calibrate.

### 13.2.3 Properties of the ALNS Framework

The ALNS framework has several advantages. For most optimization problems we already know a number of well-performing heuristics which can form the core of an ALNS algorithm. Due to the large neighborhoods and diversity of the neighborhoods, the ALNS algorithm will explore large parts of the solution space in a structured way. The resulting algorithm becomes very robust, as it is able to adapt to various characteristics of the individual instances, and seldom is trapped in local optima.

The calibration of the ALNS algorithm is quite limited as the adaptive layer automatically adjusts the influence of each neighborhood used. It is still necessary to calibrate the individual sub-heuristics used for searching the destroy and repair neighborhoods, but one may calibrate these individually or even use the parameters used in existing algorithms.

In the design of most local search algorithms the researcher has to choose between a number of possible neighborhoods. In ALNS the question is not “either-or” but rather “both-and.” As a matter of fact, our experience is that the more (reasonable) neighborhoods the ALNS heuristic makes use of, the better it performs [39, 45].

The ALNS framework is not the only one to make use of several neighborhoods in an LNS heuristic. Rousseau et al. [47] use two LNS neighborhoods for the *vehicle routing problem with time windows* (VRPTW): one removing customers and another removing arcs. They propose a *variable neighborhood descent* (VND) where one neighborhood is used until one is “sufficiently sure” that the search is trapped in a local minimum in which case the search switches to the other neighborhood. When the second neighborhood runs out of steam the first neighborhood is used again and so on.

Perron [35] used an adaptive technique to select repair methods from a portfolio by assigning weights to the repair methods based on their performance like in the ALNS. Laborie and Godard [28] propose a framework very similar to ALNS, the difference being that their framework also dynamically adjusts the parameters of the individual destroy and repair methods. The ALNS framework described in this section assumes that those parameters are fixed in advance. Palpant et al. [34] only use one destroy and repair method but propose a method for dynamically adjusting the

scope of the destroy operation in order to find the neighborhood size that allows the repair operation to be completed within reasonable time. The authors use complex, time-consuming repair methods.

When implementing an LNS or ALNS heuristic one can choose which “outer” metaheuristic to use to guide the search (if any). Some simply use a descent approach (e.g., [4, 50]), some use iterated local search (e.g., [34, 47]), and others use simulated annealing (e.g., [44, 49]). It is our experience that even a simple outer metaheuristic improves upon a pure descent approach.

The ALNS is related to the VNS metaheuristics [25, 31] in the sense that both heuristics search multiple neighborhoods. Since a local optimum with respect to one neighborhood is not necessarily a local optimum with respect to another neighborhood, changing neighborhoods in the search is a way of diversifying the search.

VNS makes use of a parameterized family of neighborhoods, typically obtained by using a given neighborhood with variable depth. When the algorithm reaches a local minimum using one of the neighborhoods, it proceeds with a larger neighborhood from the parameterized family. When the VNS algorithm gets out of the local minimum it proceeds with the smaller neighborhood. On the contrary, ALNS operates on a predefined set of large neighborhoods corresponding to the destroy (removal) and repair (insertion) heuristics. The neighborhoods are not necessarily well defined in a formal mathematical sense—they are rather defined by the corresponding heuristic algorithm.

A major challenge in designing a good VNS algorithm is to decide in what order the neighborhoods should be searched. A natural strategy is to order the neighborhoods according to the complexity of searching them, such that one starts with the least complex neighborhoods, and gradually include the more expensive. ALNS take a different approach by using roulette wheel selection with adaptive probabilities to decide which neighborhoods to use.

Another related concept is that of *hyper-heuristics*. Ross [46] describes hyper-heuristics as *heuristics to choose heuristics*, that is, algorithms where a master heuristic is choosing between several sub-ordinate heuristics. Therefore, the ALNS heuristic can be seen as a kind of hyper-heuristic: the adaptive component is choosing from the set of destroy and repair methods (which usually are heuristics).

A few examples of parallel processing LNS/ALNS implementations exist in the literature. Perron and Shaw [36] describe a parallel LNS heuristic that is applied to a network design problem and Ropke [43] describes a framework for implementing parallel ALNS heuristics. The framework is tested on the CVRP and TSP with pickup and delivery.

### 13.3 Applications of LNS

So far the LNS heuristic has been most successful within the areas of routing and scheduling problems. In this section we review the main results for these two problem classes.

### 13.3.1 Routing Problems

In this section we survey applications of LNS heuristics to variants of the TSP and VRP. There are many examples of applications of LNS to VRP variants, starting with Shaw's [50] definition of the LNS heuristic. Many of the heuristics have been successful and have provided state-of-the-art results at the time of publication. An incomplete list of papers describing the application of LNS to VRP variants, in particular the VRPTW, is [4, 5, 10, 13, 22, 23, 30, 39, 40, 44, 45, 47, 49, 50]. Reference [13] does not make the connection to the LNS heuristic, but the approach described fits nicely in the LNS framework.

Bent and Hentenryck [4] describe an LNS heuristic for the VRPTW. In the VRPTW the most common objective is to minimize first the number of vehicles and, for the same number of vehicles, to minimize the total route lengths. Bent and Hentenryck [4] propose to solve the problem in a two-stage approach. In the first stage the number of routes is minimized by a simulated annealing algorithm that uses traditional, small neighborhoods. In the second stage the total route lengths are minimized with an LNS heuristic. The destroy method uses the relatedness principle described in Section 13.2.2. The size of the neighborhood is gradually increased, starting out by only removing one customer and by steadily increasing the number of customers to remove as the search progresses. At regular intervals, the number of customers to remove is reset to one and the neighborhood size increase starts over. The repair method is implemented using a truncated branch-and-bound algorithm. The LNS algorithm only accepts improving solutions. The results obtained can be considered as state of the art at the time of publication. A similar algorithm was also proposed by the same authors [5] for the *pickup and delivery problem with time windows* (PDPTW).

Ropke and Pisinger [44] introduce the ALNS extension of the LNS previously described in Section 13.2.1. The algorithm is applied to the PDPTW. Differences with the method in [4] are as follows: (i) several different destroy/repair methods are used, (ii) fast, greedy heuristics are used as repair methods, (iii) the size of the neighborhood varies from iteration to iteration (the number of customers to remove is chosen randomly from a predefined interval), and (iv) a simulated annealing acceptance criterion is used. The heuristic has obtained state-of-the-art results for the PDPTW. In subsequent papers [39, 45] it is shown that many VRP variants (including the CVRP and VRPTW) can be transformed into a PDPTW and solved using an improved version of the ALNS heuristic from [44]. For most of the tested VRP variants the ALNS heuristic must be considered to be on par with or better than competing heuristics at the time of publication.

Prescott-Gagnon et al. [40] present an LNS heuristic for the VRPTW with an advanced repair operator that solves a restricted VRPTW through a heuristic branch-and-price algorithm. Four destroy methods are used and are chosen based on performance as in [44]. Overall, the heuristic reaches better solutions than previous LNS approaches, probably due to the advanced repair operator.

It should be mentioned that the VRPTW is one of the most studied problem class from a metaheuristic point of view. We estimate that more than 100 papers

have been published on the subject. It is therefore remarkable that LNS heuristics, as a rather young class of heuristics, have been able to be in the forefront of the development in recent years. We should also mention that the best solutions for the VRPTW are currently found using a non-LNS heuristic proposed by Nagata and Bräysy [33].

We are only aware of a few applications of LNS to TSP variants [15, 37, 49]. An explanation for the lower number of applications could be that the LNS heuristic is inherently better suited for VRP variants than for TSP variants because of the partitioning element present in VRPs.

### **13.3.2 Scheduling Problems**

LNS and ALNS lend themselves well to scheduling problems due to the tightly constrained nature of the problems. Laborie and Godard [28] present results for a self-adapting large neighborhood search, applied to single-mode scheduling problems. Godard, Laborie, and Nuijten [21] present a randomized large neighborhood search for cumulative scheduling. Carchrae and Beck [9] present results for job-shop scheduling problems. Cordeau et al. [11] present an ALNS heuristic for solving a technician scheduling problem. Instances with hundreds of tasks and technicians are solved in less than 15 min. Muller [32] recently presented an ALNS algorithm for the resource-constrained project scheduling problem. The computational results show that the algorithm is among the three best on the well-known PSPLIB benchmark instances.

## **13.4 Conclusion**

In this chapter we have reviewed the LNS heuristic and its extensions and we have briefly explained the central concepts in VLSN. Both are interesting concepts and we hope that these topics will be subject to increased research in the future. We believe that we have yet to see the full potential from both LNS and VLSN algorithms in general.

One of the key benefits of the LNS heuristic is that a heuristic can be quickly put together from existing components: an existing construction heuristic or exact method can be turned into a repair heuristic and a destroy method based on random selection is easy to implement. Therefore, we see a potential for using simple LNS heuristics for benchmark purposes when developing more sophisticated methods.

We do not have any illusions about LNS being superior to all other metaheuristics. We believe that LNS heuristics, in general, work well when the problem considered involves some kind of partitioning decision, as in, e.g., VRP, bin-packing, or generalized assignment problems. Such structure seems to be well suited for the destroy/repair operations. For problems that do not exhibit this structure it is diffi-

cult to predict the performance of the LNS heuristic and other metaheuristics may be better suited.

Large neighborhoods are no guarantee for finding better solutions. Increased complexity of the neighborhood search means that fewer iterations can be performed by a local search algorithm. Gutin and Karapetyan [24] experimentally compare a number of small and large neighborhoods for the multidimensional assignment problem, including various combinations of them. It is demonstrated that some combinations of both small and large neighborhoods provide the best results. This could indicate that hybrid neighborhoods may be a promising direction for future research.

## References

1. Ahuja, R.K., Ergun, Ö., Orlin, J.B., Punnen, A.P.: A survey of very large-scale neighborhood search techniques. *Discrete Appl. Math.* **123**, 75–102 (2002)
2. Ahuja, R.K., Orlin, J.B., Sharma, D.: New neighborhood search structures for the capacitated minimum spanning tree problem. Technical Report 99–2, 1999
3. Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J.: *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, Princeton, NJ (2006)
4. Bent, R., Van Hentenryck, P.: A two-stage hybrid local search for the vehicle routing problem with time windows. *Transport. Sci.* **38**(4), 515–530 (2004)
5. Bent, R., Van Hentenryck, P.: A two-stage hybrid algorithm for pickup and delivery vehicle routing problem with time windows. *Comput. Oper. Res.* **33**(4), 875–893 (2006)
6. Brueggemann, T., Hurink, J.L.: Matching based exponential neighborhoods for parallel machine scheduling. Technical Report Memorandum No. 1773, (2005)
7. Brueggemann, T., Hurink, J.L.: Two exponential neighborhoods for single machine scheduling. Technical Report Memorandum No. 1776, 2005
8. Brueggemann, T., Hurink, J.: Two very large-scale neighborhoods for single machine scheduling. *OR Spectr.* **29**, 513–533 (2007)
9. Carchrae, T., Beck, J.C.: Cost-based large neighborhood search. In *Workshop on the Combination of Metaheuristic and Local Search with Constraint Programming Techniques*, 2005
10. Caseau, Y., Laburthe, F., Silverstein, G.: A meta-heuristic factory for vehicle routing problems. *Lect. Notes Comput. Sci.* **1713**, 144–159 (1999)
11. Cordeau, J.-F., Laporte, G., Pasin, F., Ropke, S.: Scheduling technicians and tasks in a telecommunications company. *J. Scheduling* (2010) Forthcoming
12. Cornuejols, G., Naddef, D., Pulleyblank, W.R.: Halin graphs and the traveling salesman problem. *Math. Program.* **26**, 287–294 (1983)
13. De Franceschi, R., Fischetti, M., Toth, P.: A new ILP-based refinement heuristic for vehicle routing problems. *Math. Program.* **105**, 471–499 (2006)
14. Dowsland, K.A.: Nurse scheduling with tabu search and strategic oscillation. *Eur. J. Oper. Res.* **106**, 393–407 (1998)
15. Dumitrescu, I., Ropke, S., Cordeau, J.-F., Laporte, G.: The traveling salesman problem with pickup and delivery: polyhedral results and a branch-and-cut algorithm. *Math. Program.* **121**, 269–305 (2009)
16. Flood, M.M.: The traveling salesman problem. *Oper. Res.* **4**(1), 61–75 (1956)
17. Gamboa, D., Osterman, C., Rego, C., Glover, F.: An experimental evaluation of ejection chain algorithms for the traveling salesman problem. Technical report, School of Business Administration, University of Mississippi, 2006

18. Gendreau, M., Guertin, F., Potvin, J.-Y., Seguin, R.: Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries. Technical Report 98-10, 1998
19. Glover, F.: Ejection chains, reference structures, and alternating path algorithms for the traveling salesman problem. Technical Report, 1992
20. Glover, F., Rego, C.: Ejection chain and filter-and-fan methods in combinatorial optimization. *4OR: A Q. J. Oper. Res.* **4**, 263–296 (2006)
21. Godard, D., Laborie, P., Nuijten, W.: Randomized large neighborhood search for cumulative scheduling. In: Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS 2005), pp. 81–89, Monterey, CA, USA, 5–10 June 2005
22. Goel, A.: Vehicle scheduling and routing with driver's working hours. *Transport. Sci.* (2009) Forthcoming
23. Goel, A., Gruhn, V.: A general vehicle routing problem. *Eur. J. Oper. Res.* **191**(3), 650–660 (2008)
24. Gutin, G., Karapetyan, D.: Local search heuristics for the multidimensional assignment problem. In: Proceedings of the Columbic Festschrift, vol. 5420, pp. 100–115 (2009)
25. Hansen, P., Mladenović, N.: Variable neighborhood search: Principles and applications. *Eur. J. Oper. Res.* **130**, 449–467 (2001)
26. Hurink, J.: An exponential neighborhood for a one machine batching problem. *OR-Spektr.* **21**, 461–476 (1999)
27. Kilby, P., Prosser, P., Shaw, P.: Guided local search for the vehicle routing problem. In: Proceedings of the 2nd International Conference on Metaheuristics, Sophia-Antipolis, France, July 1997
28. Laborie, P., Godard, D.: Self-adapting large neighborhood search: Application to single-mode scheduling problems. Technical Report TR-07-001, ILOG, 2007
29. Lin, S., Kernighan, B.: An effective heuristic algorithm for the traveling salesman problem. *Oper. Res.* **21**, 498–516 (1973)
30. Mester, D., Bräysy, O.: Active guided evolution strategies for large-scale vehicle routing problems with time windows. *Comput. Oper. Res.* **32**, 1593–1614 (2005)
31. Mladenovic, N., Hansen, P.: Variable neighborhood search. *Comput. Oper. Res.* **24**, 1097–1100 (1997)
32. Muller, L.F.: An adaptive large neighborhood search algorithm for the resource-constrained project scheduling problem. In: Proceedings of MIC 2009: The VIII Metaheuristics International Conference, Hamburg, Germany
33. Nagata, Y., Bräysy, O.: A powerful route minimization heuristic for the vehicle routing problem with time windows. *Oper. Res. Lett.* **37**, 333–338 (2009)
34. Palpant, M., Artigues, C.C., Michelon, P.: LSSPER: Solving the resource-constrained project scheduling problem with large neighbourhood search. *Ann. Oper. Res.*, **131**, 237–257 (2004)
35. Perron, L.: Fast restart policies and large neighborhood search. In: Proceedings of CP-AI-OR'2003, Montreal, Canada 2003
36. Perron, L., Shaw, P.: Parallel large neighborhood search. In: Proceedings of RenPar'15, La Colle sur Loup, France 2003
37. Petersen, H.L., Madsen, O.B.G.: The double travelling salesman problem with multiple stacks – formulation and heuristic solution approaches. *Eur. J. Oper. Res.* **198**(1), 139–147 (2009)
38. Phillips, J.M., Punnen, A.P., Kabadi, S.N.: A linear time algorithm for the bottleneck traveling salesman problem on a Halin graph. *Inf. Process. Lett.*, **67**, 105–110 (1998)
39. Pisinger, D., Ropke, S.: A general heuristic for vehicle routing problems. *Comput. Oper. Res.* **34**(8), 2403–2435 (2007)
40. Prescott-Gagnon, E., Desaulniers, G., Rousseau, L.-M.: A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows. Technical Report G-2007-67, GERAD, Montreal, QC, Canada, September 2007
41. Punnen, A.P.: The traveling salesman problem: New polynomial approximation algorithms and domination analysis. *J. Inf. Optimization Sci.*, **22**, 191–206 (2001)

42. Rego, C., Gamboa, D., Glover, F.: Data structures and ejection chains for solving large scale traveling salesman problems. *Eur. J. Oper. Res.* **160**, 154–171 (2006)
43. Ropke, S.: Parallel large neighborhood search – a software framework. In: Proceedings of MIC 2009: The VIII Metaheuristics International Conference. Hamburg, Germany
44. Ropke, S., Pisinger, D.: An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transport. Sci.*, **40**(4), 455–472 (2006)
45. Ropke, S., Pisinger, D.: A unified heuristic for a large class of vehicle routing problems with backhauls. *Eur. J. Oper. Res.* **171**, 750–775 (2006)
46. Ross, P.: Hyper-heuristics. In: Burke, E.K., Kendall, G. (eds.) *Introductory Tutorials in Optimisation, Decision Support and Search Methodology*, Chapter 17, pp. 529–556. Springer, New York, NY (2005)
47. Rousseau, L.-M., Gendreau, M., Pesant, G.: Using constraint-based operators to solve the vehicle routing problem with time windows. *J. Heuristics* **8**, 43–58 (2002)
48. Sarvanov, V.I., Doroshko, N.N.: Approximate solution of the traveling salesman problem by a local algorithm with scanning neighborhoods of factorial cardinality in cubic time. In: Software: Algorithms and Programs, no. **31**, pp. 11–13. Mathematical Institute of Belorussian Academy of Science, Minsk (1981) (in Russian)
49. Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H., Dueck, G.: Record breaking optimization results using the ruin and recreate principle. *J. Comput. Phys.*, **159**(2), 139–171 (2000)
50. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: CP-98 (Fourth International Conference on Principles and Practice of Constraint Programming). *Lect. Notes Comput. Sci.*, **1520**, 417–431 (1998)
51. Sontrop, H., van der Horn, P., Uetz, M.: Fast ejection chain algorithms for vehicle routing with time windows. *Lect. Notes Comput. Sci.* **3636**, 78–89 (2005)
52. Thompson, P.M.: Local search algorithms for vehicle routing and other combinatorial problems. PhD Thesis, Operations Research Center, MIT (1988)
53. Thompson, P.M., Psaraftis, H.N.: Cyclic transfer algorithms for multivehicle routing and scheduling problems. *Oper. Res.*, **41** (1993)
54. Toth, P., Vigo, D.: An overview of vehicle routing problems. In: Toth, P., Vigo, D. (eds.) *The Vehicle Routing Problem*, vol. 9 of SIAM Monographs on Discrete Mathematics and Applications, Chapter 1, pp. 1–26. SIAM, Philadelphia, PA (2002)
55. Winter, P.: Steiner problem in Halin networks. *Discrete Appl. Math.*, **17**, 281–294 (1987)
56. Yagiura, M., Ibaraki, T., Glover, F.: A path relinking approach with ejection chains for the generalized assignment problem. *Eur. J. Oper. Res.*, **169**, 548–569 (2006)



# Chapter 14

## Artificial Immune Systems

Julie Greensmith, Amanda Whitbrook and Uwe Aickelin

**Abstract** The human immune system has numerous properties that make it ripe for exploitation in the computational domain, such as robustness and fault tolerance, and many different algorithms, collectively termed Artificial Immune Systems (AIS), have been inspired by it. Two generations of AIS are currently in use, with the first generation relying on simplified immune models and the second generation utilising interdisciplinary collaboration to develop a deeper understanding of the immune system and hence produce more complex models. Both generations of algorithms have been successfully applied to a variety of problems, including anomaly detection, pattern recognition, optimisation and robotics. In this chapter an overview of AIS is presented, its evolution is discussed, and it is shown that the diversification of the field is linked to the diversity of the immune system itself, leading to a number of algorithms as opposed to one archetypal system. Two case studies are also presented to help provide insight into the mechanisms of AIS; these are the idiotypic network approach and the Dendritic Cell Algorithm.

### 14.1 Introduction

Nature has acted as inspiration for many aspects of computer science. A trivial example of this is the use of trees as a metaphor, consisting of branched structures, with leaves, nodes and roots. Of course, a tree structure is not a simulation of a tree, but it abstracts the principal concepts to assist in the creation of useful computing systems. Bio-inspired algorithms and techniques are developed not as a means of simulation, but because they have been inspired by the key properties of the underlying metaphor. The algorithms attempt to improve computational

---

Julie Greensmith, Amanda Whitbrook and Uwe Aickelin  
School of Computer Science, University of Nottingham, Nottingham, NG8 1BB, UK  
e-mail: {jqg, amw, uxa}@cs.nott.ac.uk

techniques by mimicking (to some extent) successful natural phenomena, with the goal of achieving similar desirable properties as the natural system. This is demonstrated in both neural networks [17] and genetic algorithms [25].

Artificial Immune Systems (AIS) [20] are algorithms and systems that use the human immune system as inspiration. The human immune system is a robust, decentralised, error tolerant and adaptive system. Such properties are highly desirable for the development of novel computer systems. Unlike some other bio-inspired techniques, such as genetic algorithms and neural networks, the field of AIS encompasses a spectrum of algorithms that exist because different algorithms implement different properties of different cells. All AIS algorithms mimic the behaviour and properties of immunological cells, specifically B-cells, T-cells and dendritic cells (DCs), but the resultant algorithms exhibit differing levels of complexity and can perform a range of tasks.

The major part of AIS work to date has been the development of three algorithms derived from more simplified models: negative selection, clonal selection and immune networks. However, these first-generation AIS algorithms have often shown considerable limitations when applied to realistic applications. For this reason, a second generation of AIS is emerging, using models derived from cutting-edge immunology as their basis, not simply mechanisms derived from basic models found in text books.

The aim of this chapter is to give the reader an overview of the field of AIS by taking a high-level perspective of its evolution. In Section 14.2 an overview of the major developments in immunology is presented, incorporating a number of immunological theories. Section 14.3 describes the development of AIS over the past two decades and the next two sections showcase two particular examples of AIS algorithms: an idiotypic network in Section 14.4 and the Dendritic Cell Algorithm (DCA) in Section 14.5. Section 14.6 concludes the chapter, with a summary and details of potential future trends.

## 14.2 Immunological Inspiration

The human immune system can be used as inspiration when developing algorithms to solve difficult computational problems. This is because it is a robust, decentralised, complex, and error-tolerant biological system, i.e. it possesses properties that make it ideal for certain application areas, such as computer intrusion detection and pattern recognition. The human system is also well studied within immunology and is viewed as the most sophisticated of immune systems in nature. Although its precise function remains undetermined, it is postulated that it has two roles: to protect the body against invading micro-organisms (pathogens) and to regulate bodily functions (homeostasis).

Immunologists like to describe the immune system as consisting of two parts, namely the *innate* immune system and the *adaptive* immune system. It was originally thought that these were two distinct sub-systems with little crossover, with the innate system responding to known threats and the adaptive immune system tackling

previously un-encountered threats. However, current research suggests that it is the interplay between these two systems that provides the high level of protection required, i.e. the ability to discriminate between ‘self’ and ‘nonsel’ entities.

In this section the basic principles of immunology are introduced from the historical perspective of their development. For a more comprehensive, biological view of the immune system, the interested reader should refer to any of a number of more standard immunology texts, for example [43] and [15].

### 14.2.1 Classical Immunology

Until relatively recently, the central dogma of immunology was self–nonsel discrimination through the principles of *clonal expansion* and *negative selection*. These concepts have dominated the field since they were first described, as they provide adequate explanation of the function of the adaptive immune system over the lifetime of an individual.

In 1891, Paul Ehrlich [49] and his colleagues postulated that the defence mechanism against pathogens was the generation of immunity through the production of immunoglobulins termed *antibodies*. They showed that the antibodies generated are specific to the pathogen (antigen) being targeted and suggested that the immune system must remove these antigens before an infection spreads, *without responding to its own cells*. This led to the theory of the *horror autotoxicus*, which states that “an organism would not normally mobilise its immunological resources to effect a destructive reaction against its own tissues” [48]. It was later discovered that a particular type of lymphocyte (white blood cell) termed a B-cell is responsible for the production of antibodies and that the antibodies are proteins that can potentially bind to the proteins present on the invading antigens.

Following the characterisation of antibodies, the theory of *clonal selection* was proposed by Burnet [12]. This mechanism corroborated the notion of *horror autotoxicus* and found that “an individual somehow manages to prevent all future ability to respond to auto antigens i.e. self, leaving intact the ability to respond actively to the universe of other antigens i.e. nonsel” [12]. The notion implied that immune function contains a mechanism of tolerance, which Burnet described as an “irreversibly determined immunological self”. This formed a major constituent of a theory known as *central tolerance* and was subsequently proven as valid experimentally, earning Burnet a Nobel Prize for his efforts.

The clonal selection theory has two constituents. First, B-cells are selected to be fit for purpose during a ‘training period’. Cells expressing receptors (cell surface-bound antibodies) that can match antigen are kept to form the B-cell population, but cells that cannot bind to antigens are removed. Once B-cells are released into the periphery, encounter with external antigens causes the cells to produce free versions of the B-cell receptor, i.e. antibodies, which can bind to the matching antigen. Second, the process of antibody tuning occurs through *somatic hypermutation* and *affinity maturation* [46]. If a B-cell matches an external antigen, the cell

clones itself. However, the hypermutation process ensures that exact clones are not formed; the clones express B-cell receptors that are slight variants on the parent cell's receptor. This is a type of biological optimisation, ultimately resulting in antibodies that can bind more successfully to external antigens. The antibodies can therefore be used as markers of nonself entities within the body. The whole process is termed affinity maturation and is used to generate the most responsive antibodies.

As the century progressed, a second class of white blood cell, T-cells, were characterised, and in 1959, the principle of *negative selection* was proposed by Joshua Lederberg, a then colleague of Burnet. He established the link between foetal development and the generation of tolerance to self-substances, termed self-antigen, noting the co-occurrence of the initial production of T-cells and tolerance to self-antigen. This led to the idea that the selection process implied "self learning through negative selection", and caused Lederberg to suggest that "whenever produced, lymphocytes (T-cells) undergo a period of immaturity during which antigen recognition results in their death" [48]. He also proposed that further activation of the T-cells in the tissue is needed for the cells to develop the ability to remove pathogens such as bacterial agents and virally infected self-cells.

During embryonic development in the womb, T-cells migrate to an immune system organ, the thymus. Whilst in the thymus, the newly created T-cells are exposed to a comprehensive sample of self-antigen. Any T-cell displaying a receptor which matches a self-antigen is removed. This process continues until puberty, after which the thymus shrinks to a negligible size. So-called self-reactive T-cells are thus eliminated through this filtering process.

### **14.2.2 The Immunologists' 'Dirty Little Secret'**

According to Ehrlich's *horror autotoxicus*, the immune system should not respond to self and should aim to eliminate all sources of nonself. However, this phenomenon is not always observed and numerous noteworthy exceptions have been discovered [43], questioning the credibility of the self–nonself dogma, for example:

1. Vaccinations and immunisations require adjuvants, namely microbial particles, that provide additional stimulation of the immune system;
2. What the body classes as self changes over time, an effect termed *changing self*. This phenomenon is observed in women during pregnancy;
3. Human intestines are host to colonies of bacteria that serve a symbiotic function. These organisms are clearly nonself, yet no immune response is mounted;
4. In the western world, an individual's immune system can sometimes start to respond to benign particles such as pollen, cat saliva, latex, peanut proteins, resulting in allergic reactions;
5. An individual's immune system can sometimes begin to attack its host in the form of autoimmune diseases, for example, multiple sclerosis and rheumatoid arthritis.

### 14.2.3 Costimulation, Infectious Nonself and The Danger Theory

Three main theories have both challenged and augmented the process of self–nonself discrimination including:

- Costimulation;
- Infectious nonself;
- Danger signal recognition.

Some of the cells involved in these theories are part of the innate immune system that was first observed by Metchikoff in 1882 [48]. He noted that invertebrates such as shrimp and starfish mobilise phagocytes, which ingest invading pathogens, clearing the threat from the host. This first line of defence is also found in humans and consists of a diverse array of interacting cell types. The innate system was initially seen as the adaptive system's lesser counterpart, as it did not appear to be as sophisticated. However, there has been renewed interest in the innate system, as it is now thought to provide some of the answers to the problems associated with the theories of adaptive immunity.

The concept of costimulation was introduced in an attempt to overcome a problem observed in the hypermutation of antibodies, i.e. if the resulting hypermutated antibodies have a structure that could react to self cells, it would cause *horror autotoxicus*. It was hence suggested that B-cells work in conjunction with T-cells [48] and that a B-cell would be eliminated if it did not receive a *costimulatory signal* from a 'helper T-cell'. Later it was shown that helper T-cells are also regulated by a 'stimulator cell' that provides the costimulatory signals. These *professional antigen-presenting cells* are known as dendritic cells (DCs) and are part of the innate immune system. The process of costimulation casts doubt on the theory of central tolerance, placing the innate system in control of the immune response.

The infectious nonself model proposed by Janeway in 1989 [34] further improved understanding of costimulation. Janeway suggested that the DCs perform their own version of self–nonself discrimination. This is based on their ability to recognise the signatures of bacterial presence innately, a skill developed over millennia throughout the evolution of the species. It is shown that DCs contain a repertoire of receptors on their surface, tuned for binding to molecules produced exclusively by bacteria. These molecules are collectively termed PAMPs (pathogen-associated molecular patterns). Janeway showed that the induction of an immune response is facilitated by the production of costimulatory molecules from DCs. When exposed to PAMPs and antigen, the DC produces a collection of molecules that assist in their binding to a T-cell, increasing the time a T-cell remains in contact with a presented antigen. This timing issue is thought to be crucial in the activation of T-cells.

Infectious nonself can explain the need to add adjuvants to vaccines. Adjuvants are formed from neutered bacterial detritus, which, according to the theory, provide the PAMPs necessary to mount an immune response. It also explains why no response is mounted to changing self, as the absence of a second signal leads to the deactivation of T-cells. However, the infectious nonself model cannot explain tolerance to symbiotic bacteria, which produce PAMPs, yet are not eradicated from the

body. Furthermore, this model cannot explain the phenomena of autoimmunity and its relatively high frequency of occurrence in the western world.

Despite the addition of a second costimulatory signal to the self–nonself model, it became apparent that a piece of the immunological puzzle was still missing. It was unclear why the immune system should respond to self or why bacteria producing PAMPs were not classed as foreign. In 1994, Matzinger proposed that the immune system is controlled by the detection of damage to the body [42], not the detection of specific antigen structures or bacterial products. Matzinger suggested that the activating *danger signals* do not come from external sources, but are produced by the cells of the body when a cell dies unexpectedly (necrosis). The danger theory also proposes that the cells of the innate immune system can actively suppress an immune response in the absence of danger and in the presence of molecular signals produced when cells die normally (apoptosis).

DCs are sensitive to both the signals of necrosis and apoptosis in addition to PAMPs and are attracted to areas in which cells are dying. They collect debris, including potential antigens, and all of the molecules found in the extracellular matrix (their environment) contribute to the regulation of their internal signal processing mechanism. If a DC is exposed to the molecules from necrosing cells, it transforms to a mature state. If it is exposed to the suppressive molecules of apoptosing cells, then it is transformed to a semi-mature state. The DC eventually complexes with a T-cell, i.e. a DC and T-cell bind if the antigen collected and presented by the DC has a sufficient binding affinity with the T-cell antigen receptor. If the DC is in the mature state, the T-cell becomes activated and all entities bearing that antigen are eliminated. If the DC is semi-mature, the T-cell is tolerised to the presented antigen and no response to it is generated. In this way, the processing of the input molecular signals provides the immune system with a sense of context; in other words, if an entity is foreign but harmless, then the immune system does not waste resources responding to it.

The peripheral-tolerance danger model can also account for the effects of autoimmunity; when a self-protein is present in the same place and at the same time as the antigen of a pathogen, the immune system may respond to its own tissue, as both host and foreign antigens are collected by the same DCs. This has been framed within the context of multiple sclerosis, as the symptoms frequently appear in combination with bacterial or viral infection.

Despite its ability to explain several key anomalies, acceptance of the danger theory has been slow within immunology. There has been a lack of experimental evidence to support Matzinger's ideas, and no single 'danger signal' has been discovered, though characterisation of the molecules involved is improving as molecular techniques advance.

#### 14.2.4 Idiotypic Networks: Interantibody Interactions

In addition to the research on mechanisms of immune discrimination, theories exist that attempt to explain the various emergent properties of the immune system. One

of these theories is the *idiotypic immune network theory*, initially proposed by Jerne in 1974 [35]. The theory postulates that interactions between immune cells (and not necessarily external agents) cause modulation in the behaviour of the immune system as a whole. This modulation is proposed to lead to the generation of *immune memory*, i.e. the ability of the immune system to remember past encounters with pathogens, and hence provide a secondary response that is both accurate and rapid. The idiotypic network model does not attempt to contradict the principles outlined in classical immunology, but provides a complementary theory of antibody stimulation, where antibodies can influence other antibodies in addition to antigens. Idiotypic models have been developed, although no physical evidence exists to support the theory.

#### 14.2.5 Summary

Immunologists classify the human immune system into two distinct sub-systems, the innate and adaptive. Until recently the adaptive system, responsible for modification of the immune response over the lifetime of an individual (through the tuning of B- and T-cells), was viewed as far more sophisticated than its innate counterpart. The selection mechanisms of the B- and T-cells, and their processes of adaptation form the major part of the self–nonself principle, which states that the immune system is activated in response to the detection of foreign antigen, but does not respond to self-antigen. However, the adaptive model of immune activation has problems associated with it and these have led immunologists to look in greater detail at the innate immune system, adapted over the lifetime of the species, which responds quickly to invaders based on receptors encoded within the genome. It is now thought that it is the interaction between the innate and adaptive systems and their cells that provides the necessary protection, and consequently, there has been fresh interest in the cells of the innate system, for example DCs. These are responsible for translating and integrating information from the tissue to the T-cells, which results in either activation or tolerisation of the immune system. While the classical self–nonself view is important to immune function, the interplay between the two systems and the corresponding cells influences the ultimate decision as to whether or not to respond to an antigen.

### 14.3 The Evolution of Artificial Immune Systems

AIS is the collective name for a number of algorithms inspired by the human immune system. Unlike genetic algorithms, for which there is an archetypal algorithm and variants thereupon, there exists no single algorithm from which all immune algorithms are derived. However, all research within AIS stems from foundations in theoretical immunology and numerous parallel streams of research have been conducted over the past 20 years, resulting in the development of distinct sub-streams, including computational immunology. The evolution of the various approaches that

exist within AIS is depicted in Figure 14.1, which shows major milestones in research, significant papers (given in quotes) or algorithms that have shaped the field of AIS. The white-ringed hubs represent the significant works within a particular sub-stream and the terminating rectangles show branches of the research that are not currently active. In addition, the proximity of the sub-stream to the stream of theoretical immunology in the centre represents the extent of the immunological modelling, with more modern AIS approaches closer to the underlying metaphor.

The diagram also shows that AIS are classified into two distinct groups; first- and second-generation algorithms. The first-generation algorithms use simplistic models of immunology as the initial inspiration, for example negative and clonal selection. In contrast, the second-generation algorithms, for example, the Dendritic Cell Algorithm (DCA) [29], are built on a foundation of interdisciplinary research that allows for a much finer-grained encapsulation of the underlying immunology. Although most of the second-generation algorithms are still in their infancy, and require much more theoretical study, they are showing great promise in a number of application areas.

In this section each of the sub-streams and its applications are described individually and in chronological order, so that the evolution of AIS can be traced. In particular, negative selection, clonal selection and immune network approaches (the key first-generation algorithms) are discussed in detail, and the recently developed second-generation algorithms that use the ‘Conceptual Framework’ methodology, are also treated.

### **14.3.1 Computational and Theoretical Immunology**

A vein of computational and theoretical immunology lies at the core of AIS, as the process of developing mathematical models of immunological mechanisms is similar, at least, in principle to the development of immune-inspired algorithms. It is not surprising, therefore, that theoretical models of immune phenomena acted as a foundation for the initial AIS algorithms, clonal and negative selection, and immune network-based approaches.

In the case of the clonal selection principle, this was initially based on works carried out in the 1970s by Burnett [12], where affinity metrics were first characterised mathematically. In combination with this model, Jerne’s idiotypic network model was formalised by Farmer et al. in the 1980s [22] and stipulated the interaction between antibodies mathematically. The network model was seen as having computationally useful properties and provided a network-based approach distinct from both neural networks and genetic algorithms. The model was also interpreted by Bersini and Varela [9] with numerous refinements, and the combination of these two approaches forms the cornerstone of all AIS work that abstracts the idiotypic network.

Similarly, a theoretical model of the selection of T-cells in the thymus by Perelson et al. [10] resulted in the development of negative selection as a technique within AIS. This model detailed the selection of T-cells (based on affinity metrics) to model

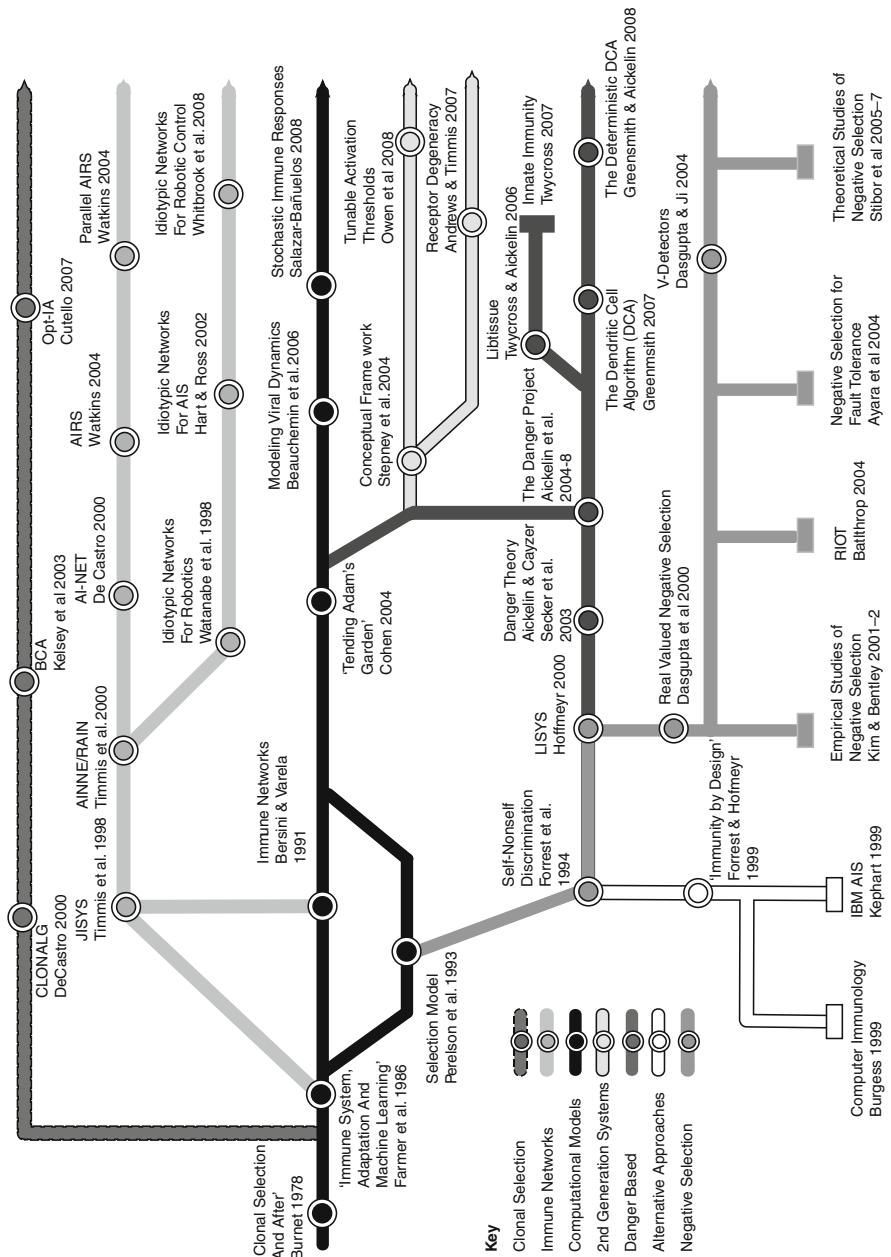


Fig. 14.1 The evolution of AIS from 1978 to 2008 showing specific research milestones and researchers involved, not individual publications.

the suitability of a T-cell receptor (TCR) for the detection of potential nonself antigen. The transfer from theoretical immunology to an AIS algorithm was by virtue of a collaboration between Perelson and Forrest, using Forrest's expertise in machine learning to improve Perelson's model.

It was around this time that the primary algorithms were applied to a battery of computational tasks and AIS began to diverge from theoretical immunology. However, although the initial performance of the developed algorithms was good, the techniques proved no *better* than the state-of-the-art algorithms that already existed in the chosen problem domains. Consequently, AIS researchers started to turn back to the underlying immunology (both experimental and theoretical), as it was assumed that the developed algorithms were based on out-dated, oversimplified models of the computation actually performed by the human immune system. Of course, theoretical immunology had also progressed since the 1990s.

In 2004, Cohen published a book entitled *Tending Adam's Garden* [14] that described the immune system as a complex adaptive system. Other similar research into a systemic perspective of the immune system, paired with the increase in popularity of interdisciplinary approaches, enticed AIS researchers to renew their interest in the underlying metaphors. At this point, theoretical immunologists were welcomed into the field of AIS, acting as translators between the complicated and dynamic world of experimental immunology and computation. The AIS algorithms developed as a result of this incorporated many new ideas from modern immunology and showed promise to out-perform older systems. In addition, in a bid to attract more researchers with a background in immunology, the AIS community devised a computational immunology stream as part of its conference [8]. Three examples of high-quality research in this area include a model of viral dynamics [7], an investigation into the cellular maximal frustration principle [1], and a model of the stochastic nature of immune responses [47].

As computational and theoretical immunology becomes more sophisticated, it seems likely that the boundaries between the two fields will blur, resulting in the development of more sophisticated AIS algorithms. AIS practitioners are hopeful that any new system developed will remain faithful to the underlying principles, as stipulated by the creators of the 'Conceptual Framework' [51] approach to AIS development. Whether this approach will bear fruit is conjecture, but it has certainly given a lease of life to a field that has strayed far from its initial roots.

### 14.3.2 Negative Selection Approaches

The first example of an implemented AIS performing a useful computational task was an incarnation of a self–nonself discrimination system, used for the detection of computer virus executables [24]. (Incidentally, the precursor to this system was the original collaboration between computer science and immunology, i.e. the development of a genetic algorithm-based approach for understanding the mechanisms of pattern recognition within the immune system [50].) The self–nonself

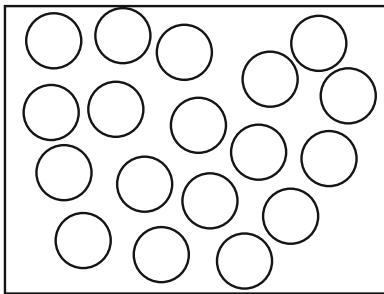
discrimination system involved creating a behaviour profile of sequences of system calls on a computer network during a period of normal function. To aid in detecting malicious intruders, any subsequent sequences were matched against the normal profile and any deviations reported as a possible intrusion. This research and its related work are perhaps the most widely known and popularised AIS to date [23], as the data used is popular amongst the intrusion detection community, with nearly one thousand citations.

The approach attracted a great deal of attention from the security community, as exemplified in the research of Kephart [38]. This was the first attempt to apply AIS within a commercial setting and consequently introduced AIS to a wider audience. The research was inspired by the efforts of Forrest et al. [23] and subsequent work by Hofmeyr and Forrest [32] in their paper *Immunity by Design*, and inspired a more systemic approach to AIS development, as pursued by Burgess [11]. Kephart also attempted to build on the system-profiling approach to intrusion detection by implementing a heterogeneous AIS. His work, and also that of Burgess, is a good example of alternative approaches based on the self–nonself principle.

However, the major development in this sub-stream was the introduction of a true negative selection algorithm in a system named ‘Lisys’, which consisted of three phases. Here, the first phase was used for the definition of self, i.e. the normal profile was generated from input data to encompass normal behaviour patterns defined in advance to form a sense of self. The second phase involved the generation of a set of random detectors containing a representation suitable for matching the patterns used to create the self-profile. The final phase implemented the detection of anomalies in previously unseen data by comparing each detector against all self patterns contained within the self-profile. If any of the randomly generated detectors matched a self-pattern, the detector was deemed unsuitable and was removed from the detector set. However, if the detector did not match any self items it was saved and became part of the pool used for anomaly detection. Thus, when the highly tuned detector set was presented with unseen test data, if any detector matched a pattern, the pattern was classed as anomalous and marked accordingly. A depiction of the algorithm at the core of this system is given in Figure 14.2.

A full description of the multiple-stage negative selection algorithm is given in the work by Hofmeyr [31], which encoded the detectors as bit strings and used an r-contiguous bit function for matching. Extensions to the work include the incorporation of real values into the encoding, the use of multi-dimensional vector representations known as V-detectors and the use of adaptable thresholds to reduce false positives [6]. Negative-selection algorithms have also been employed to solve fault tolerance problems and numerous other anomaly detection problems.

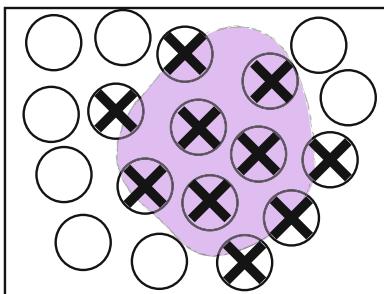
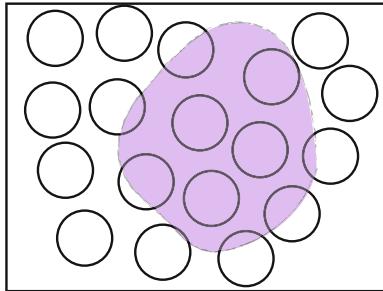
Despite its initial promise, negative selection has been shown to have a number of associated problems that can render it somewhat undesirable for use in network intrusion detection. First, the necessity to create a randomly generated initial detector population can be prohibitive, because, as the dimensionality of the feature space increases, the number of detectors required to cover it increases exponentially. Second, negative selection is a one-shot supervised learning algorithm, where the definition of normal is not updated as time progresses. This is particularly relevant

**Step One:**

Randomly generate initial detector-population with  $n$  detectors to cover the feature space, where each circle is one detector.

**Step Two:**

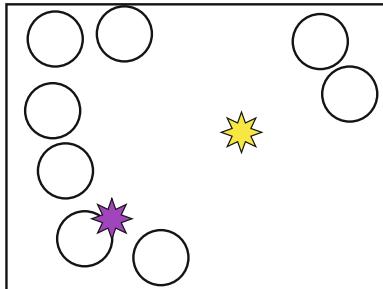
Using the training data, define regions of 'self' space, representing normal.

**Step Three:**

Delete all detectors which overlap with the defined self region, leaving detectors primed to detect nonself entities.

**Step Four:**

Introduce new pattern (antigen) and calculate affinity with nearest detector. If affinity is greater than a defined threshold, the detector is activated and the antigen is classed as anomalous (). Antigen with insufficient affinity are classed as normal ().



**Fig. 14.2** An illustration of negative selection.

to computer security where what is defined as normal has the tendency to change over time. Negative selection algorithms can therefore cause excessive numbers of false positive alerts, which can cripple a system. The problems with the algorithm are discussed further in Kim and Bentley [39] and are proven theoretically by Stibor et al. [52]. Although numerous modifications and variants in representation have

been made, such as the addition of variable length detectors, the algorithm seems fit for purpose only for small, constrained problems where the definition of normal is not likely to change and the set encompassing normal is small. For a comprehensive overview of the negative selection algorithm, the interested reader should refer to the review by Ji and Dasgupta [36].

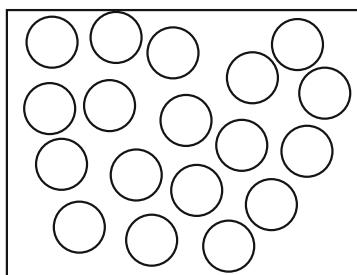
### 14.3.3 Clonal Selection Approaches

During the early years of AIS, researchers recognised that, in addition to the T-cell inspiration employed by Forrest et al. [32], basic models of B-cells and their corresponding antibodies could act as a good underlying metaphor. B-cells produce antibodies of a specific configuration, and their diversity is stimulated upon encounter with a foreign antigen, where the resulting B-cell clones vary the receptor configuration in order to perform a biological local search to find the best-fitting receptor. The B-cell model appeared ripe for exploitation, given the similarities with local search and optimisation techniques, and in 2000 a theoretical model of the *hypermutation process* proposed by Burnet [12] served as inspiration for CLONALG [19], a popular AIS algorithm involving an abstract version of the cloning and hypermutation process.

All clonal selection-based algorithms (CSA) essentially centre around a repeated cycle of *match*, *clone*, *mutate* and *replace* and numerous parameters can be tuned, including the cloning rate, the initial number of antibodies and the mutation rate for the clones. CLONALG, AI-NET, the B-cell algorithm [37] and AIRS [60] all incorporate this basic functionality. (AI-NET contains constituents of both CSA and immune network approaches [19].) The CSA used in CLONALG is illustrated in Figure 14.3.

CSAs have a strong resemblance to genetic algorithms without crossover, but their notion of affinity and their significantly higher mutation rate (the hypermutation component) distinguish them from similar adaptive algorithms. CSAs also share properties with both K-means and K-nearest neighbour approaches. The CSA technique would be most similar to a K-nearest neighbour scheme where K is one, combined with features of K-means where the position of the centroids is adjusted (analogous to the creation of memory antibodies). However, again, the affinity metrics and the hypermutation components make CSA somewhat distinct from these methods.

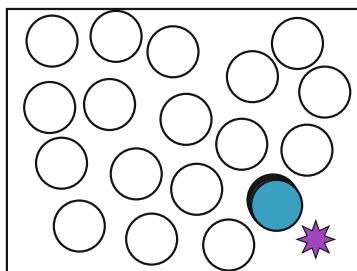
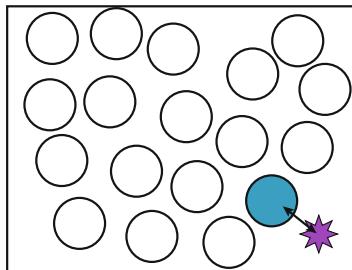
The primary uses of CSA are for pattern recognition and optimisation, exemplified by the successful application of an optimised variant of CLONALG termed Opt-AI [18] to the prediction of protein secondary structure. It is the hypermutation component of CSA, where a dynamic local search is performed, that implies its suitability for optimisation, and this is exemplified with Opt-IA. Another example of a CSA is AIRS, a successful multi-class classifier that contains a clonal selection component. This system also employs memory cells, created when a stimulated B-cell has a sustained affinity. Immune memory models frequently accompany clonal selection approaches, but the underlying immunology is rather unclear, even regarding

**Step One:**

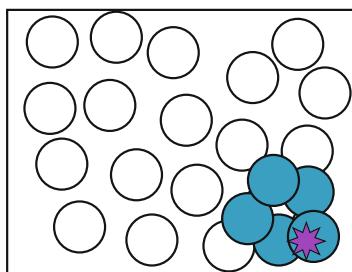
Randomly generate initial antibody population with  $n$  detectors, where each circle is one antibody.

**Step Two:**

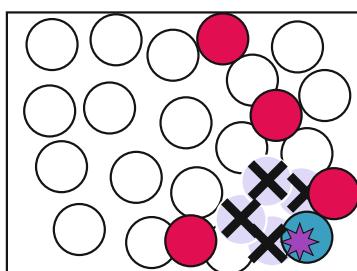
Introduce new pattern (antigen ) and select the nearest clone (coloured) using a defined distance metric such as the Euclidean distance.

**Step Three:**

Clone nearest antibody in proportion to the affinity between antibody and antigen. The greater the affinity the greater the number of clones produced.

**Step Four:**

Mutate clones, with distance of mutation inversely proportional to affinity. The greater the affinity the greater the distance between mutant antibodies.

**Step Five:**

Find best matching clone and assign clone's class to antigen. Delete other superfluous clones and for each deletion, replace with new randomly generated antibody. Repeat steps two to five until a stopping condition is met.

**Fig. 14.3** An illustration of clonal selection.

the existence of such cells. This has made the development of specific models very difficult but possible, as demonstrated by Wilson et al. [63] with the motif-tracking algorithm.

The process of repeated filtering of candidate solutions in the form of antibody populations results in a type of optimisation when taken within an AIS context, although it is debatable whether the solutions provided by the human immune system itself are optimal. As argued by Timmis and Hart [54], CSA has produced solutions that have a tendency to be the most robust, though not necessarily the most optimal. This makes them particularly suited for more complex optimisation problems such as multi-objective optimisation. Their robustness, coupled with the fact that they are one of the most well understood of the AIS algorithms, makes them a popular choice amongst similar techniques.

#### 14.3.4 Idiotypic Network Approaches

In this section, the basic principles of the idiotypic network theory proposed by Jerne [35] are explained, and a particular example of a hybrid system that combines an immune network with a clonal selection-based model is presented and discussed. A more detailed example of an artificial idiotypic network is provided in the case study in Section 14.4.

In order to understand the principles of the idiotypic network theory, it is necessary to introduce the concepts of epitopes, paratopes and idiotypes. The clonal selection theory states that division occurs for B-cells with receptors that have a high degree of match to a stimulating antigen's binding region or *epitope* pattern and that these cells then mature into plasma cells that secrete the matching receptors or antibodies into the bloodstream. Once in the bloodstream the antibody combining sites or *paratopes* bind to the antigen epitopes, causing other cells to assist with elimination. Antibody paratopes and antigen epitopes are hence complementary and are analogous to keys and locks. Paratopes can be viewed as master keys that may open a set of locks and some locks can be opened by more than one key.

However, Jerne's network theory suggests that antibodies also possess a set of epitopes and so are capable of being recognised by other antibodies. Epitopes unique to an antibody type are termed *idiotypes* and the group of antibodies sharing the same idiotpe belongs to the same *idiotype*. When an antibody's idiotpe is recognised by the paratopes of other antibodies, it is suppressed and its concentration is reduced. However, when an antibody's paratope recognises the idiotypes of other antibodies, or the epitopes of antigens, it is stimulated and its concentration increases.

The idiotypic network theory hence views the immune system as a complex network of paratopes that recognise idiotypes and idiotypes that are recognised by paratopes. This implies that B-cells are not isolated, but are communicating with each other via collective dynamic network interactions. The network continually adapts itself, maintaining a steady state that reflects the global results of interacting with the environment. This is in contrast to the clonal selection theory,

which supports the view that promotion of a B-cell to a memory cell is the result of antibody–antigen interactions only. Jerne states that each individual develops a unique, self-regulating immune network, and when it is established, it must possess stable features. He hence proposes that immunological memory may be more dependent upon network changes than upon the endurance of populations of memory cells.

His theory asserts that antibodies continue communicating even in the absence of antigens, which produces continual change of concentration levels. A more recent model by Farmer et al. [22] adds additional dynamics that account for the domination of a single antibody in the presence of antigen, since the cell with the paratope that best fits the antigen epitope contributes more to the collective response. It presents itself to the system as the antigenic antibody, which disturbs the network, inducing further interantibody suppression and stimulation.

Although the theory has been largely ignored by the wider immunology community, it has gained much popularity within AIS due to its ability to produce flexible selection mechanisms. Furthermore, the behaviour of an idiotypic network can be considered intelligent, as it is both adaptive at a local level, and shows emergent properties at a global level. The system is also autonomous and completely decentralised, making it ideal for applications such as mobile-robot behaviour arbitration [40, 41, 59], identifying good matches for recommendation software [13], and negotiating options for configuring communication software [53].

An early example of AIS research inspired by Jerne's theory is the system named 'Jisys', developed by Hunt and Cooke, and later Timmis [33]. The system was based on the idiotypic network model formalised by Farmer et al. [22] and later Bersini and Varela et al. [9] and utilised the concepts of stimulation and suppression effects within a network of antibodies. The system can be considered as something of a hybrid, since it also incorporates the concepts of clonal expansion and somatic hypermutation within the antibody populations. The system led to the development of a number of other network-based systems, including ANNIE/RAIN [55], which is a resource-based unsupervised clustering algorithm and AINET [19]. Components from ANNIE/RAIN are incorporated into AIRS in addition to elements of clonal selection.

### ***14.3.5 Danger Theory Approaches***

All of the algorithms described above (clonal and negative selection and the immune networks) diverged from the underlying immunology at an early stage in their development. This phenomenon often occurs in AIS because, as novel variants are created, any remaining immune inspiration is abstracted away in order to produce systems that are easy to characterise computationally. Consequently, the resulting systems may fail to model certain computationally desirable features of the immune system. In addition, since the algorithms are developed from a computational perspective, it can be difficult to distinguish AIS approaches from more established machine learning techniques. This is exemplified by the similarities of

CSAs with K-nearest neighbour approaches and evolutionary search techniques. Although the first-generation algorithms continue to be applied to numerous pattern recognition, detection and classification problems, little progress has been made with the algorithms themselves for a number of years. This, coupled with the somewhat mediocre performances achieved by such algorithms on benchmark tests, has recently led AIS researchers to re-think the fundamentals of AIS design [54]. Instead of using highly simplified models of isolated immune components, systems could be designed to incorporate more complex, current and sophisticated models. The idea gave rise to a hypothesis; would the incorporation of finer grained models improve the performance of AIS algorithms and make them more applicable?

The Danger Project (Aickelin et al. [2]), a 4-year interdisciplinary collaboration between an AIS development team and practical immunologists aimed to answer this fundamental question. Their research was chiefly motivated by the scaling and false-positive problems associated with negative selection and was based on a proof of concept paper by Aickelin and Cayzer [3]. Here, the immune system was re-examined in an attempt to overcome the difficulties and it was postulated that negative selection-based intrusion detection systems may be missing a key constituent; danger signals.

As described in Section 14.2, the human immune system cannot rely on self–nonself discrimination alone, so it seems unreasonable to design AIS systems that depend only upon this principle. The aim of the Danger Project was to incorporate the danger theory into AIS, with a view to producing robust intrusion detection systems, capable of fast real-time analysis and low rates of false alarms. At the start of the project in 2004, the working methodology of the research team was unique in AIS; the practical immunologists gave the computer scientists insight into the actual mechanisms of detection employed by the immune system and the AIS researchers were able to build abstract computational models of the cells involved in the detection of danger signals, which formed the basis of novel algorithms and frameworks. Moreover, the practical immunologists were able to assist in refining the models by performing experiments to fill in any gaps in knowledge that were identified.

Two separate areas of research arose out of the danger project in addition to the published immunological results. The first was the development of the libtissue system, an agent-based framework that facilitated a style of agent-based simulation to house the novel algorithms [57]. A novel algorithm (termed ‘tlr’) was developed to test the framework and showed some success when applied to the detection of anomalous system calls [56]. The algorithm is one of the few instances of AIS where more than one cell type is employed, in this case, DCs and T-cells. The second research area was the creation of the DCA [26], a second-generation example, and the newest addition to the mainstream set of AIS algorithms. The DCA is based on a model of the function of dermal dendritic cells and their ability to discriminate between healthy and infected tissue. In nature DCs correlate molecular signals found within tissue and use this information to assess the context of the monitored area. In addition to signal processing, DCs collect debris, which is processed to form antigen. After a period of time, DCs mature and migrate

from the tissue to a lymph node, where they present their context information and their antigen to a population of T-cells, instructing the T-cell with the appropriate response.

In the DCA, the DC mechanisms are abstracted and used to form the model. To date the DCA has been applied to port scan detection, insider attack detection, botnet zombie machine detection, standard machine learning intrusion datasets, robotic security, schedule overrun detection in embedded systems, sensor networks, and other real-time, dynamic problems. In numerous cases the algorithm is performing well, producing low rates of false positives, and a deterministic variant that has enhanced computational performance is currently under investigation [28]. The algorithm is described in detail in Section 14.5.

#### 14.3.6 Conceptual Framework Approaches

In parallel with the Danger Project, Stepney et al. [51] also identify the lack of rigour in the metaphors used to inspire AIS. To overcome this problem, they propose a framework (the ‘Conceptual Framework’) for the successful development of AIS. The methodology employs an iterative approach for the creation and testing of novel immune-inspired algorithms and four stages are identified as key:

- *Observation*: the biological system is probed using practical experimentation.
- *Models*: computational models are constructed to examine the biological system further, and abstract models are created from the computational models for translation into algorithms.
- *Algorithms*: computational systems are developed, implemented, and studied theoretically using the abstract models as a blueprint.
- *Applications*: the developed algorithms are applied to specific problems, with feedback to the algorithm for refinement.

The design of the framework stipulates that the flow of information between components is bi-directional, and involves an iterative process, updating the models and algorithms as information is incorporated. A framework for constructing algorithms is certainly necessary in principle, since it clearly defines the role each discipline must play, i.e. observation by immunologists, modelling by mathematicians, algorithm development by computer scientists and application testing by engineers.

Models of receptor degeneracy by Andrews and Timmis [4] are in development using the Conceptual Framework approach, with one modification, i.e. no direct collaboration with practical immunologists is formed. Instead, sophisticated immunological literature is used as inspiration to construct a novel computational model. Here, the constructed model is of T-cell activation within a lymph node, and a computational model of the interactions between T-cells and antigen presenting cells (e.g. DCs) is implemented using principles of cellular automata. In this work, it is identified that one key feature of activation is the degeneracy of receptors across the T-cell population. Degeneracy is defined as “elements which are structurally different but produce the same function...”. For example, one particular T-cell receptor can respond to more than one binding agent with similar effects. Degeneracy is a

desirable property that is inherent in numerous biological systems and is of particular interest to AIS as it may enable reduction of the number of detectors required. This would impact on the dynamics of the first-generation approaches, negative selection included.

The initial model is extended to incorporate tuneable activation thresholds for the responses of T-cells [5]. Dynamic thresholds are employed based on an existing immunological model, where the signal strength needed to activate the T-cell is derived from the frequency and magnitude of the stimulation of the cell over time. Similar research into formalising threshold methods using a type of process algebra known as stochastic  $\pi$  calculus has also been carried out and allows for the formulation of models within a defined modelling language [45], utilising the Conceptual Framework for its development.

Both approaches have yielded immunologically and computationally interesting results. However, neither technique has matured to the stage of a workable algorithm and, thus, their applicability to the wider AIS context is still undetermined. It is hoped that the integration of these mechanisms will impact on the function of AIS at some point, stimulating others to follow the Conceptual Framework. However, at the present time, no realistic claims about its effectiveness can be made, as it is too new to have mature work associated with it.

#### 14.3.7 Summary

AIS is a diverse field of study within bio-inspired computation, with the algorithms developed as distinct as the various parts of the immune system itself. This results in not one single AIS algorithm, but a collection of algorithms fit to solve a wide range of problems. Two generations of AIS are currently in use and development. The first-generation approaches draw inspiration from theoretical immunology models in combination with ‘text-book style’ mechanisms. Two major techniques from the first-generation, clonal and negative selection share properties with other machine-learning methods, such as K-nearest neighbour. Recently, second-generation algorithms have emerged, based on an interdisciplinary methodology. Although these approaches are still in the early stages of development, preliminary results, and the increasing popularity of algorithms like the DCA, suggest that second-generation algorithms may prove extremely useful.

To reinforce the concepts presented in this section, two examples of immune-inspired algorithms are examined in more detail. These are the idiotypic network and the DCA, representing first and second-generation algorithms, respectively.

### 14.4 Case Study 1: The Idiotypic Network Approach

Systems inspired by the idiotypic network theory include the interaction between antibodies, in addition to interactions between antibodies and antigens. Such systems are computationally useful, despite the fact that no immunological evidence

exists to support the underlying principles. Idiotypic network-based systems are largely inspired by the Farmer et al. computational model [22] of Jerne's idiotypic network theory [35], where binary strings of a given length  $l$  represent epitopes and paratopes. The model simplifies the biology so that each antigen and each antibody have only one epitope. Each antibody thus has a pair of binary strings  $[p, e]$ , and each antigen has a single string  $[e]$ . The degree of fit between epitope and paratope strings is analogous to the affinity between real epitopes and paratopes and uses the exclusive OR operator to test the bits of the strings (where 0 and 1 yield a positive score).

Exact matching between  $p$  and  $e$  is not required and, as strings can match in any alignment, one needs only to define a threshold value  $s$  below which there is no reaction. For example, if  $s$  was set at 6 and there were 5 matches (0 and 1 pairs) for a given alignment, the score for that alignment would be 0. If there were 6 matches, the score would be 1 and if there were 7 the score would be 2. The strength of reaction  $G$  for a given alignment is thus

$$G = 1 + \mu, \quad (14.1)$$

where  $\mu$  is the number of matching bits in excess of the threshold. The strength of reaction for all possible alignments  $m_{ij}$  between two antibodies  $i$  and  $j$  is given by

$$m_{ij} = \sum_{k=1}^q G_k, \quad (14.2)$$

where  $q$  is the number of possible alignments. In the Farmer model, differential equation (14.3) describes continuous antibody concentration changes occurring as a result of antigen stimulation, interantibody stimulation and suppression, and the natural death rate. Here,  $N$  is the number of antibodies and  $n$  is the number of antigens. The match specificities are given by  $m$ , with the first index referring to the epitope and the second to the paratope:

$$\frac{dx_i}{dt} = c \left[ \sum_{j=1}^n m_{ji} x_i y_j - k_1 \sum_{j=1}^N m_{ij} x_i x_j + \sum_{j=1}^N m_{ji} x_i x_j \right] - k_2 x_i. \quad (14.3)$$

The first sum in the square bracket expresses stimulation in response to all antigens. The  $x_i y_j$  terms model that the probability of a potential collision (i.e. match) between an antibody and an antigen (and hence the probability of stimulation) is dependent on their relative concentrations. The second and third sums represent suppression and stimulation, respectively, in response to all other antibodies. Parameter  $k_1$  allows possible inequalities between interantibody stimulation and suppression, and the  $k_2$  term outside the brackets is a damping factor, which denotes the tendency of antibodies to die, at a constant rate, in the absence of interactions. Parameter  $c$  is a 'rate' constant that simulates both the number of collisions per unit time and the rate of antibody production when a collision occurs. After each iteration, antibody concentrations are usually reduced using some sort of squashing function, for example, the one shown in Equation (14.4):

$$x_i(t+1) = \frac{1}{1 + \exp(0.5 - x_i(t+1))}. \quad (14.4)$$

The final antibody selected to tackle the presented antigen is the antibody with the highest concentration or the best antibody according to a metric that encompasses concentration and affinity.

Within mobile robotics, many researchers (e.g. Watanabe et al. [59]) encode antigens as environmental signals, and antibodies as robot behaviours, with the epitopes, paratopes and idiotypes encoded as binary strings. The antibodies have an action and a precondition (the paratope part), which are taken from fixed sets of actions and preconditions. They also have an idioype part—a disallowed condition, which defines antibody connection. The final antibody structures are determined using a genetic algorithm that evolves suitable combinations of idioype parameters, actions and preconditions. Both Farmer et al. [22] and Vargas et al. [58] have likened this technique to learning classifier systems (LCS) where antibodies can be thought of as classifiers with a condition and action part (the paratope) and a connection part (the idioype). In LCS the action part must be matched to a condition (antigen epitope) and the connections show how the classifier is linked to the others. The presence of environmental conditions causes variations in classifier concentration levels in the same way that antigens disturb the antibody dynamics.

In contrast to the binary-coding techniques, the work of Whitbrook et al. [61], which is concerned with mobile robot navigation, uses a fixed idioype matrix of real numbers  $I$  representing the degree of belief that an antibody–antigen combination is poor. A variable paratope-matrix of real numbers  $P$ , derived from antibody–antigen reinforcement learning (RL) scores, is used to model antibody–antigen affinities. Later work by these authors [62] deals with a set of  $N$  antibodies, where each is associated with a particular antigen, i.e.  $N = n$ . The technique evolves the action part of each antibody in the set and there are  $z$  sets. There are thus  $z$  matching antibodies for each antigen. The variable paratope is determined by RL as before, but the idioype is derived directly from the paratope and is also variable.

$P$  and  $I$  are used together to assess similarity between antibodies and hence determine interantibody stimulation and suppression levels. The antibody with the highest affinity to the presented antigen  $v$  is selected as the antigenic antibody, i.e. the antibody with the highest paratope value  $P_{iv}, i = 1, \dots, z$ . The system works by suppressing antibodies dissimilar to the antigenic antibody and stimulating similar ones. This is done by comparing the idioype of the antigenic antibody with the paratopes of the other antibodies to determine how much each is stimulated and by comparing the antigenic paratope with the idioypes of the others to calculate how much each should be suppressed. If the antigenic antibody is the  $r$ th antibody and  $n$  is the number of antigens, Equations (14.5) and (14.6) govern the increase  $\varepsilon$  and decrease  $\delta$  in affinity to  $v$  for each of the  $z$  matching antibodies, when stimulation and suppression occur, respectively:

$$\varepsilon_{iv} = \sum_{j=1}^n (1 - P_{ij}) I_{rj} x_{ij} x_{rj} \quad i = 1, \dots, z, \quad (14.5)$$

$$\delta_{iv} = k_1 \sum_{j=1}^n P_{rj} I_{ij} x_{ij} x_{rj} \quad i = 1, \dots, z. \quad (14.6)$$

The new affinity  $(P_{iv})_2$  is hence given by

$$(P_{iv})_2 = (P_{iv})_1 + \varepsilon_{iv} - \delta_{iv}. \quad (14.7)$$

All concentrations are re-evaluated using a variation of Farmer's equation, and the antibody selected is the one with the highest activation, which is a product of concentration and affinity. The chosen antibody may be the antigenic antibody or it may be some other that matches the presented antigen, in which case an *idiotypic difference* is said to occur. The research has so far shown that both real and virtual robots can navigate through mazes and other obstacle courses much more successfully when they employ the idiotypic selection mechanism, as opposed to relying on RL only. The authors have also attempted to examine the relationships between the parameters  $k_1$ ,  $k_2$  and  $c$  and the rates of idiotypic difference in order to gain insight into the mechanisms that underlie the algorithm's superior performance.

## 14.5 Case Study 2: The Dendritic Cell Algorithm (DCA)

The DCA, a second-generation algorithm based on an abstract model of natural DCs, is one of the most recent additions to the AIS family. It is essentially a meta-heuristic that uses input signals (heuristic approximations of what is normal and anomalous) to perform context-sensitive anomaly detection through both correlation and classification. The primary use to date has been the detection of intrusions in the fields of network [27] and robotic security [44].

Natural DCs are part of the innate immune system and are responsible for initial pathogen detection, acting as an interface between the innate and adaptive systems. They exist in three states of differentiation: immature, mature (exposed to the molecules from necrosing cells) and semi-mature (exposed to the molecules from apoptosing cells), and it is their terminal state of differentiation that is used by the adaptive immune system to decide whether or not to respond to a potentially harmful entity.

The DCA abstracts a multi-resolution model of a natural DC. To this end, four data types are required:

- *Antigen*: an enumerated-type object with a value that is used as an identifier for the suspect data to be classified. For ideal functioning, a number of antigens of the same value should be used to form an antigen type.
- *PAMP signal*: a real-valued attribute, where an increase in value is a definite indicator of anomaly.
- *Danger signal*: a real-valued attribute, where an increase in value is a probable indicator of damage, but there is less certainty than with a PAMP signal.
- *Safe signal*: a real-valued attribute, where an increase in value is a probable indicator of normality within a system. High values of the safe signal can cancel

out the effects of both the PAMP and the danger signals, possibly reducing the false-positive rate of the DCA.

Unlike the negative selection approaches, the DCA does not have an adaptive component and thus requires no formal training phase. Signal processing, the correlation between antigen and signals, is performed at the individual cell level, but the classification of antigen types occurs at the population level. In other words, each cell's input signals are transformed into cumulative output signals, with two output values per cell; the *costimulatory signal* (CSM), and the context value  $k$ , which is used to determine the terminal state of the cell. (A negative  $k$  represents a semi-mature cell and a positive  $k$  indicates a mature cell.) Each DC executes three steps per sampling iteration:

1. *Sample antigen*: the DC collects antigen from an external source (in the form of an antigen array) and places the antigen in its own data structure for storage.
2. *Update input signals*: the DC collects values of all input signals from a defined input signal array.
3. *Calculate interim output signals*: at each iteration, each DC generates three temporary output signal values from the received input signals, and these output values are added to obtain the cell's CSM and  $k$  values.

The DCA works on a static population of cells in which every cell removed is immediately replaced. Diversity is maintained across the population by random allocation of lifespan values within a specified range, i.e. when a cell is created it is given a limited time window for data sampling. This is also thought to add robustness to the algorithm. However, the lifespan reduces whenever the CSM value increases, since the CSM value is automatically derived from it. Once the lifespan is over, the cell ceases its sampling iterations and presents all the collected antigens, so that its  $k$  value can be determined. The cell is then reset and returned to the sampling iterations with a new lifespan value.

The antigens presented by each cell are processed across all presentations by all members of the DC population. The anomaly score,  $K_\alpha$ , of an antigen type is calculated using the  $k$  values presented in conjunction with each antigen type. Initial implementations of the DCA [30] are based on numerous probabilistic components, including random sorting of the DC population, probabilities of antigen collection, decay rates of signals, and numerous other parameters. While the same research shows that the algorithm can be successfully applied to real-world intrusion data, the system is very difficult to analyse given the sheer numbers of probabilities and parameters. Recently, a deterministic DCA [28] has been developed in order to obtain a greater understanding of the algorithm's function. The pseudocode for the deterministic version of DCA is given in Algorithm 1.

While the DCA removes the need to define self, it is still necessary to select a suitable antigen representation and to perform pre-classification of input signals. This represents a fundamental difference between this approach and negative selection for example, as the DCA relies on heuristic-based signals that are not absolute

---

**Algorithm 1** Pseudocode of the deterministic DCA
 

---

```

input : Antigen and Signals
output: Antigen Types and cumulative k values

set number of cells;
initialise DCs();
while data do
  switch input do
    case antigen
      antigenCounter++;
      cell index = antigen counter modulus number of cells ;
      DC of cell index assigned antigen;
      update DC's antigen profile;
    end
    case signals
      calculate csm and k;
      for all DCs do
        DC.lifespan -= csm;
        DC.k += k;
        if DC.lifespan <= 0 then
          log DC.k, number of antigen and cell iterations ;
          reset DC();
        end
      end
    end
  end
end
for each antigen Type do
  | calculate anomaly metrics;
end
  
```

---

representations of normal or anomalous. Although it bears an initial resemblance to neural networks, the variable lifespan, the population dynamics and the combined functionality of filtering, correlation and classification set it apart from these other approaches. Full details of the DCA are presented in Greensmith [26] and Greensmith et al. [29].

## 14.6 Conclusions

This chapter has examined AIS from the underlying immunology and the controversy surrounding competing theories to the application and implementation of immune algorithms as exemplified by the DCA. As mentioned, AIS are distinct from other fields within bio-inspired computing as not one archetypal system is used, but different methods are employed for different purposes. There are two reasons for this: first, the immune system is itself multifunctional, performing many different natural computational tasks, and second, AIS researchers model the immune system in different ways to suit their goals. For example, clonal selection with its

hypermutation function performs a type of local search and can be modified to perform optimisation. Alternatively, the DCA is based on natural DCs, which are responsible for initial pathogen detection in tissue and can hence be used for anomaly detection.

A shift is also evident in the methods used to create and develop AIS algorithms, from somewhat simplistic models based on outdated immunology to sophisticated interdisciplinary approaches based on rigorous abstract modelling, see the Danger Project [2] and Conceptual Framework [51]. While the Conceptual Framework approaches are not yet mature enough to yield tangible results, the DCA (developed using this method) is performing well across a range of problems [29] in comparison with other nature-inspired techniques.

### 14.6.1 Future Trends in AIS

The percentage of research directed towards specific areas of AIS can be estimated by looking at the AIS work reported in the 2008 ICARIS annual conference [8]. Exactly half of the papers in these proceedings are related to the application of first-generation algorithms and their variants. The second largest category is that of the second-generation approaches, followed by theoretical studies of AIS and computational immunology. This differs vastly from the state of the field a mere 5 years ago, where the two largest groups of papers were applications of clonal and negative selection, respectively, followed by idiotypic networks. This change in focus of the field suggests that, as the characterisation of the second-generation approaches improves, they will increase in popularity and may eventually dominate the field. What the future holds for AIS, like any discipline, is uncertain given that AIS algorithms are still evolving. As our knowledge of immunology increases, at some point in the future we may have the grounding and computational resources to build full, biologically accurate *computational immune systems*, based on both the innate and adaptive systems and their numerous cell types.

**Acknowledgments** This work is supported by the EPSRC (EP/D071976/1). The authors would like to thank Jon Timmis for his feedback and assistance.

## References

1. de Abreu, F., Nolte 'Hoen, E., Almeida, C., Davis, D. Cellular frustration: a new conceptual framework for understanding cell-mediated immune responses. In: Proceedings of the 5th International Conference on Artificial Immune Systems (ICARIS), LNCS 4163, pp. 37–51. Lisbon, Portugal (2006)
2. Aickelin, U., Bentley, P., Cayzer, S., Kim, J., McLeod, J. Danger theory: the link between AIS and IDS. In: Proceedings of the 2nd International Conference on Artificial Immune Systems (ICARIS), LNCS 2787, pp. 147–155. Springer (2003)
3. Aickelin, U., Cayzer, S. The danger theory and its application to artificial immune systems. In: Proceedings of the 1st International Conference on Artificial Immune Systems (ICARIS), pages 141–148. University of Kent at Canterbury Printing Unit (2002)

4. Andrews, P., Timmis, J. A computational model of degeneracy in a lymph node a computational model of degeneracy in a lymph node a computational model of degeneracy in a lymph node. In: Proceedings of the 5th International Conference on Artificial Immune Systems (ICARIS), LNCS 4163, pp. 164–177 Springer (2006)
5. Andrews, P., Timmis, J. Adaptable lymphocytes for artificial immune systems. In: Proceedings of the 7th International Conference on Artificial Immune Systems (ICARIS), pp. 376–386. Phuket, Thailand (2008)
6. Balthrop, J., RIOT: a responsive system for mitigating computer network epidemics and attacks. Master's thesis, University of New Mexico (2005)
7. Beauchemin, C., Forrest, S., Koster, F.T. Modeling influenza viral dynamics in tissue. In: Proceedings of the 5th International Conference on Artificial Immune Systems (ICARIS), LNCS 4163, pp. 23–36. Springer (2006)
8. Bentley, P., Lee, D., Jung, S. (eds.) Artificial Immune Systems, 7th International Conference, ICARIS 2008, Phuket, Thailand, August 10–13, 2008. Proceedings, vol. 5132 of Lecture Notes in Computer Science. Springer (2008)
9. Bersini, H., Varela, F. Hints for adaptive problem solving gleaned from immune network. In: Parallel Problem Solving from Nature, pp. 343–354 Springer (1991)
10. De Boer, R., Perelson, A., Kevrekidis, I. Immune network behaviour: From stationary states to limit cycle oscillations. *Bull. Math. Biol.* **55**(4):745–780 (1993)
11. Mark Burgess. Computer immunology. In: LISA '98: Proceedings of the 12th USENIX conference on System administration, pp. 283–298. USENIX Association. Berkeley, CA, USA (1998)
12. Burnet, F. Clonal selection and after. In: Bell, G., Perelson, A., Pimbley, G. (eds.) Theoretical Immunology, pp. 63–85. Marcel Dekker Inc., New York, NY (1978)
13. Cayzer, S., Aickelin, U. A recommender system based on idiotypic artificial immune networks. *J. Math. Model. Algorithms*, **4**:181–198 (2005)
14. Cohen, I.R. Tending Adam's Garden : Evolving the Cognitive Immune Self. Academic (2004)
15. Coico, R., Sunshine, G., Benjamini, E. Immunology: A Short Course. Wiley, New York, NY (2003)
16. Coutinho, A. The Le Douarin phenomenon: a shift in the paradigm of developmental self-tolerance. *Int. J. Develop. Biol.* **49**(2–3):131–136 (2005)
17. Cross, S., Harrison, R., Kennedy, R. Introduction to neural networks. *Lancet* **346**(8982):1075–1079 (1995)
18. Cutello, V., Nicosia, G., Pavone, M., Timmis, J. An immune algorithm for protein structure prediction on lattice models. *IEEE Trans. Evol. Comput.* **11**(1):101–117 (2007)
19. de Castro, L., Timmis, J. An artificial immune network for multimodal function optimization. In: Proceedings of the Congress on Evolutionary Computation (CEC), vol. 1, pp. 699–704, IEEE Computer Society. Los Alamitos, CA, USA (2002)
20. de Castro, L., Timmis, J. Artificial Immune Systems: A New Computational Approach. Springer London, UK (2002)
21. de Castro, L., Von Zuben, F. The clonal selection algorithm with engineering applications. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), Workshop on Artificial Immune Systems, pp. 36–37. Morgan Kaufmann, Las Vegas, NV (2000)
22. Farmer, J., Packard, N., Perelson, A. The immune system, adaptation and machine learning. *Physica D*, **2**(1–3):187–204 (1986)
23. Forrest, S., Hofmeyr, S., Somayaji, A., Longstaff, T. A sense of self for unix processes. In: Proceedings of the IEEE Symposium on Research in Security and Privacy, pp. 120–128. IEEE Computer Society Press (1996)
24. Forrest, S., Perelson, A., Allen, L., Cherukuri, R. Self-nonself discrimination in a computer. In: Proceedings of the IEEE Symposium on Security and Privacy, pp. 202–209. IEEE Computer Society (1994)
25. Goldberg, D. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Longman Publishing Co., Inc., (1989)

26. Greensmith, J. The Dendritic Cell Algorithm. Ph.D. thesis, School of Computer Science, University Of Nottingham (2007)
27. Greensmith, J., Aickelin, U. Dendritic cells for syn scan detection. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2007), pp. 49–56 (2007)
28. Greensmith, J., Aickelin, U. The deterministic dendritic cell algorithm. In: Proceedings of the 7th International Conference on Artificial Immune Systems (ICARIS), pp. 291–302. Springer (2008)
29. Greensmith, J., Aickelin, U., Feyereisl, J. The DCA-SOMe comparison: a comparative study between two biologically-inspired algorithms. *Evolutionary Intelligence: Special Issue on Artificial Immune Systems*, accepted for publication (2008)
30. Greensmith, J., Aickelin, U., Twycross, J. Articulation and clarification of the Dendritic Cell Algorithm. In: Proceedings of the 5th International Conference on Artificial Immune Systems (ICARIS), LNCS 4163, pp. 404–417 (2006)
31. Hofmeyr, S. An immunological model of distributed detection and its application to computer security. Ph.D. thesis, University Of New Mexico (1999)
32. Hofmeyr, S., Forrest, S. Immunity by design. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), pp. 1289–1296 (1999)
33. Hunt, J., Timmis, J., Cooke, J., King, C. Jisys: the development of an artificial immune system for real world applications. In: Dasgupta, D. (ed.) *Applications of Artificial Immune Systems*, pp. 157–186. Springer (1998)
34. Janeway, C. Approaching the asymptote? Evolution and revolution in immunology. *Cold Spring Harbor Symposium on Quant Biology*, **1**: 1–13 (1989)
35. Jerne, N. Towards a network theory of the immune system. *Ann. Immunol. (Inst. Pasteur)*, **125 C** 373–389 (1974)
36. Ji, Z., Dasgupta, D. Revisiting negative selection algorithms. *Evol. Comput.* **15**(2), 223–251 (2007)
37. Kelsey, J., Timmis, J., Hone, A. Chasing chaos. In: Proceedings of the Congress on Evolutionary Computation (CEC), pp. 413–419 (2003)
38. Kephart, J., Sorkin, G., Swimmer, M., White, S. Blueprint for a computer immune system. In: *Virus Bulletin International Conference* (1999)
39. Kim, J., Bentley, P. Evaluating negative selection in an artificial immune system for network intrusion detection. In: Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), pp. 1330–1337, July (2001)
40. Michael Krautmacher and Werner Dilger. Ais based robot navigation in a rescue scenario. In: Proceedings of the 3rd International Conference on Artificial Immune Systems (ICARIS), LNCS 3239, pp. 106–118. Springer (2004)
41. Luh, G., Liu, W. Reactive immune network based mobile robot navigation. In: Proceedings of the 3rd International Conference on Artificial Immune Systems (ICARIS), LNCS 3239, pp. 119–132 (2004)
42. Matzinger, P. Tolerance, danger and the extended family. *Ann. Rev. Immunol.* **12**:991–1045 (1994)
43. Murphy, K., Travers, P., Walport, M. *Janeway's Immunobiology*. 7th (edn.) Garland Science (2008)
44. Oates, R., Greensmith, J., Aickelin, U., Garibaldi, J., Kendall, G. The application of a dendritic cell algorithm to a robotic classifier. In: Proceedings of the 6th International Conference on Artificial Immune Systems (ICARIS), LNCS 4628, pp. 204–215. Springer, Santos (2007)
45. Owens, N., Timmis, J., Greensted, A., Tyrrell, A. Modelling the tunability of early t cell signalling events. In: Proceedings of the 7th International Conference on Artificial Immune Systems (ICARIS), pp. 12–23. Springer, Phuket (2008)
46. Rajewsky, K. Clonal selection and learning in the antibody system. *Nature* **381**(6585), 751–758 (1996)
47. Salazar-Bañuelos, A. Immune responses: a stochastic model. In: Proceedings of the 7th International Conference on Artificial Immune Systems (ICARIS), pp. 24–35. Springer, Phuket (2008)

48. Silverstein, A. Cellular versus humoral immunology: a century-long dispute. *Nat. Immunol.* **4**(5):425–428 (2003)
49. Silverstein, A. Paul Ehrlich, archives and the history of immunology. *Nat. Immunol.* **6**(7):639–639 (2005)
50. Smith, R., Forrest, S., Perelson, A. Searching for diverse, cooperative subpopulations with Genetic Algorithms. *Evol. Comput.* **1**(2):127–149 (1993)
51. Stepney, S., Smith, R., Timmis, J., Tyrrell, A. Towards a conceptual framework for artificial immune systems. In: Proceedings of the 3rd International Conference on Artificial Immune Systems (ICARIS), LNCS 3239, pp. 53–64. Springer, Catania (2004)
52. Stibor, T., Timmis, J., Eckert, C. On permutation masks in hamming negative selection. In: Proceedings of the 5th International Conference on Artificial Immune Systems (ICARIS), LNCS 4163, pp. 122–135 Springer, Banff (2006)
53. Suzuki, J., Yamamoto, Y. Building an artificial immune network for decentralised policy negatioation in a communication end system: Open webserver/inexus study. In: Proceedings of the 4th World Conference on Systemics, Cybernetics and Informatics (SCI). International Institute of Informatics and Systemics Online Publications, Orlando, FL (2000)
54. Timmis, J. Artificial immune systems: today and tomorrow. *Nat. Comput.* **6**(1):1–18 (2007)
55. Timmis, J., Neal, M. A resource limited artificial immune system for data analysis. In: Proceedings of ES2000 Research and Development in Intelligent Systems XVII, pp. 19–32. Springer, London (2000)
56. Twycross, J. Integrated Innate and Adaptive Artificial Immune Systems Applied to Process Anomaly Detection. Ph.D. thesis, University Of Nottingham (2007)
57. Twycross, J., Aickelin, U. libtissue—implementing innate immunity. In: Proceedings of the Congress on Evolutionary Computation (CEC), pp. 499–506. IEEE Computer Socitey, Vancouver (2006)
58. Vargas, P., de Castro, L., Von Zuben, F. Mapping artificial immune systems into learning classifier systems. In: Proceedings of the 5th International Workshop on Learning Classifier Systems (IWLCS), pp. 163–186. Springer, Grenada (2002)
59. Watanabe, Y., Ishiguro, A., Shirai, Y., Uchikawa, Y. Emergent construction of behaviour arbitration mechanism based on the immune system. In: Proceedings of the IEEE International Conference on Evolutionary Computation (ICEC), pp. 481–486. IEEE, Anchorage, AK (1998)
60. Watkins, A., Timmis, J., Boggess, L. Artificial immune recognition system (AIRS): an immune-inspired supervised learning algorithm. *Genet. Prog. Evolv. Mach.*, **5**(3), 291–317 (2004)
61. Whitbrook, A.M., Aickelin, U., Garibaldi, J.M., Idiotypic immune networks in mobile robot control. *IEEE Trans. Sys., Man Cybernetics- Part B: Cybernetics*, **37**(6):1581–1598 (2007)
62. Whitbrook, A.M., Aickelin, U., Garibaldi, J.M. An idiotypic immune network as a short-term learning architecture for mobile robots. In: Proceedings of the 7th International Conference on Artificial Immune Systems (ICARIS), LNCS 5132, pp. 266–278. Springer, Phuket (2008)
63. Wilson, W., Birkin, P., Aickelin, U. Motif detection inspired by immune memory. In: Proceedings of the 6th International Conference on Artificial Immune Systems (ICARIS), LNCS 4628, pp. 276–287. Springer, Santos (2007)

# Chapter 15

## A Classification of Hyper-heuristic Approaches

Edmund K. Burke, Matthew Hyde, Graham Kendall, Gabriela Ochoa,  
Ender Özcan, and John R. Woodward

**Abstract** The current state of the art in hyper-heuristic research comprises a set of approaches that share the common goal of automating the design and adaptation of heuristic methods to solve hard computational search problems. The main goal is to produce more generally applicable search methodologies. In this chapter we present an overview of previous categorisations of hyper-heuristics and provide a unified classification and definition, which capture the work that is being undertaken in this field. We distinguish between two main hyper-heuristic categories: heuristic *selection* and heuristic *generation*. Some representative examples of each category are discussed in detail. Our goals are to clarify the mainfeatures of existing techniques and to suggest new directions for hyper-heuristic research.

### 15.1 Introduction

The current state of the art in hyper-heuristic research comprises a set of approaches that share the common goal of automating the design and adaptation of heuristic methods in order to solve hard computational search problems. The motivation behind these approaches is to raise the level of generality at which search methodologies can operate [6]. The term hyper-heuristic was first used in 1997 [21] to describe a protocol that combines several Artificial Intelligence methods in the context of automated theorem proving. The term was independently used in 2000 [18] to describe ‘heuristics to choose heuristics’ in the context of combinatorial optimisation. In this context a hyper-heuristic is a high-level approach that, given a particular problem instance and a number of low-level heuristics, can select and apply an

---

Edmund K. Burke, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan  
and John R. Woodward

Automated Scheduling, Optimisation and Planning (ASAP) Group, School of Computer Science,  
University of Nottingham, Nottingham, UK

e-mail: {ekb,mvh,gxk,gxo,exo,jrw}@cs.nott.ac.uk

appropriate low-level heuristic at each decision point [6, 50]. The idea of automating the heuristic design process, however, is not new. Indeed, it can be traced back to the early 1960s [20, 26, 27] and was independently developed by a number of authors during the 1990s [24, 32, 33, 43, 58, 62]. Some historical notes and a brief overview of early approaches can be found in [6] and [52], respectively. A more recent research trend in hyper-heuristics attempts to automatically *generate* new heuristics suited to a given problem or class of problems. This is typically done by combining, through the use of genetic programming, for example, *components* or *building blocks* of human-designed heuristics [7].

A variety of hyper-heuristic approaches using high-level methodologies, together with a set of low-level heuristics and applied to different combinatorial problems, have been proposed in the literature. The aim of this chapter is to provide an updated version of the hyper-heuristic chapter [6] in the 2003 edition of the *Handbook of Metaheuristics*. We present an overview of previous categorisations of hyper-heuristics and provide a unified classification and definition, which capture all the work that is being undertaken in this field. Our goals are to clarify the main features of existing techniques and to suggest new directions for hyper-heuristic research.

The next section outlines previous classifications of hyper-heuristics. Section 15.3 proposes both a unified classification and a new definition of the term. Sections 15.4 and 15.5 describe the main categories of the proposed classification, giving references to work in the literature and discussing some representative examples. Finally, Section 15.6 summarises our categorisation and suggests future research directions in the area.

## 15.2 Previous Classifications

In [57], hyper-heuristics are categorised into two types: (i) with learning and (ii) without learning. Hyper-heuristics without learning include approaches that use several heuristics (neighbourhood structures), but select the heuristics to call according to a predetermined sequence. Therefore, this category contains approaches such as variable neighbourhood search [42]. The hyper-heuristics with learning include methods that dynamically change the preference of each heuristic based on their historical performance, guided by some learning mechanism. As discussed in [57], hyper-heuristics can be further classified with respect to the learning mechanism employed, and a distinction is made between approaches which use a genetic algorithm from those which use other mechanisms. This is because many hyper-heuristics so far have been based on genetic algorithms. In these genetic algorithm-based hyper-heuristics the idea is to evolve the solution methods, not the solutions themselves.

In [2], hyper-heuristics are classified into those which are *constructive* and those which are *local search* methods. This distinction is also mentioned by Ross [52]. Constructive hyper-heuristics build a solution incrementally by adaptively selecting heuristics, from a pool of constructive heuristics, at different stages of the

construction process. Local search hyper-heuristics, on the other hand, start from a complete initial solution and iteratively select, from a set of neighbourhood structures, appropriate heuristics to lead the search in a promising direction.

When genetic programming started being used for hyper-heuristic research in the late 2000s (see [7] for an overview), a new class of hyper-heuristics emerged. This new class was explicitly and independently mentioned in [1] and [10]. In the first class of heuristics, or ‘heuristics to choose heuristics’, the framework is provided with a set of pre-existing, generally widely known heuristics for solving the target problem. In contrast, in the second class, the aim is to generate new heuristics from a set of *building blocks* or *components* of known heuristics, which are given to the framework. Therefore, the process requires, as in the first class of hyper-heuristics, the selection of a suitable set of heuristics known to be useful in solving the target problem. But, instead of supplying these directly to the framework, the heuristics are first decomposed into their basic components. Genetic programming hyper-heuristic researchers [1, 7, 10] have also made the distinction between ‘disposable’ and ‘reusable’ heuristics. A disposable heuristic is created just for one problem and is not intended for use on unseen problems. Alternatively, the heuristic may be created for the purpose of re-using it on new unseen problems of a certain class.

In [16], hyper-heuristics are classified into four categories: (i) hyper-heuristics based on the random choice of low-level heuristics, (ii) greedy and peckish hyper-heuristics, which requires preliminary evaluation of all or a subset of the heuristics in order to select the best performing one, (iii) metaheuristics-based hyper-heuristics, and (iv) hyper-heuristics employing learning mechanisms to manage low-level heuristics.

### 15.3 The Proposed Classification and New Definition

Building upon some of the previous classifications discussed above and realising that hyper-heuristics lie at the interface of optimisation and machine learning research, we propose a general classification of hyper-heuristics according to two dimensions: (i) the nature of the heuristic search space and (ii) the source of feedback during learning. These dimensions are orthogonal in that different heuristic search spaces can be combined with different sources of feedback and thus different machine learning techniques.

We consider that the most fundamental hyper-heuristic categories from the previous classifications are those represented by the processes of

- **Heuristic selection:** methodologies for choosing or selecting existing heuristics
- **Heuristic generation:** methodologies for generating new heuristics from components of existing heuristics

There is no reason why the higher level strategy (for selecting or generating heuristics) should be restricted to be a heuristic. Indeed, sophisticated knowledge-based techniques such as case-based reasoning have been employed in this way

with good results for university timetabling [14]. This leads us to propose the following more general definition of the term ‘hyper-heuristic’ which is intended to capture the idea of a method for automating the heuristic design and selection process:

A hyper-heuristic is an automated methodology for selecting or generating heuristics to solve hard computational search problems

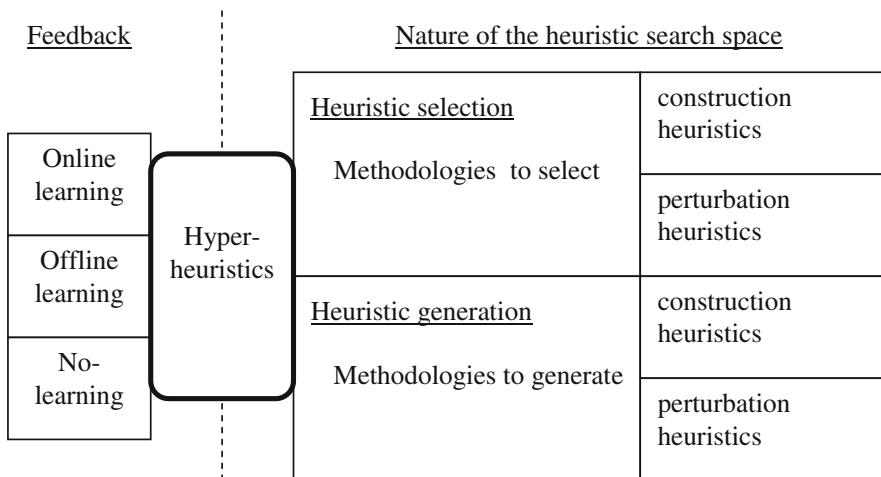
From this definition, there are two clear categories of hyper-heuristics: heuristic selection and heuristic generation, which form the first branch in our first dimension (the nature of the search space). The second level in this dimension corresponds to the distinction between constructive and local search hyper-heuristics, also discussed in Section 15.2. Notice that this categorisation is concerned with the nature of the low-level heuristics used in the hyper-heuristic framework. Our classification uses the terms *construction* and *perturbation* to refer to these classes of low-level heuristics. Sections 15.4 and 15.5 describe these categories in more detail, discussing some concrete examples of recent approaches that can be found in the literature.

We consider a hyper-heuristic to be a learning algorithm when it uses some feedback from the search process. Therefore, non-learning hyper-heuristics are those that do not use any feedback. According to the source of the feedback during learning, we propose a distinction between *online* and *offline* learning. Notice that in the context of heuristic generation methodologies, an example of which is genetic programming-based hyper-heuristics (discussed in Section 15.2), the notions of *disposable* and *reusable* have been used to refer to analogous ideas to those of online and offline learning, as described:

**Online learning hyper-heuristics:** The learning takes place whilst the algorithm is solving an instance of a problem. Therefore, task-dependent local properties can be used by the high-level strategy to determine the appropriate low-level heuristic to apply. Examples of online learning approaches within hyper-heuristics are the use of reinforcement learning for heuristic selection and, generally, the use of metaheuristics as high-level search strategies across a search space of heuristics.

**Offline learning hyper-heuristics:** The idea is to gather knowledge in the form of rules or programs, from a set of training instances, that would hopefully generalise to the process of solving unseen instances. Examples of offline learning approaches within hyper-heuristics are learning classifier systems, case-base reasoning, and genetic programming.

The proposed classification of hyper-heuristic approaches can be summarised as follows (see also Figure 15.1):



**Fig. 15.1** A classification of hyper-heuristic approaches, according to two dimensions: (i) the nature of the heuristic search space and (ii) the source of feedback during learning.

- With respect to the nature of the heuristic search space:
  - **Heuristic selection methodologies:** Produce combinations of pre-existing
    - Construction heuristics
    - Perturbation heuristics
  - **Heuristic generation methodologies:** Generate new heuristic methods using basic components (building blocks) of
    - Construction heuristics
    - Perturbation heuristics
- With respect to the source of feedback during learning:
  - **Online learning hyper-heuristics:** Learn whilst solving a given instance of a problem
  - **Offline learning hyper-heuristics:** Learn, from a set of training instances, a method that would generalise to unseen instances
  - **No-learning hyper-heuristics:** Do not use feedback from the search process

Note that these categories describe current research trends. There is, however, nothing to stop the exploration of hybrid methodologies that combine, for example, construction with perturbation heuristics or heuristic selection with heuristic generation methodologies. These hybrid approaches are already starting to emerge.

## 15.4 Heuristic Selection Methodologies

This section is not intended to be an exhaustive survey. The intention is to present a few examples to give the reader a flavour of the research that has been undertaken in this area.

## 15.4.1 Approaches Based on Construction Low-Level Heuristics

These approaches build a solution incrementally. Starting with an empty solution, the goal is to intelligently select and use construction heuristics to gradually build a complete solution. The hyper-heuristic framework is provided with a set of pre-existing (generally problem specific) construction heuristics and the challenge is to select the heuristic that is somehow the most suitable for the current problem state. This process continues until the final state (a complete solution) is obtained. Notice that there is a natural end to the construction process, that is, when a complete solution is reached. Therefore the sequence of heuristic choices is finite and determined by the size of the underlying combinatorial problem. Furthermore, there is, in this scenario, the interesting possibility of learning associations between partial solution stages and adequate heuristics for those stages.

Several approaches have been recently proposed to generate efficient hybridisations of existing construction heuristics in domains such as bin packing [40, 55], timetabling [13, 14, 53, 54], production scheduling [63], and stock cutting [60, 63]. Both online and offline machine learning approaches have been investigated. Examples of online approaches are the use of metaheuristics in a search space of construction heuristics. For example, genetic algorithms [25, 33, 62, 63], tabu search [13], and other single point-based search strategies [51]. For this type of hyper-heuristics, recent research is starting to explore the structure of the heuristic search space or hyper-heuristic landscape, in both timetabling [45] and production scheduling [46]. Examples of offline techniques are the use of learning classifier systems [41, 55], messy genetic algorithms [50, 53, 61], and case-based reasoning [14].

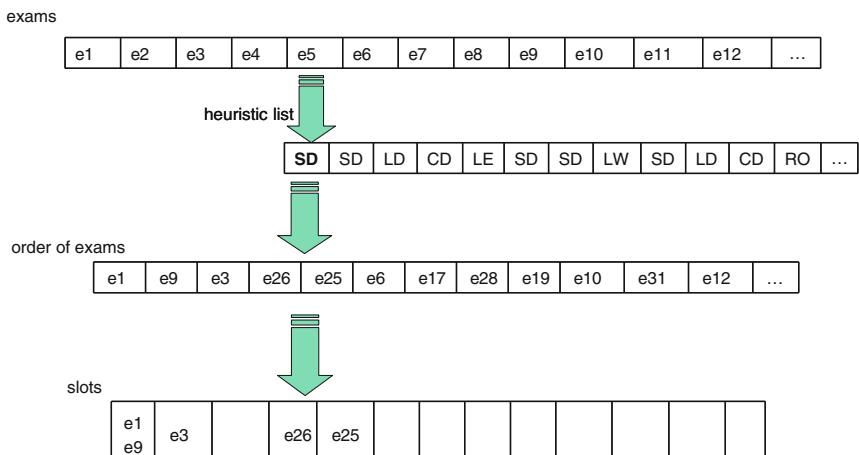
### 15.4.1.1 Representative Examples

Two hyper-heuristics based on construction heuristics are described here in more detail. The first approach is online and is based on graph-colouring heuristics for timetabling problems, whilst the second is offline and is based on bin packing heuristics.

**Graph-colouring hyper-heuristic for timetabling:** In educational timetabling, a number of courses or exams need to be assigned to a number of timeslots, subject to a set of both hard and soft constraints. Timetabling problems can be modelled as graph colouring problems, where nodes in the graph represent events (e.g. exams), and edges represent conflicts between events. Graph heuristics in timetabling use the information in the graph to order the events by their characteristics (e.g. number of conflicts with other events or the degree of conflict) and assign them one by one into the timeslots. These characteristics suggest how difficult it is to schedule the events. Therefore, the most difficult event, according to the corresponding ordering strategy, will be assigned first. The graph-based hyper-heuristic developed in [13] implements the following five graph colouring-based heuristics, plus a random-ordering heuristic:

- *Largest degree (LD)*: Orders the events decreasingly based on the number of conflicts the event has with the others events.
- *Largest weighted degree (LW)*: The same as LD, but the events are weighted by the number of students involved.
- *Colour degree (CD)*: Orders the events decreasingly in terms of the number of conflicts (events with common students involved) each event has with those already scheduled.
- *Largest enrolment (RO)*: Orders the events decreasingly based on the number of enrolments.
- *Saturation degree (SD)*: Orders the events increasingly based on the number of timeslots available for each event in the timetable at that time.

A candidate solution in the heuristic search space corresponds to a sequence (list) of these heuristics. The solution (timetable) construction is an iterative process where, at the  $i$ th iteration, the  $i$ th graph-colouring heuristic in the list is used to order the events not yet scheduled at that step, and the first  $e$  events in the ordered list are assigned to the first  $e$  least-cost timeslots in the timetable (see Figure 15.2).



**Fig. 15.2** A solution (timetable) is constructed by iteratively considering each heuristic in the list and using it to order the events not yet scheduled. The first  $e$  events (in the figure  $e = 5$ ) in the resulting ordering are assigned to the first  $e$  least-cost timeslots in the timetable.

Tabu Search is employed as the high-level search strategy for producing good sequences of the low-level heuristics. Each heuristic list produced by tabu search is evaluated by sequentially using the individual heuristics to order the unscheduled events and thus construct a complete timetable. Each heuristic in the list is used to schedule a number  $e$  of events. Therefore, the length of the heuristic list is  $n/e$  where  $n$  is the number of events to be scheduled. Values in the range of  $e = 1, \dots, 5$  were tested (details can be found in [13]). This work also highlights the existence of two search spaces in constructive hyper-heuristics (the heuristic space and the problem solution). The approach was tested on both course and exam

timetabling benchmark instances with competitive results. This graph-based hyper-heuristic was later extended in [51] where a formal definition of the framework is presented. The authors also compare the performance of several high-level heuristics that operate on the search space of heuristics. Specifically, a steepest descent method, iterated local search and variable neighbourhood search are implemented and compared to the previously implemented tabu search. The results suggest that the choice of a particular neighbourhood structure on the heuristic space is not crucial to the performance. Moreover, iterative techniques such as iterated local search and variable neighbourhood search were found more effective for traversing the heuristic search space than more elaborate metaheuristics such as tabu search. The authors suggest that the heuristic search space is likely to be smooth and to contain large plateaus (i.e. areas where different heuristic sequences can produce similar quality). The work also considers hybridisations of the hyper-heuristic framework with local search operating on the solution space. This strategy greatly improves the performance of the overall system, making it competitive with state-of-the-art approaches on the studied benchmark instances.

In a further study [45], the notion of fitness landscapes is used to analyse the search space of graph-colouring heuristics. The study confirms some observations about the structure of the heuristic search space discussed in [51]. Specifically, these landscapes have a high level of neutrality (i.e. the presence of plateaus). Furthermore, although rugged, they have the encouraging feature of a globally convex or *big valley* structure, which indicates that an optimal solution would not be isolated but surrounded by many local minima. The study also revealed a *positional bias* in the search space comprising sequences of heuristics. Specifically, changes in the earlier positions of a heuristic sequence have a larger impact on the solution quality than changes in the later positions. This is because early decisions (heuristic choices) in a construction process have a higher impact on the overall quality of the solution than later decisions.

**Classifier system hyper-heuristic for bin packing:** Classifier systems [34] are rule-based learning systems that evolve fixed length stimulus–response rules. The rules are encoded as ternary strings, made of the symbols  $\{0, 1, \#\}$ , and have an associated strength. The system operates in two phases. First, the population of classification rules is applied to some task; and second a genetic algorithm generates a new population of rules by selection based on strength, and by the application of standard genetic operators. Calculating the strength of each rule is a *credit assignment problem*, which refers to determining the contribution made by each sub-component or partial solution, in decomposable problems being solved collectively by a set of partial solutions.

In [55], a modern classifier system (accuracy-based classifier system [64]) was used, in the domain of one-dimensional bin packing, to learn a set of rules that associate characteristics of the current state of a problem with different low-level construction heuristics. In the one-dimensional bin packing problem, there is an unlimited supply of bins, each with capacity  $c$ . A set of  $n$  items are to be packed into the bins, the size of each item is given, and items must not overfill any bin. The task is to minimise the total number of bins required.

The set of rules evolved by the classifier system is used as follows: given the initial problem characteristics  $P$ , a heuristic  $H$  is chosen to pack an item, thus gradually altering the characteristics of the problem that remains to be solved. At each step a rule appropriate to the current problem state  $P'$  is selected and the process continues until all items have been packed. For the training phase a total of 890 benchmark instances from the literature were used. The authors chose four bin packing heuristics from the literature, the selection being based on those that produced the best results on the studied benchmark set. These heuristics were as follows:

- *Largest-fit-decreasing*: Items are taken in order of size, largest first, and put in the first bin where they fit (a new bin is opened if necessary and effectively all bins stay open).
- *Next-fit-decreasing*: An item is placed in the current bin if possible, or else a new bin is opened into which the piece is placed. This new bin becomes the current bin.
- *Djang and Finch's (DJD)*: A heuristic that considers combinations of up to three items to completely fill partially full bins.
- *A variation of DJD*: A variation of the previous heuristic that considers combinations of up to five items to completely fill partially full bins.

A simplified description of the current state of the problem is proposed. This description considers the number of items remaining to be packed and calculates the percentage of items in each of four size ranges (huge, large, medium and small); where the size of the items is judged in proportion to the bin size. The approach used single-step environments, meaning that rewards were available after each action had taken place. The classifier system was trained on a set of example problems and showed good generalisation to unseen problems. In [41], the classifier system approach is extended to multi-step environments. The authors test several reward schemes in combination with alternate exploration/exploitation ratios and several sizes and types of multi-step environments. Again, the approach was tested using a large set of one-dimensional benchmark bin packing problems. The classifier system was able to generalise well and create solution processes that performed well on a large set of NP-hard benchmark instances. The authors report that multi-step environments can obtain better results than single-step environments at the expense of a higher number of training cycles.

### 15.4.2 Approaches Based on Perturbation Low-Level Heuristics

These approaches start with a complete solution, generated either randomly or using simple construction heuristics, and thereafter try to iteratively improve the current solution. The hyper-heuristic framework is provided with a set of neighbourhood structures and/or simple local searchers and the goal is to iteratively select and apply them to the current complete solution. This process continues until a stopping condition has been met. Notice that these approaches differ from those based on construction heuristics, in that they do not have a natural termination condition. The

sequence of heuristic choices can, in principle, be arbitrarily extended. This class of hyper-heuristics has the potential to be applied successfully to different combinatorial optimisation problems, since general neighbourhood structures or simple local searchers can be made available. Hyper-heuristics based on perturbation have been applied to personnel scheduling [12, 18], timetabling [5, 12], shelf space allocation [3, 4], packing [23], and vehicle routing problems [50].

So far, the approaches that combine perturbation low-level heuristics in a hyper-heuristic framework use online learning, in that they attempt to adaptively solve a single instance of the problem under consideration. Furthermore, the majority of the proposed approaches are single-point algorithms, in that they maintain a single incumbent solution in the solution space. Some approaches that maintain a population of points in the heuristic space have been attempted [17].

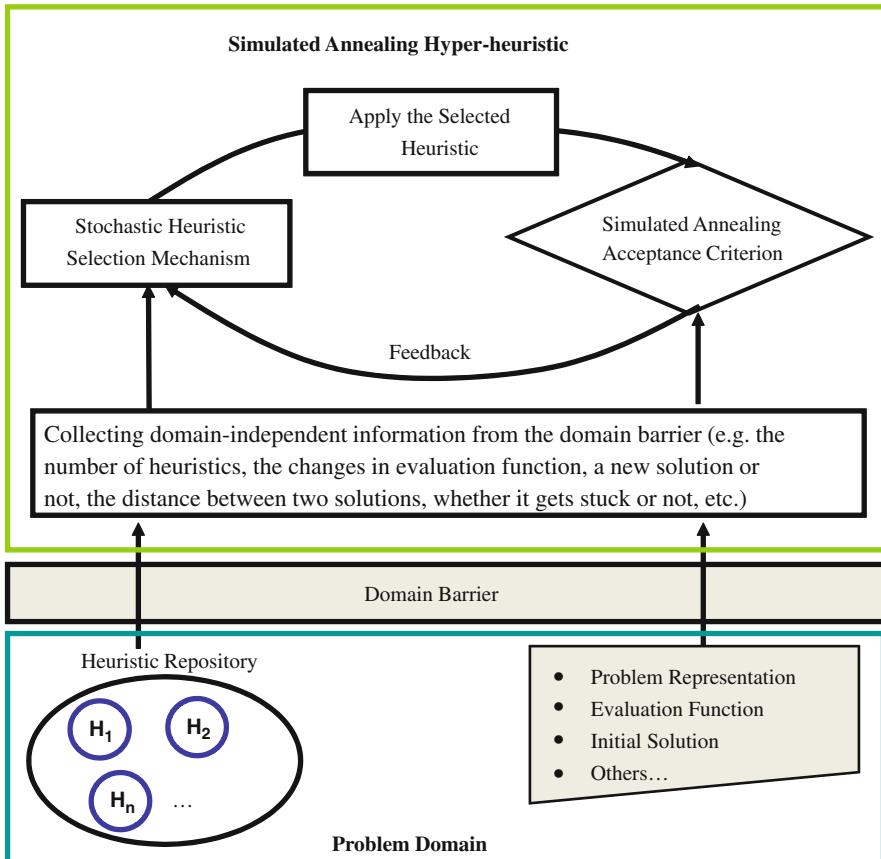
As suggested in [5, 48, 49] perturbation hyper-heuristics can be separated into two processes: (i) (low-level) heuristic selection and (ii) move acceptance strategy. The authors classify hyper-heuristics with respect to the nature of the heuristic selection and move acceptance components. The heuristic selection can be done in a *non-adaptive* (simple) way: either randomly or along a cycle, based on a prefixed heuristic ordering [18, 19]. No learning is involved in these approaches. Alternatively, the heuristic selection may incorporate an *adaptive* (or online learning) mechanism based on the probabilistic weighting of the low-level heuristics [12, 44], or some type of performance statistics [18, 19]. Both non-adaptive and adaptive heuristic selection schemes are generally embedded within a single-point local search high-level heuristic.

The acceptance strategy is an important component of any local search heuristic. Many acceptance strategies have been explored within hyper-heuristics. Move acceptance strategies can be divided into two categories: *deterministic* and *non-deterministic*. In general, a move is accepted or rejected, based on the quality of the move and the current solution during a single-point search. At any point in the search, deterministic move acceptance methods generate the same result for the same candidate solution(s) used for the acceptance test. However, a different outcome is possible if a non-deterministic approach is used. If the move acceptance test involves other parameters, such as the current time, then these strategies are referred to as non-deterministic strategies. Well-known meta-heuristic components are commonly used as non-deterministic acceptance methods, such as those of great deluge [38] and simulated annealing [4, 23].

#### 15.4.2.1 Representative Examples

Two hyper-heuristics based on perturbation heuristics are described here. The first is applied to a real-world packing problem, whilst the second uses large neighbourhood heuristics and is applied to five variants of the well-known vehicle routing problem.

**A simulated annealing hyper-heuristic for determining shipper sizes:** In [23] the tabu search hyper-heuristic, originally presented in [12], is integrated within a simulated annealing framework. That is, a simulated annealing acceptance



**Fig. 15.3** Simulated annealing hyper-heuristic framework.

strategy is combined with the previously proposed heuristic selection mechanism. Figure 15.3 outlines the simulated annealing-based hyper-heuristic.

The tabu search hyper-heuristic [12] selects the low-level heuristics according to learned utilities or ranks. The framework also incorporates a dynamic tabu list of low-level heuristics that are temporarily excluded from the selection pool. The algorithm deterministically selects the low-level heuristic with the highest rank (not included in the tabu list) and applies it once regardless of whether the selected move causes an improvement or not (all moves acceptance). If there is an improvement, the rank is increased. If the new solution is worse, the rank of the low-level heuristic is decreased and it is made tabu. The rank update scheme is additive and the tabu list is emptied each time a non-improvement move is accepted. This general tabu search approach was evaluated on various instances of two distinct timetabling and rostering (personal scheduling) problems and the obtained results were competitive with those obtained using state-of-the-art problem-specific techniques. Apart from the simulated annealing acceptance criteria, some modifications are also introduced

in [23]. In particular, a single application of a low-level heuristic  $h$  is defined to be  $k$  iterations of  $h$ . Therefore, the decision points are set for every  $k$  iterations, and the feedback for updating the quality of heuristic  $h$  is based on the best cost obtained during those  $k$  iterations. Additionally, a nonmonotonic cooling schedule is proposed as a means to deal with the effects of having different neighbourhood sizes (given by the pool of low-level heuristics used). The methodology was applied to a packing problem in the cosmetics industry, where the shipper sizes for storage and transportation had to be determined. Real data were used for generating the instances and the approach was compared with a simpler local search strategy (random descent), with favourable results.

**A general heuristic for vehicle routing problems:** In [50], a unified methodology is presented, which is able to solve five variants of the vehicle routing problem: the vehicle routing problem with time windows, the capacitated vehicle routing problem, the multi-depot vehicle routing problem, the site-dependent vehicle routing problem and the open vehicle routing problem. All problem variants are transformed into a rich pickup and delivery model and solved using an adaptive large neighbourhood search methodology (ALNS), which extends a previous framework presented in [56]. ALNS can be based on any local search framework, e.g. simulated annealing, tabu search or guided local search. The general framework is outlined in Figure 15.4, where the repeat loop corresponds to the local search framework at the master level. Implementing a simulated annealing algorithm is straightforward as one solution is sampled in each iteration of the ALNS. In each iteration of the main loop, the algorithm chooses one destroy ( $N-$ ) and one repair neighbourhood ( $N+$ ). An adaptive layer stochastically controls which neighbourhoods to choose according to their past performance (score,  $P_i$ ). The more a neighbourhood  $N_i$  has contributed to the solution process, the larger score  $P_i$  it obtains, and hence it has a larger probability of being chosen. The adaptive layer uses roulette wheel selection for choosing a destroy and a repair neighbourhood.

The pickup and delivery model is concerned with serving a number of transportation requests using a limited number of vehicles. Each request involves moving a number of goods from a pickup location to a delivery location. The task is to construct routes that visit all locations such that the corresponding pickups and

```

Construct a feasible solution x; set x*:=x
Repeat
    Choose a destroy and a repair neighbourhood: N- and N+
    using roulette wheel selection based on previously obtained scores (Pi)
    Generate a new solution x' from x using the heuristics
        corresponding to the chosen destroy and repair neighbourhoods
    If x' can be accepted then set x:=x'
    Update scores Pi of N- and N+
    If f(x) < f(x*) set x*:=x
Until a stopping criteria is met
return x*

```

**Fig. 15.4** Outline of the Adaptive Large Neighbourhood framework.  $N-$  and  $N+$  correspond to destroy and repair neighbourhoods respectively, whilst  $P_i$  stands for the score associated to the heuristic  $i$ .

deliveries are placed on the same route and such that a pickup is performed before the corresponding delivery. Different constraints are added to model the different problem variants. The proposed framework adaptively chooses among a number of insertion and removal heuristics to intensify and diversify the search. These competing sub-heuristics are selected with a frequency corresponding to their historic performance (stored as learned weights for each heuristic). The approach uses a simulated annealing acceptance strategy with a standard exponential cooling rate. A large number of tests were performed on standard benchmarks from the literature on the five variants of the vehicle routing problem. The results proved to be highly promising, as the methodology was able to improve on the best-known solutions of over one-third of the tested instances.

## 15.5 Heuristic Generation Methodologies

This section provides some examples of approaches that have the potential to automatically generate heuristics for a given problem. Many of the approaches in the literature to generate heuristics use genetic programming [7], a branch of evolutionary computation concerned with the automatic generation of computer programs [40]. Besides the particular representation (using trees as chromosomes<sup>1</sup>), it differs from other evolutionary approaches in its application area. Whilst most applications of evolutionary algorithms deal with optimisation problems, genetic programming could instead be positioned in the field of machine learning. Genetic programming has been successfully applied to the automated generation of heuristics that solve hard combinatorial optimisation problems, such as boolean satisfiability, [1, 28–30, 39], bin packing [8, 9, 11], travelling salesman problem [36, 37], and production scheduling [22, 31, 59].

Some genetic programming-based hyper-heuristics have evolved local search heuristics [1, 29, 30, 36, 37] or even evolutionary algorithms [47]. An alternative idea is to use genetic programming to evolve a program representing a function, which is part of the processing of a given problem-specific construction heuristic [8, 9, 11, 22, 31, 59]. Most examples of using genetic programming as a hyper-heuristic are offline in that a training set is used for generating a program that acts as a heuristic, which is thereafter used on unseen instances of the same problem. That is, the idea is to generate *reusable* heuristics. However, research on *disposable* heuristics has also been conducted [1, 36, 37]. In other words, heuristics are evolved for solving a single instance of a problem. This approach is analogous to the on-line heuristic selection methodologies discussed in Section 15.4, except that a new heuristic is generated for each instance, instead of choosing a sequence of heuristics from a predefined set.

---

<sup>1</sup> According to the genetic programming literature, programs can be represented in ways other than trees. Research has already established the efficacy of both linear and graph-based genetic programming systems.

The adaptation of heuristic orderings can also be considered as a methodology for heuristic generation. The adaptive approach proposed in [15] starts with one heuristic and adapts it to suit a particular problem instance ‘on the fly’. This method provides an alternative to existing forms of ‘backtracking’, which are often required to cope with the possible unsuitability of a heuristic. The adaptive method is more general, significantly easier to implement and produces results that are at least comparable (if not better) than the current state-of-the-art examination timetabling algorithms.

### 15.5.1 Representative Examples

We discuss two representative examples of heuristic generation using genetic programming. The first evolves packing heuristics that operate on a constructive framework, whilst the second evolves complete local search algorithms, using components of successful, existing local search heuristics, for boolean satisfiability.

**Generation of construction heuristics for bin packing:** As mentioned earlier, the one-dimensional bin packing problem involves a set of integer pieces  $L$ , which must be packed into bins of a certain capacity  $C$ , using the minimum number of bins possible. In the online version of the problem, the number of pieces and their sizes are not known in advance. This is in contrast to the offline version of the problem where the set of items to be packed is available at the start. An example of a construction heuristic used in online bin packing is first-fit, which packs a set of pieces one at a time, in the order that they are presented. The heuristic iterates through the open bins and the current piece is placed in the first bin into which it fits.

In [8, 9, 11], construction heuristics are evolved for the online bin packing problem. The evolved heuristics, represented as trees (see Figure 15.5 for an example) operate within a fixed framework that resembles the operation of the first-fit heuristic discussed above. The key idea is to use the attributes of the pieces and bin capacities that represent the state of the problem, in order to evolve functions (expressions) that would direct the process of packing. Each evolved function (GP tree) is applied in turn to the available bins, returning a value. If the value is zero or less then the system moves on to the next bin, but if the value is positive the piece is packed into the bin. In this way, it is the expression which decides when to stop the search for a suitable bin and place the piece. The algorithm (depicted in Figure 15.6)

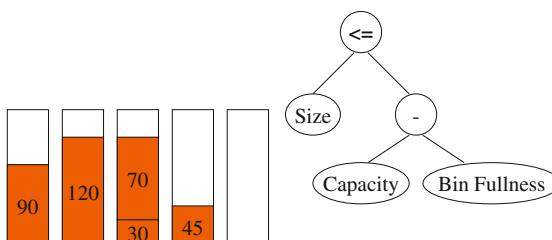


Fig. 15.5 Evolving one-dimensional packing heuristics with genetic programming.

```

For each piece
  For each bin b
    output := evaluate Heuristic
    If (output > 0)
      place piece p in bin b
      break
    End If
  End For
End For

```

**Fig. 15.6** Pseudo code showing the overall program structure within which an evolved packing heuristic operates.

then repeats the process for each of the other pieces until all the pieces have been packed.

In a genetic programming framework, the set of terminals and functions need to be specified. The hyper-heuristic framework for online bin packing uses some attributes that describe the state of the problem to define the terminals. In [8, 9], the authors use the following terminals:

- $S$  the size of the current item,
- $C$  the capacity of a bin (this is a constant for the problem) and
- $F$  the fullness of a bin (i.e. the total size of all of the items occupying that bin).

Later [11], these three attributes were replaced by two:  $S$ , the size of the current item, and  $E (= C - F)$  the emptiness of a bin (i.e. how much space is remaining in the bin or how much more space can be allocated to it before it exceeds its capacity). The function set used in [8, 9] consisted of  $\leq, +, -, \times, \%$ , where  $\%$  is the ‘protected divide function’[40]. The results in [9] show that a simple genetic programming system can discover human designed heuristics such as first-fit, whilst in [8, 11], heuristics that outperformed first-fit were evolved. In [8], it was also shown empirically that the choice of the training instances (categorised according to the piece size distribution), impacts on the trade-off between the performance and generality of the heuristics generated and their applicability to new problems.

**Generation of local search heuristics for satisfiability testing:** The boolean satisfiability problem consists of finding the true/false assignments of a set of boolean variables, to decide if a given propositional formula or expression (in conjunctive normal form) can be satisfied. The problem, denoted as SAT, is a classic NP-complete problem.

In [28–30] a genetic programming system, named CLASS (Composite Learned Algorithms for SAT Search), is proposed which automatically discovers new SAT local search heuristics. Figure 15.7 illustrates a generic SAT local search algorithm, where the ‘key detail’ is the choice of a *variable selection heuristic* in the inner loop. Much research in the past decade has focused on designing a better variable selection heuristic, and as a result, local search heuristics have improved dramatically since the original method. The CLASS system was developed in order to automatically discover variable selection heuristics for SAT local search. It was noted in [28] that many of the best-known SAT heuristics (such as GSAT, HSAT, Walksat, and Novelty [30]) could be expressed as decision tree-like combinations

```

A:= randomly generated truth assignment
For j:= 1 to termination condition
    If A satisfies formula then return A
    v:= Choose variable using
        "variable selection heuristic"
    A:= A with value of v inverted
End If
End For
return FAILURE (no assignment found)

```

**Fig. 15.7** A generic SAT local search algorithm. The “variable selection heuristic” is replaced by the evolved function.

of a set of primitives. Thus, it should be possible for a machine learning system to automatically discover new, efficient variable selection heuristics by exploring the space of combinations of these primitives. Examples of the primitives used in human-designed SAT heuristics are the gain obtained by flipping a variable (i.e. the increase in the number of satisfied clauses in the formula) or the age of a variable (i.e. how long since it was last flipped).

The results using CLASS [30] show that a simple genetic programming system is able to generate local search heuristics that are competitive with efficient implementations of state-of-the-art heuristics (e.g. Walksat and Novelty variants), as well as previous evolutionary approaches. The evolved heuristics scale and generalise fairly well on random instances as well as on more structured problem classes.

## 15.6 Summary and Discussion

The defining feature of hyper-heuristic research is that it investigates methodologies that operate on a search space of heuristics rather than directly on a search space of problem solutions. This feature provides the potential for increasing the level of generality of search methodologies. Several hyper-heuristic approaches have been proposed that incorporate different search and machine learning paradigms. We have suggested an updated definition of the term ‘hyper-heuristic’ to reflect recent work in the area.

With the incorporation of genetic programming [40], and other methods such as *squeaky wheel* optimisation [35], into hyper-heuristic research, a new class of approaches can be identified, that is, heuristic generation methodologies. These approaches provide richer heuristic search spaces and thus the freedom to create new methodologies for solving the underlying combinatorial problems. However, they are more difficult to implement, than their counterpart, heuristic selection methodologies, since they require the decomposition of existing heuristics and the design of an appropriate framework. We have further categorised the two main classes of hyper-heuristics (*heuristic selection* and *heuristic generation*), according to whether they use *construction* or *perturbation* low-level heuristics. These categories describe current research trends. However, the possibilities are open for the exploration of hybrid approaches.

We also considered an additional orthogonal criterion for classifying hyper-heuristics with respect to the source of the feedback during the learning process, which can be either one instance (online approaches) or many instances of the underlying problem (offline approaches). Both online and offline approaches are potentially useful and therefore worth investigating. Although having a reusable method will increase the speed of solving new instances of problems, using online (or disposable) methods can have other advantages. In particular, searching over a space of heuristics may be more effective than directly searching the underlying problem space, as heuristics may provide an advantageous search space structure. Moreover, in newly encountered problems there may not be a set of related instances on which to train offline hyper-heuristic methods.

Hyper-heuristic research lies in the interface between search methodologies and machine learning methods. Machine learning is a well-established artificial intelligence sub-field with a wealth of proven tools. The exploration of these techniques for automating the design of heuristics is only in its infancy. We foresee increasing interest in these methodologies in the coming years.

## References

1. Bader-El-Den, M., Poli, R.: Generating sat local-search heuristics using a gp hyper-heuristic framework. In Proceedings of Artificial Evolution (EA'07). Tours, France, Springer, Berlin (2007)
2. Bai, R.: An Investigation of Novel Approaches for Optimising Retail Shelf Space Allocation. Ph.D. thesis, School of Computer Science and Information Technology, University of Nottingham, September (2005)
3. Bai, R., Burke, E.K., Kendall, G.: Heuristic,meta-heuristic and hyper-heuristic approaches for fresh produce inventory control and shelf space allocation. *J. Oper. Res. Soc.* **59**, 1387–1397 (2008)
4. Bai, R., Kendall, G.: An investigation of automated planograms using a simulated annealing based hyper-heuristics. In: Ibaraki, T., Nonobe, K., Yagiura, M. (eds.) *Metaheuristics: Progress as Real Problem Solver—(Operations Research/Computer Science Interface Series, vol. 32)*, pp. 87–108. Springer, Berlin (2005)
5. Bilgin, B., Özcan, E., Korkmaz, E.E.: An experimental study on hyper-heuristics and exam timetabling. Proceedings of the 6th Practice and Theory of Automated Timetabling (PATAT 2006). Lecture Notes in Computer Science, vol. 3867, pp. 394–412. Springer, Heidelberg (2007)
6. Burke, E.K., Hart, E., Kendall, G., Newall, J., Ross, P., Schulenburg, S.: Hyper-heuristics: an emerging direction in modern search technology. In: Glover, F., Kochenberger, G. (eds.) *Handbook of Metaheuristics*, pp. 457–474. Kluwer (2003)
7. Burke, E.K., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Woodward, J.: Exploring hyper-heuristic methodologies with genetic programming. In: Mumford C., Jain, L. (eds.) *Collaborative Computational Intelligence*, pp. 177–201. Springer, Berlin (2009)
8. Burke, E.K., Hyde, M.R., Kendall, G., Woodward, J.: Automatic heuristic generation with genetic programming: evolving a jack-of-all-trades or a master of one. Proceedings of 9th ACM Genetic and Evolutionary Computation Conference (GECCO'07), London, UK, July 2007, pp. 1559–1565. ACM, New York (2007)
9. Burke, E.K., Hyde, M.R., Kendall, G.: Evolving bin packing heuristics with genetic programming. In Proceedings of the 9th International Conference on Parallel Problem Solving from

- Nature (PPSN 2006), Reykjavik, Iceland vol. 4193 of Lecture Notes in Computer Science, vol. 4193 pp. 860–869, Springer, Berlin September. (2006)
- 10. Burke, E.K., Hyde, M.R., Kendall, G., Woodward, J.: A genetic programming hyper-heuristic approach for evolving two dimensional strip packing heuristics. *IEEE Trans. Evol. Comput.* accepted, to appear (2010)
  - 11. Burke, E.K., Hyde, M.R., Kendall, G., Woodward, J.R.: The scalability of evolved on line bin packing heuristics. In 2007 IEEE Congress on Evolutionary Computation, pp. 2530–2537. Singapore IEEE Computational Intelligence Society, IEEE Press (2007)
  - 12. Burke, E.K., Kendall, G., Soubeiga, E.: A tabu-search hyperheuristic for timetabling and rostering. *J. Heuristics* **9**(6), 451–470 (2003)
  - 13. Burke, E.K., McCollum, B., Meisels, A., Petrovic, S., Qu, R.: A graph-based hyper-heuristic for educational timetabling problems. *Euro. J. Oper. Res.* **176**, 177–192 (2007)
  - 14. Burke, E.K., Petrovic, S., Qu, R.: Case based heuristic selection for timetabling problems. *J. Scheduling* **9**(2), 115–132 (2006)
  - 15. Burke, E.K., Newall, J.: Solving examination timetabling problems through adaptation of heuristic orderings. *Ann. Oper. Res.* **129**, 107–134 (2004)
  - 16. Chakhlevitch, K., Cowling, P.I.: Hyperheuristics: recent developments. In: Cotta, C., Sevaux, M., Sørensen, K. eds. *Adaptive and Multilevel Metaheuristics*, Studies in Computational Intelligence, vol. 136, pp. 3–29. Springer, Berlin (2008)
  - 17. Cowling, P., Kendall, G., Han, L.: An investigation of a hyperheuristic genetic algorithm applied to a trainer scheduling problem. In: Proceedings of the Congress on Evolutionary Computation (CEC2002), pp. 1185–1190. Hilton Hawaiian Village Hotel, Honolulu, Hawaii, USA, 12–17 May (2002)
  - 18. Cowling, P., Kendall, G., Soubeiga, E.: A hyperheuristic approach for scheduling a sales summit. In: Selected Papers of the Third International Conference on the Practice And Theory of Automated Timetabling, PATAT 2000, Konstanz, Germany, Lecture Notes in Computer Science, pp. 176–190. Springer August (2000)
  - 19. Cowling, P., Kendall, G., Soubeiga, E.: Hyperheuristics: a tool for rapid prototyping in scheduling and optimisation. In: Cagnoni, S., Gottlieb, J., Hart, E., Middendorf, M., Goenther, R., (eds.) *Applications of Evolutionary Computing: Proceeding of Evo Workshops 2002*, Kinsale, Ireland, Lecture Notes in Computer Science, vol. 2279, pp. 1–10. Springer April 3–4 (2002)
  - 20. Crowston, W.B., Glover, F., Thompson, G.L., Trawick, J.D.: Probabilistic and parametric learning combinations of local job shop scheduling rules. ONR Research Memorandum, Carnegie-Mellon University, Pittsburgh, PA (1963)
  - 21. Denzinger, J., Fuchs, M., Fuchs, M.: High performance ATP systems by combining several ai methods. In: Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI 97), pp. 102–107. Morgan Kaufmann, CA, USA (1997)
  - 22. Dimopoulos, C., Zalzala, A.M.S.: Investigating the use of genetic programming for a classic one-machine scheduling problem. *Advances in Engineering Software*, **32**(6), 489–498 (2001)
  - 23. Dowsland, K.A., Soubeiga, E., Burke, E.K.: A simulated annealing hyper-heuristic for determining shipper sizes. *Euro. J. Oper. Res.* **179**(3), 759–774 (2007)
  - 24. Fang, H.L., Ross, P., Corne, D.: A promising genetic algorithm approach to job shop scheduling, rescheduling, and open-shop scheduling problems. In Forrest, S. (ed.) *Fifth International Conference on Genetic Algorithms*, San Mateo pp. 375–382. Morgan Kaufmann, CA, USA (1993).
  - 25. Fang, H.L., Ross, P., Corne, D.: A promising hybrid GA/heuristic approach for open-shop scheduling problems. In: Cohn, A. (ed.) *Eleventh European Conference on Artificial Intelligence*. John Wiley & Sons, NJ, USA (1994)
  - 26. Fisher, H., Thompson, G.L.: Probabilistic learning combinations of local job-shop scheduling rules. In: Factory Scheduling Conference, Carnegie Institute of Technology, May 10–12 (1961)

27. Fisher, H., Thompson, G.L.: Probabilistic learning combinations of local job-shop scheduling rules. In: Muth, J.F., Thompson, G.L. (eds.) *Industrial Scheduling*, pp. 225–251. Prentice-Hall, New Jersey (1963)
28. Fukunaga, A.: Automated discovery of composite SAT variable selection heuristics. In: Proceedings of the National Conference on Artificial Intelligence (AAAI), pp. 641–648. Edmonton, Canada (2002)
29. Fukunaga, A.S.: Evolving local search heuristics for SAT using genetic programming. In: *Genetic and Evolutionary Computation—GECCO-2004, Part II*, Lecture Notes in Computer Science, pp. 483–494. Springer, Edmonton, Canada (2004)
30. Fukunaga, A.S.: Automated discovery of local search heuristics for satisfiability testing. *Evol. Comput.* **16**(1), 31–61 (2008)
31. Geiger, C.D., Uzsoy, R., Aytug, H.: Rapid modeling and discovery of priority dispatching rules: an autonomous learning approach. *J. Scheduling* **9**, 7–34 (2006)
32. Gratch, J., Chien, S.: Adaptive problem-solving for large-scale scheduling problems: a case study. *J. Artif. Intell. Res.*, **4**, 365–396 (1996)
33. Hart, E., Ross, P., Nelson, J.A.D.: Solving a real-world problem using an evolving heuristically driven schedule builder. *Evolu. Comput.* **6**(1), 61–80 (1998)
34. Holland, J.H., Reitman, J.S.: Cognitive systems based on adaptive algorithms. In: *Pattern-Directed Inference Systems*. Academic, New York (1978)
35. Joslin, D., Clements, D.P. “squeaky wheel” optimization. *J. Artif. Intell. Res.* **10**, 353–373 (1999)
36. Keller, R.E., Poli, R.: Cost-benefit investigation of a genetic-programming hyperheuristic. In: *Proceedings of Artificial Evolution (EA'07)*, pp. 13–24. Tours, France (2007)
37. Keller, R.E., Poli, R.: Linear genetic programming of parsimonious metaheuristics. In: *Proceedings of Congress on Evolutionary Computation (CEC 2007)*, Singapore (2007)
38. Kendall, G., Mohamad, M.: Channel assignment in cellular communication using a great deluge hyper-heuristic. In: *Proceedings of the 2004 IEEE International Conference on Network (ICON2004)*, Singapore, 16–19 November, pp. 769–773 (2004)
39. Kibria, R.H., Li, Y.: Optimizing the initialization of dynamic decision heuristics in DPLL-SAT solvers using genetic programming. In: *Proceedings of the 9th European Conference on Genetic Programming*, Lecture Notes in Computer Science, Budapest, Hungary, vol. 3905, pp. 331–340. Springer, Berlin (2006)
40. Koza, J.R.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, MA, USA (1992)
41. Marín-Blázquez, J.G., Schulenburg, S.: A hyper-heuristic framework with XCS: learning to create novel problem-solving algorithms constructed from simpler algorithmic ingredients. In *IWLCS*, Lecture Notes in Computer Science, vol. 4399 pp. 193–218. Springer, Berlin/Heidelberg (2005)
42. Mladenovic, N., Hansen, P.: Variable neighborhood search. *Computers and Operations Research*, **24**(11), 1097–1100, (1997)
43. Mockus, J., Mockus, L.: Bayesian approach to global optimization and applications to multi-objective constrained problems. *Journal of Optimization Theory and Applications*, **70**(1), 155–171, July (1991)
44. Nareyek, A.: Choosing search heuristics by non-stationary reinforcement learning. In Resende, M.G.C., de Sousa, J.P. eds. *Metaheuristics: Computer Decision-Making*, chapter 9, pp. 523–544. Kluwer, (2003)
45. Ochoa, G., Qu, R., Burke, E.K.: Analyzing the landscape of a graph based hyper-heuristic for timetabling problems. *Proceedings of the ACM Genetic and Evolutionary Computation Conference (GECCO 2009)*, pp. 341–348. Montreal, Canada. (2009)
46. Ochoa, G., Váquez-Rodríguez, J.A., Petrovic, S., Burke, E.K., Dispatching rules for production scheduling: a hyper-heuristic landscape analysis. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2009)*, pp. 1873–1880. Trondheim, Norway (2009)
47. Oltean, M.: Evolving evolutionary algorithms using linear genetic programming. *Evolu. Comput.* **13**(3), 387–410 (2005)

48. Özcan, E., Bilgin, B., Korkmaz, E.E.: Hill climbers and mutational heuristics in hyper-heuristics. Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN 2006), Reykjavik, Iceland, September. Lecture Notes in Computer Science, vol. 4193, pp. 202–211. Springer, Heidelberg (2006)
49. Özcan, E., Bilgin, B., Korkmaz, E.E.: A comprehensive analysis of hyper-heuristics. *Intell. Data Anal.* **12**(1), 3–23 (2008)
50. Pisinger, D., Ropke, S.: A general heuristic for vehicle routing problems. *Comput. Oper. Res.*, **34**, 2403–2435 (2007)
51. Qu R., Burke, E.K.: Hybridisations within a graph based hyper-heuristic framework for university timetabling problems. *J. Oper. Res. Soc.* (2008) to appear, doi: 10.1057/jors.2008.102
52. Ross, P.: Hyper-heuristics. In: Burke, E.K., Kendall, G., (eds.) *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, ch 17, pp. 529–556. Springer (2005)
53. Ross, P., Marín-Blázquez, J.G.: Constructive hyper-heuristics in class timetabling. In: *IEEE Congress on Evolutionary Computation*, Edinburgh, UK. pp. 1493–1500. IEEE, NJ, USA (2005)
54. Ross, P., Marin-Blazquez, J.G., Hart, E.: Hyper-heuristics applied to class and exam timetabling problems. In: *Proceedings of the 2004 IEEE Congress on Evolutionary Computation*, pp. 1691–1698 IEEE Press Portland, Oregon (2004)
55. Ross, P., Schulenburg, S., Marín-Blázquez, J.G., Hart, E.: Hyper-heuristics: learning to combine simple heuristics in bin-packing problem. In: *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO'02. Morgan-Kauffman, CA, USA (2002)
56. Shaw, P.: Using constraint programming and local search methods to solve vehicle routing problems. In: *Proceedings of International Conference on Principles and Practice of Constraint Programming (CP'98)*, Lecture Notes in Computer Science, vol. 1520, pp. 417–431. Springer (1998)
57. Soubeiga, E.: Development and Application of Hyperheuristics to Personnel Scheduling. Ph.D. thesis, School of Computer Science and Information Technology, University of Nottingham, June (2003)
58. Storer, R.H., Wu, S.D., Vaccari, R.: Problem and heuristic space search strategies for job shop scheduling. *ORSA J. Comput.*, **7**(4), 453–467 (1995)
59. Tay, J.C., Ho, N.B.: Evolving dispatching rules using genetic programming for solving multi-objective flexible job-shop problems. *Comput Indust Eng* **54**, 453–473 (2008)
60. Terashima-Marín, H., Flores-Álvarez, E.J., Ross, P.: Hyper-heuristics and classifier systems for solving 2D-regular cutting stock problems. In: Beyer, H.-G., O'Reilly, U.-M. (eds.) *Genetic and Evolutionary Computation Conference*, GECCO 2005, Proceedings, Washington DC, USA, June 25–29, 2005, pp. 637–643. ACM, NY, USA (2005)
61. Terashima-Marín, H., Moran-Saavedra, A., Ross, P.: Forming hyper-heuristics with GAs when solving 2D-regular cutting stock problems. In: *Proceedings of the 2005 IEEE Congress on Evolutionary Computation*, vol. 2, pp. 1104–1110, IEEE Press Edinburgh, Scotland, UK (2005)
62. Terashima-Marín, H., Ross, P., Valenzuela-Rendon, M.: Evolution of constraint satisfaction strategies in examination timetabling. In: *Genetic and Evolutionary Computation Conference*, GECCO'99, pp. 635–642 (1999)
63. Vazquez-Rodriguez, J.A., Petrovic, S., Salhi, A.: A combined meta-heuristic with hyper-heuristic approach to the scheduling of the hybrid flow shop with sequence dependent setup times and uniform machines. In: Baptiste, P., Kendall, G. Munier, A. Sourd, F. (eds.) *Proceedings of the 3rd Multidisciplinary International Scheduling Conference: Theory and Applications (MISTA 2007)* (2007)
64. Wilson, S.W.: Classifier systems based on accuracy. *Evolu. Comput.*, **3**(2), 149–175 (1995)

# Chapter 16

## Metaheuristic Hybrids

Günther R. Raidl, Jakob Puchinger and Christian Blum

**Abstract** Over the last years, so-called hybrid optimization approaches have become increasingly popular for addressing hard optimization problems. In fact, when looking at leading applications of metaheuristics for complex real-world scenarios, many if not most of them do not purely adhere to one specific classical metaheuristic model but rather combine different algorithmic techniques. Concepts from different metaheuristics are often hybridized with each other, but they are also often combined with other optimization techniques such as branch-and-bound and methods from the mathematical programming and constraint programming fields. Such combinations aim at exploiting the particular advantages of the individual components, and in fact well-designed hybrids often perform substantially better than their “pure” counterparts. Many very different ways of hybridizing metaheuristics are described in the literature, and unfortunately it is usually difficult to decide which approach(es) are most appropriate in a particular situation. This chapter gives an overview of this topic by starting with a classification of metaheuristic hybrids and then discussing several prominent design templates which are illustrated by concrete examples.

### 16.1 Introduction

Most of the other chapters of this book illustrate the existence of a large number of different metaheuristics. Simulated annealing, tabu search, evolutionary algorithms such as genetic algorithms and evolution strategies, ant colony optimization, particle

---

Günther R. Raidl

Institute of Computer Graphics and Algorithms, Vienna University of Technology, Vienna Austria  
e-mail: raidl@ads.tuwien.ac.at

Jakob Puchinger

NICTA Victoria Laboratory, University of Melbourne, Melbourne, Australia  
e-mail: jakobp@csse.unimelb.edu.au

Christian Blum

ALBCOM Research Group, Universitat Politècnica de Catalunya, Barcelona, Spain  
e-mail: cblum@lsi.upc.edu

swarm optimization, scatter search, path relinking, the greedy randomized adaptive search procedure, multi-start and iterated local search, and variable neighborhood search are—among others—prominent examples. Each of them has an individual historical background, follows certain paradigms and philosophies, and puts one or more particular strategic concepts in the foreground.

Over the last years a large number of algorithms were reported that do not purely follow the concepts of one single traditional metaheuristic, but they combine various algorithmic ideas, often originating from other branches of optimization and soft-computing. These approaches are commonly referred to as *metaheuristic hybrids* or *hybrid metaheuristics*. Note that the lack of a precise definition of these terms is sometimes subject to criticism. In our opinion, however, the relatively open nature of these terms is rather helpful, as strict borderlines between related fields of research are often a hindrance for creative thinking and the exploration of new research directions.

The motivation behind hybridizations of different algorithmic concepts is usually to obtain better performing systems that exploit and unite advantages of the individual pure strategies; i.e., such hybrids are believed to benefit from *synergy*. In fact, today it seems that choosing an adequate combination of multiple algorithmic concepts is the key for achieving top performance in solving most difficult problems. The vastly increasing number of reported applications of metaheuristic hybrids and dedicated scientific events such as the *Workshops on Hybrid Metaheuristics* [7, 12, 15, 22], the *Workshops on Matheuristics* [53, 64], and the conferences on the *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems* [74] documents the popularity, success, and importance of this specific line of research.

The idea of hybridizing metaheuristics is not new but dates back to their origins. At the beginning, however, such combinations were not very popular since several relatively strongly separated and sometimes competing communities of researchers existed who tended to consider “their” favorite class of metaheuristics generally superior to others and dogmatically followed their specific philosophies. For example the evolutionary computation community grew up quite isolated and followed quite strictly the biologically inspired thinking. The situation changed, according to many researchers, with the no free lunch theorems [104] when people recognized that there cannot exist a general optimization strategy which is always better than any other. In fact, solving a specific problem most effectively almost always requires a particularly tuned algorithm that is compiled of an adequate combination of sometimes very problem-specific parts often originating from different metaheuristics and other algorithmic techniques. Exploiting problem-specific knowledge in the best possible ways, picking the right algorithmic components, and combining them in the most appropriate way are key ingredients for leading optimization algorithms.

Unfortunately, developing a highly effective hybrid approach is in general a difficult task and sometimes even considered an art. Nevertheless, there are several strategies that have proven successful on many occasions, and they can provide some guidance. In the next section, we will start with a general classification of metaheuristic hybrids. The following sections will discuss the most prominent

algorithmic templates of combinations and illustrate them with selected examples from the literature. For a more comprehensive review on hybrid metaheuristics, we recommend the book by Blum et al. [20].

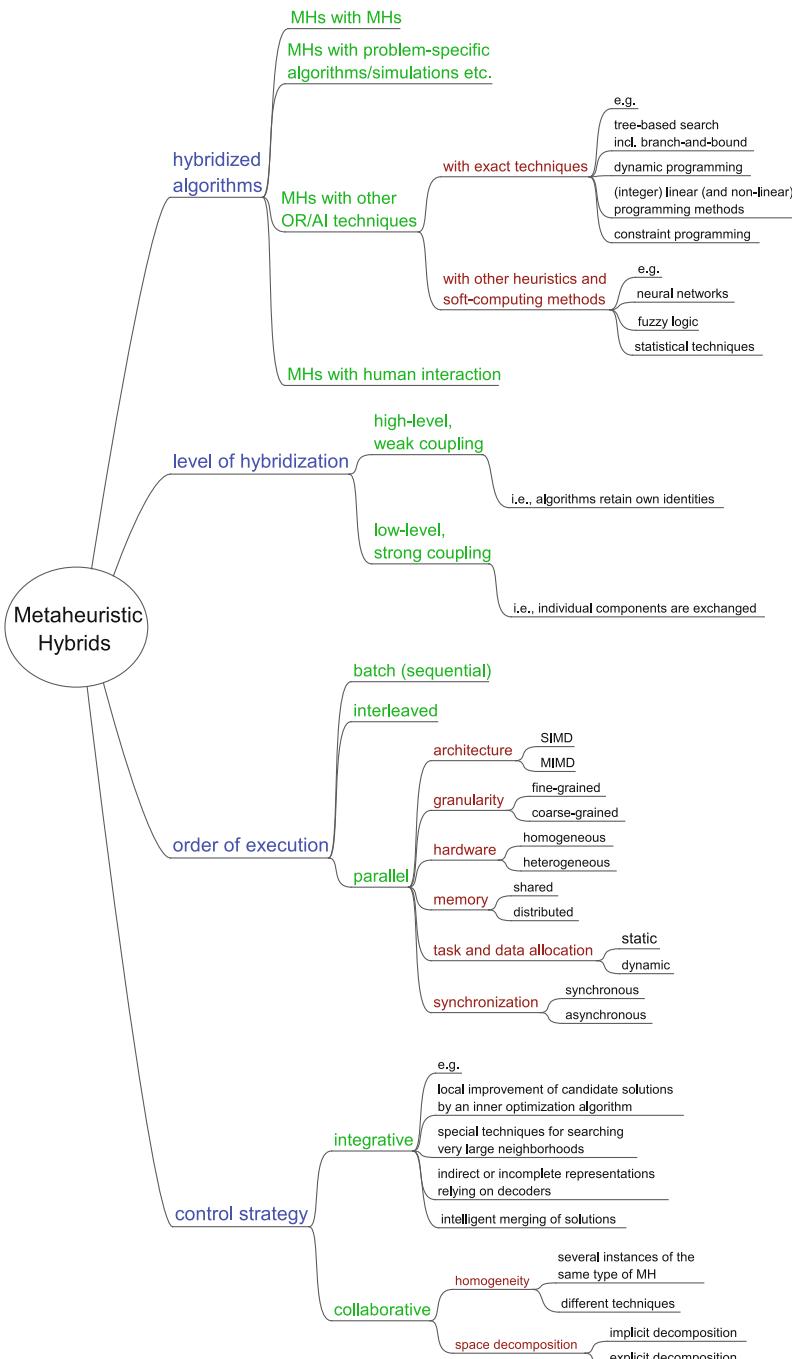
## 16.2 Classification

Several classifications and taxonomies of hybrid metaheuristics can be found in the literature. Here we primarily follow the classification from Raidl [87] that combines aspects of the taxonomy introduced by Talbi [94] with the points of view from Cotta [29] and Blum and Roli [21]. Differentiations with regard to parallel metaheuristics and hybridization of metaheuristics with exact optimization techniques are adopted from El-Abd and Kamel [37] and from Puchinger and Raidl [81], respectively. Figure 16.1 illustrates our classification.

We primarily distinguish hybrid metaheuristics according to four criteria, namely the kinds of algorithms that are hybridized, the level of hybridization, the order of execution, and the control strategy.

**Hybridized algorithms.** First, one might combine (parts of) different metaheuristic (MH) strategies, which is probably the most common approach. Second, highly problem-specific algorithms, such as entire simulations for acquiring the quality of candidate solutions, are sometimes used in conjunction with metaheuristics. As a third class we consider the combination of metaheuristics with other more general techniques coming from fields like operations research (OR) and artificial intelligence (AI). Here, we can further distinguish between combinations with exact techniques or with other heuristics and soft-computing methods. Prominent examples for exact techniques that are often very successfully combined with metaheuristics are tree-search-based methods such as branch-and-bound (B&B), dynamic programming, linear programming (LP) and mixed integer programming (MIP) methods as well as nonlinear programming techniques, and constraint programming (CP). For a survey dedicated to combinations of metaheuristics with MIP techniques see [88], for an overview on combinations of local search-based methods with CP see [45], and for a review on combinations of local search methods with exact techniques see [36]. Examples of other heuristic and soft-computing techniques include neural networks, fuzzy logic, and several statistical techniques. As a fourth class we want to mention the combination of metaheuristics with a human interaction component called *human guided search*. In particular for problems where it is difficult to quantify the quality of solutions in mathematically precise ways, where candidate solutions can be well visualized, and where human intuition and intelligence present a particular advantage, such interactive systems are often highly effective in practice [60].

**Level of hybridization.** Hybrid metaheuristics can further be differentiated according to the *level* (or strength) at which the individual algorithms are coupled:



**Fig. 16.1** Classification of metaheuristic (MH) hybrids based on Raidl [87].

*High-level* combinations retain in principle the individual identities of the original algorithms and cooperate over a relatively well-defined interface; there is no direct, strong relationship of the internal workings of the algorithms. On the contrary, algorithms in *low-level* combinations strongly depend on each other; individual components or functions of the algorithms are exchanged.

**Order of execution.** In the simplest case, the *batch* execution, the individual algorithms are performed in a sequential way, and results of the first algorithm are used as input for the second. More sophisticated approaches apply the individual algorithms in an *intertwined* or even *parallel* way, and information is exchanged more frequently, usually in a bidirectional way. Parallel metaheuristics are an important research area by themselves and independent classifications of hybrid parallel approaches have been proposed in [6, 37]. They distinguish the following major criteria: (a) the architecture (SIMD: single instruction, multiple data streams versus MIMD: multiple instructions, multiple data streams), (b) the granularity of parallelization (fine- or coarse-grained), (c) the hardware (homogeneous or heterogeneous), (d) the memory (shared or distributed), (e) task and data allocation (static or dynamic), and (f) whether the parallel processes run asynchronously or are synchronized in some way.

**Control strategy.** Last but not least, we distinguish metaheuristic hybrids according to their control strategy, which can be either *integrative* (coercive) or *collaborative* (cooperative).

In the extremely popular integrative case, one algorithm is the subordinate, embedded component of another. Examples include the local improvement of candidate solutions by an inner optimization algorithm (as in memetic algorithms, see also Section 16.3), special techniques for searching large neighborhoods (see Section 16.9.1), indirect or incomplete representations relying on decoders (see Section 16.5), and intelligent merging (recombination) of solutions (see Section 16.6).

In contrast, in collaborative approaches the individual algorithms exchange information but are not part of each other. For example, the popular island model [26] for parallelizing evolutionary algorithms is of this type. Collaborative approaches can further be classified into homogeneous approaches, where several instances of one and the same algorithm are performed (as in traditional island models), and heterogeneous approaches. An example for the latter are *asynchronous teams* (A-Teams) [95]: An A-Team consists of a collection of agents and memories connected into a strongly cyclic directed network. Each of these agents is an optimization algorithm that works asynchronously on the target problem, on a relaxation of it, i.e., a superproblem, or on a subproblem. Information is exchanged via shared memory. Denzinger and Offermann [32] presented a similar multi-agent approach for achieving cooperation between search algorithms following different search paradigms, such as B&B and evolutionary algorithms. In especially collaborative combinations, another question is which search spaces are actually explored by the individual algorithms. Implicit decomposition results from different initial solutions, parameter settings, or random decisions,

while an explicit decomposition is obtained when each algorithm works on its individually defined subspace. Effectively decomposing large problems is often an important issue in practice. Occasionally, problems decompose in a relatively natural way, see Sections 16.4 and 16.9, but most often finding a strong decomposition into weakly related or even unrelated subproblems is a difficult task, and (self-)adaptive schemes are sometimes applied.

Starting with the next section, we will consider several templates of implementing metaheuristic hybrids, which have successfully been applied on many occasions.

## 16.3 Finding Initial or Improved Solutions by Embedded Methods

The most natural way of hybridizing two optimization algorithms is probably to embed one algorithm into another either for obtaining promising starting solutions or for possibly improving intermediate solutions.

Problem-specific construction heuristics are often used for finding initial solutions which are then further improved by local search or metaheuristics. A frequently applied and more general strategy for obtaining initial solutions is to solve a relaxation of the original problem (e.g., the LP relaxation) and eventually repair the obtained solution in some heuristic way. Examples of such approaches can also be found in Section 16.7.

The *greedy randomized adaptive search procedure* (GRASP) [40] systematically extends the principle of locally improving a starting solution by iterating a randomized construction process, and each of the resulting solutions is then used as a starting point for local search.

The so-called *proximate optimality principle* (POP) was first mentioned by Glover and Laguna in the context of tabu search [48]. It refers to the general intuition that good solutions are likely to have a similar structure and can therefore be found close to each other in the search space. Fleurent and Glover transferred this principle in [44] from complete to partial solutions in the context of GRASP. They suggested that mistakes introduced during the construction process may be undone by applying local search during (and not only at the end of) the GRASP construction phase. They proposed a practical implementation of POP in GRASP by applying local search at a few stages of the construction phase only. Another application of this concept can be found in [14] for the job-shop scheduling problem.

Often local search procedures or more sophisticated improvement algorithms are applied within an outer metaheuristic for “fine-tuning” intermediate candidate solutions. While the outer metaheuristic is responsible for diversification, the inner improvement algorithm focuses on intensification. For example, most *memetic algorithms* [69] rely on this principle: The outer metaheuristic is an evolutionary algorithm, and intermediate candidate solutions are locally improved. If each intermediate solution is always turned into a local optimum, the evolutionary algorithm

exclusively searches the space of local optima (w.r.t. the neighborhood structure of the inner local improvement procedure) only. Memetic algorithms are often more successful than simple evolutionary algorithms, because intensification is typically considered a weakness of traditional evolutionary algorithms. By adjusting how much effort is spent in the local improvement, one can tune the balance between intensification and diversification. Note that the inner local improvement does not always have to be just a simple local search. Occasionally, more sophisticated strategies like tabu search or even exact techniques for solving a restricted problem are applied. This also leads to the related *large neighborhood search* methods, which we will consider in Section 16.9.1.

Another example is *variable neighborhood search* (VNS) [55], where each candidate solution undergoes some kind of local improvement and a hierarchy of different neighborhood structures is utilized. In especially *general variable neighborhood search*, a more sophisticated *variable neighborhood descent* is used as the inner local improvement procedure.

Considering exact techniques, B&B approaches strongly rely on good upper and lower bounds in order to prune the search tree as strongly as possible. Metaheuristic techniques are frequently applied to obtain a promising initial solution or to improve intermediate solutions in order to find tight(er) bounds. Sections 16.6 and 16.8 contain several examples such as [31, 91] that also fall into this category.

## 16.4 Multi-stage Approaches

Some optimization approaches consist of multiple sequentially performed stages, and different techniques are applied at the individual stages.

In many complex real-world applications, the problem naturally decomposes into multiple levels. If the decision variables associated with the lower level(s) have a significantly weaker impact on the objective value than the higher-level variables or if the impact of these variable sets is only loosely correlated, it is a very reasonable approach to optimize the individual levels in a strictly sequential manner. Different techniques can be applied at the individual levels yielding simple but often very effective hybrid approaches.

For example, for certain vehicle routing applications where the aim is to deliver goods to customers, it may be a meaningful approach to first partition the customers into groups which are then independently treated by finding appropriate delivery tours; finally, the drivers' specific time-plans are derived from these tours. For certain job scheduling problems, it might be feasible to first assign the jobs to machines and then independently optimize the schedules for each machine. For large communication network design problems it might be wise to first optimize a possibly redundant backbone infrastructure, then design the individual local access network structures, and finally decide about the concrete cable laying and technical parameters such as the capacities of the individual links.

We remark that in practice such multi-stage approaches will usually not lead to optimal solutions, as the sequentially solved subproblems are typically not

independent. However, for many complicated real-world problems of large size, as for example when designing a communication infrastructure for a larger city, a multi-stage approach is the only viable choice. Furthermore, multi-stage approaches are often very meaningful for relatively quickly finding first approximate solutions. Therefore, they are frequently used in practice.

Multi-stage approaches are sometimes even applied when such a problem decomposition is not so obvious but results in algorithmic advantages. Classical pre-processing techniques, where the problem is usually reduced to a hard-to-solve core by applying certain problem-specific simplification strategies, are an example.

A more general, systematic approach is based on tools from the field of parameterized complexity. It offers both a framework of complexity analysis and toolkits for algorithm design. One of the tools for algorithm design is known as *problem kernelization*. The idea is to reduce a given problem instance in polynomial time to a so-called problem kernel such that an optimal solution to the problem kernel can, in polynomial time, be transformed into an optimal solution to the original problem instance. In [47], Gilmour and Dras propose several different ways of using the information given by the kernel of a problem instance for making ant colony system more efficient for solving the minimum vertex cover problem. The most intuitive version applies the ant colony system directly to the problem kernel and subsequently transforms the best solution obtained for the kernel into a solution to the original problem instance.

*Multi-level refinement strategies* [101] can also be considered as a special class of multi-stage approaches. They involve a recursive coarsening to create a hierarchy of approximations to the original problem. An initial solution is identified for the coarsest level and is then iteratively refined at each level—coarsest to finest—typically by using some kind of (meta-)heuristic at each level. *Solution extension* operators transfer the solution from one level to the next. In *iterated multi-level algorithms*, solutions are not just refined but occasionally also re-coarsened, and the refinement process is iterated. These strategies have been successfully applied on several problems including multi-level graph partitioning, graph coloring, very large traveling salesman problems, vehicle routing, and DNA sequencing.

*Variable fixing* strategies where variables of the original problem are fixed to certain values (according to some, usually heuristic, criterion) to perform the optimization over a restricted search space are also related to the above-mentioned strategies. Examples of effective variable fixing strategies are the *core concepts* for knapsack problems [77, 85].

Some approaches determine a set of (complete) initial solutions by a first-stage method and apply one (or even more) other technique(s) for further improving upon them. For example, occasionally a metaheuristic is used for finding a pool of diverse high-quality solutions, and *merging* is performed to identify a single final solution combining the particularly beneficial properties of the intermediate solutions. We will consider merging in more detail in Section 16.6.

Tamura et al. [96] tackle a job-shop scheduling problem and start from its ILP formulation. For each variable, they take the range of possible values and partition it into a set of subranges, which are then indexed. The encoded solutions of a genetic

algorithm (GA) are defined so that each position represents a variable, and its value corresponds to the index of one of the subranges. The fitness of such a chromosome is calculated using Lagrangian relaxation in order to obtain a bound on the optimal solution subject to constraints on the variable values which must fall within the represented ranges. When the GA terminates, an exhaustive search of the region identified as the most promising is carried out in a second stage.

A special kind of a sequential combination of B&B and a GA is described by Nagar et al. [70] for a two-machine flow-shop scheduling problem. Candidate solutions are represented as permutations of jobs. In the first stage, B&B is executed down to a predetermined branching depth  $k$  and suitable bounds are calculated and recorded at each node of the explicitly stored B&B tree. During the execution of the GA in the second stage, each partial solution up to position  $k$  is mapped onto the corresponding tree node. If the associated bounds indicate that no path below this node can lead to an optimal solution, the permutation undergoes a mutation that has been specifically designed to change the early part in a favorable way.

In [99], Vasquez and Hao present a two-stage approach for tackling the 0–1 multi-dimensional knapsack problem (MKP). Given  $n$  items and  $m$  resources, each object has an associated profit  $c_i$  and resource consumptions  $a_{i,j}$ ,  $\forall i = 1, \dots, n$ ,  $\forall j = 1, \dots, m$ , and each resource has a capacity  $b_j$ . The goal of the MKP is to choose a subset of the  $n$  objects such that its total profit is maximized without violating the capacity constraints. In the ILP formulation of the MKP, a binary variable  $x_i \in \{0, 1\}$  is defined for each object. In the first stage of the proposed hybrid solution method a series of LP relaxations with additional constraints is solved. They are of the form  $\sum_{i=1}^n x_i = k$  where  $k \in \{k_{\min}, \dots, k_{\max}\}$ , i.e., the number of items to be selected is fixed to  $k$ . Each setting of  $k$  defines an LP that is solved to optimality. In the second stage of the process, tabu search is used to search for better solutions around the usually non-integral optimal solutions of the  $k_{\max} - k_{\min} + 1$  LPs. The approach has been improved in [100] by additional variable fixing.

A combination of tabu search and CP for the job-shop scheduling problem is presented by Watson and Beck [102]. First an iterated simple tabu search [103] is run for a limited time. The best resulting solutions are used for initializing a *solution-guided multi-point constructive search* (SGMPCS) [13]. SGMPCS is a recent algorithm combining CP tree search with the population concept of metaheuristics. In the CP tree search, a single variable is chosen (variable-ordering heuristic) and assigned a value (value-ordering heuristic). The domains of the remaining variables are reduced accordingly and another variable is chosen, repeating the process as long as unassigned variables exist or until the partial solution becomes infeasible, in which case backtracking to the last choice-point is performed. The main idea of SGMPCS is to perform a standard tree search that is restarted if the number of failures exceeds a certain limit. The value-ordering heuristic is partly determined by an incumbent solution: If the value of the chosen variable in this solution is allowed by its current domain, the variable is assigned that value; else another value is chosen using another kind of value-ordering heuristic. SGMPCS maintains a small set of solutions which is updated in an elitist way. The tree search is restarted with a randomly chosen solution from the set until a stopping criterion is met.

## 16.5 Decoder-Based Approaches

The hybrid metaheuristic design template considered in this section is particularly popular for problems where solutions must fulfill certain constraints and a fast construction heuristic yielding feasible solutions exists. In *decoder-based* approaches, a candidate solution is represented in an indirect or incomplete way and a problem-specific decoding algorithm is applied for transforming the encoded solution into a complete feasible solution. This principle is often applied in evolutionary algorithms, where encoded solutions are denoted as *genotypes* and the decoded counterparts are called *phenotypes* [51].

A prominent, quite generic way of indirectly representing solutions is by means of *permutations* of solution attributes. The decoder is then usually a greedy construction heuristic which composes a solution by trying to add the solution attributes in the order given by the permutation, i.e., the order of an attribute in the permutation is the greedy criterion. Cutting and packing problems are examples where such decoder-based methods are frequently used [59]. The overall performance obviously depends strongly on the quality and the speed of the decoder. Such approaches are often straightforward and relatively easy to implement, in particular as standard metaheuristics with traditional neighborhoods for permutations can directly be applied. On the downside, more elaborate metaheuristics based on direct encodings and tuned problem-specific operators are often likely to achieve better performance, as they might exploit problem-specific features in better ways.

Particularly attractive are decoder-based approaches where the decoder is a more sophisticated algorithm rather than a simple construction procedure. For example, a mixed integer linear programming problem (MIP) can be approached by splitting the variables into the integral and continuous parts. One can then apply a metaheuristic to optimize the integer part only; for each candidate solution, corresponding optimal fractional variable values are efficiently determined by solving the remaining LP. Such approaches are described in conjunction with GRASP by Neto and Pedroso [71] and in conjunction with tabu search by Pedroso [73].

Glover [49] proposed an alternative *parametric tabu search* for heuristically solving MIPs. A current search point is indirectly represented by the solution to the LP relaxation of the MIP plus additional *goal conditions* that restrict the domains of a subset of the integer variables. Instead of considering the goal conditions directly as hard constraints when applying an LP solver, they are relaxed and brought into the objective function similarly as in Lagrangian relaxation. In this way, the approach can also be applied to problems where it is hard to find any feasible integer solution. The approach further uses a variety of intensification and diversification strategies based on adaptive tabu memory in order to make the heuristic search more efficient.

Besides problem-specific heuristics and LP solvers, other efficient techniques are sometimes used as a decoder to augment incompletely represented solutions. For example, Hu and Raidl [58] consider the generalized traveling salesman problem in which a clustered graph is given and a shortest tour visiting exactly one node from each cluster is requested. Their approach is based on VNS and represents a candidate solution in two different ways: On the one hand, a permutation of clusters

is given, representing the order in which the clusters are to be visited. A dynamic programming procedure is used as decoder to derive a corresponding optimal selection of particular nodes. On the other hand, only the unordered set of selected nodes from each cluster is given, and the classical chained Lin–Kernighan heuristic for the traveling salesman problem [66] is used as a decoder to obtain a corresponding high-quality tour. The VNS uses several types of neighborhood structures defined for each representations.

Last but not least, decoder-based approaches have recently been used in ant colony optimization (ACO). For example, Blum and Blesa [19] present a decoder-based ACO for the general  $k$ -cardinality tree problem. Given an undirected graph, this problem involves finding among all trees with exactly  $k$  edges a tree such that a certain objective function is minimized. In contrast to a standard ACO algorithm that constructs trees (i.e., solutions) with exactly  $k$  edges, the decoder-based approach of [19] builds  $l$ -cardinality trees, where  $l > k$ . Subsequently, an efficient dynamic programming algorithm is applied for finding the best  $k$ -cardinality tree that is contained in the  $l$ -cardinality tree. Results show that this approach has clear advantages over standard ACO approaches.

## 16.6 Solution Merging

The basic idea of *solution merging* is to derive a new, hopefully better solution from the attributes appearing in two or more promising input solutions. The observation that high-quality solutions usually have many attributes in common is exploited.

In the simplest form, this operation corresponds to the classical recombination (crossover) which is considered the primary operator in GAs: Usually two parent solutions are selected and an offspring is constructed by inheriting attributes from both of them based on random decisions. While such an operation is computationally cheap, created offspring are often worse than the respective parents, and many repetitions are usually necessary for achieving strong improvements.

Alternatively, one can put more effort into the determination of such offspring. An established technique is *path relinking* [50]. It traces a path in the search space from one parent to a second by always exchanging only a single attribute (or more generally by performing a move in a simple neighborhood structure toward the target parent). An overall best solution found on this path is finally taken as offspring.

This concept can further be extended by considering not just solutions on an individual path between two parents, but the whole subspace of solutions defined by the joined attributes appearing in a set of two or more input solutions. An *optimal merging* operation returns a best solution from this subspace, i.e., it identifies a best possible combination of the ancestors' features that can be attained without introducing new attributes. Depending on the underlying problem, identifying such an optimal offspring is often a hard optimization problem on its own, but due to the limited number of different properties appearing in the parents, it can often be solved in reasonable time in practice.

Applegate et al. [8, 9] were among the first to apply more sophisticated merging in practice. For the traveling salesman problem, they derive a set of different tours by a series of runs of the chained Lin–Kernighan iterated local search algorithm. The sets of edges of all these solutions are merged and the traveling salesman problem is finally solved to optimality on this strongly restricted graph. Solutions are achieved that are typically superior to the best ones obtained by the iterated local search.

Besides the one-time application of merging to a set of heuristically determined solutions in a multi-stage way, sophisticated merging can also replace classical recombination in evolutionary and memetic algorithms. Aggarwal et al. [1] originally suggested such an approach for the independent set problem. The subproblem of identifying the largest independent set in the union of two parental independent sets is solved exactly by an efficient algorithm. Ahuja et al. [4] apply this concept to a GA for the quadratic assignment problem. As the optimal recombination problem is more difficult in this case, they use a matching-based heuristic that quickly finds high-quality offspring solutions. Optimal merging is also used by Blum [17] in the context of an evolutionary algorithm for the  $k$ -cardinality tree problem. The individuals are trees with  $k$  edges. Crossover first combines two parent trees, producing hereby a larger  $l$ -cardinality tree. Dynamic programming is then used to reduce this tree to the best feasible subtree with  $k$  edges.

Eremeev [38] studies the computational complexity of producing a best possible offspring from two parents for binary representations from a theoretical point of view. He concludes that the optimal recombination problem is polynomially solvable for the maximum weight set packing problem, the minimum weight set partition problem, and linear Boolean programming problems with at most two variables per inequality. On the other hand, determining an optimal offspring is NP-hard for 0/1 integer programming with three or more variables per inequality, like the knapsack, set covering, and  $p$ -median problems, among others.

Cotta and Troya [30] discuss merging in the light of a more general framework for hybridizing B&B and evolutionary algorithms. They show the usefulness of applying B&B for identifying optimal offspring on various benchmarks.

For mixed integer programming, Rothberg [91] suggests a tight integration of an evolutionary algorithm in a branch-and-cut based MIP solver. At regular intervals the evolutionary algorithm is applied as a B&B tree node heuristic. Optimal recombination is performed by first fixing all variables that are common in the selected parental solutions and by applying the MIP solver to this reduced subproblem. Mutation selects one parent, fixes a randomly chosen subset of variables, and calls the MIP solver for determining optimal values for the remaining problem. Since the number of variables to be fixed is a critical parameter, an adaptive scheme is applied to control it. Experimental results indicate that this hybrid is able to find significantly better solutions than other heuristic methods for several very difficult MIPs. This method is integrated in the commercial MIP solver CPLEX<sup>1</sup> since version 10.

---

<sup>1</sup> <http://www.ilog.com>

## 16.7 Strategic Guidance of Metaheuristics by Other Techniques

Many successful hybrid metaheuristics use other optimization techniques for guiding the search process. This may be done by either using information gathered by applying other algorithms such as optimal solutions to problem relaxations or directly enhancing the functionality of a metaheuristic with algorithmic components originating from other techniques. In the following two sections we give examples for both variants.

### 16.7.1 Using Information Gathered by Other Algorithms

Guiding metaheuristics using information gathered by applying other algorithms is often a very successful approach that is commonly used. *Problem relaxations*, where some or all constraints of a problem are loosened or omitted, are often used to efficiently obtain bounds and approximate (not necessarily feasible) solutions to the original problem. The gathered information can be utilized for guiding the search, since an optimal solution to a relaxation often indicates in which parts of the original problem's search space good or even optimal solutions might be found.

Sometimes an optimal solution to a relaxation can be repaired by a problem-specific procedure in order to make it feasible for the original problem and to use it as a promising starting point for a subsequent metaheuristic (or exact) search; see also Section 16.3. For example, Raidl [86] applies this idea in a GA for the MKP. The MKP's LP relaxation is solved and a randomized rounding procedure derives an initial population of diverse solutions from the LP-optimum. Furthermore, the LP-optimum is also exploited for guiding the repair of infeasible candidate solutions and for local improvement. The variables are sorted according to increasing LP values. The greedy repair procedure considers the variables in this order and removes items from the knapsack until all constraints are fulfilled. In the greedy improvement procedure, items are considered in reverse order and included in the knapsack as long as no constraint is violated. Many similar examples for exploiting LP solutions—also including the biasing of variation operators such as recombination and mutation in evolutionary algorithms—exist.

Plateau et al. [78] combine interior point methods and metaheuristics for solving the MKP. In a first step an interior point method is performed with early termination. By rounding and applying several different ascent heuristics, a population of different feasible candidate solutions is generated. This set of solutions is then the initial population for a path relinking/scatter search.

Puchinger and Raidl [83] suggest a new variant of VNS: *relaxation guided variable neighborhood search*. It is based on the general VNS scheme and a new embedded variable neighborhood descent (VND) strategy utilizing different types of neighborhood structures. For a current incumbent solution, the order in which the neighborhoods are searched is determined dynamically by first solving relaxations of them. The objective values of these relaxations are used as indicators for the potential gains of searching the corresponding neighborhoods, and more promising

neighborhoods are searched first. The proposed approach has been tested on the MKP but is more generally applicable. Computational experiments involving several types of ILP-based neighborhoods show that the adaptive neighborhood ordering is beneficial for the heuristic search, improving obtained results.

Occasionally, dual variable information of LP solutions is also exploited. Chu and Beasley [25] make use of it in their GA for the MKP by calculating so-called *pseudo-utility ratios* for the primal variables and using them in similar ways as described above for the primal solution values. For the MKP, these pseudo-utility ratios tend to be better indicators for the likeliness of the corresponding items to be included in an optimal integer solution than the primal variable values and several other measures, see [85].

Other relaxations besides the LP relaxation are occasionally also exploited in conjunction with metaheuristics. A successful example is the hybrid Lagrangian GA for the prize collecting Steiner tree problem from Haouari and Siala [56]. It is based on a Lagrangian decomposition of a minimum spanning tree-like ILP formulation of the problem. The volume algorithm, which is a special variant of subgradient search [11], is used for solving the Lagrangian dual. After its termination, the GA is started and exploits results obtained from the volume algorithm in several ways: (a) The volume algorithm creates a sequence of intermediate spanning trees as a by-product. All edges appearing in these intermediate trees are marked, and only this reduced edge set is further considered by the GA; i.e., a core of edges is derived from the intermediate primal results when solving the Lagrangian dual. (b) A subset of diverse initial solutions is created by a Lagrangian heuristic, which greedily generates solutions based on the reduced costs appearing as intermediate results in the volume algorithm. (c) Instead of the original objective function, an alternate one, based on the reduced costs that are obtained by the volume algorithm, is used. The idea is to focus the search even stronger on promising regions of the search space, where also better solutions with respect to the original objective function can presumably be found.

Pirkwieser et al. [76] describe a similar combination of Lagrangian decomposition and a GA for the knapsack constrained maximum spanning tree problem. The problem is decomposed into a minimum spanning tree and a 0–1 knapsack problem. Again, the volume algorithm is employed to solve the Lagrangian dual. While graph reduction takes place as before, the objective function remains unchanged. Instead, final reduced costs are exploited for biasing the initialization, recombination, and mutation operators. In addition, the best feasible solution obtained from the volume algorithm is used as a seed in the GA’s initial population. Results indicate that the volume algorithm alone is already able to find solutions of extremely high quality even for large instances. These solutions are polished by the GA, and in most cases proven optimal solutions are finally obtained.

Dowsland et al. [34] propose an approach where bounding information available from partial solutions is used to guide an evolutionary algorithm. An indirect, order-based representation of candidate solutions is applied. Phenotypes are derived by a specific decoding procedure which is a construction heuristic that is also able to calculate upper bounds for intermediate partial solutions (considering a maximization

problem). Given a certain target value, which is, e.g., the objective value of the so far best solution, a *bound point* is determined for each candidate solution in the population: It is the first position in the genotype for which the corresponding partial solution has a bound that is worse than the target value. A modified one-point crossover is then guided by this bound information: The crossover point must be chosen in the part of the first chromosome before its bound point. In this way, recombinations definitely leading to worse offspring are avoided. The authors successfully tested this concept on a relatively simple pallet loading problem and a more complex two-dimensional packing problem with non-identical pieces.

### 16.7.2 Enhancing the Functionality of Metaheuristics

One of the basic ingredients of an optimization technique is a mechanism for exploring the search space. An important class of algorithms tackles an optimization problem by exploring the search space along a so-called *search tree*. This class of algorithms comprises approximate as well as complete techniques. An example of a complete method belonging to this class is B&B. An interesting heuristic derivative of breadth-first B&B is *beam search* [72]. While B&B (implicitly) considers all nodes at a certain level in the search tree, beam search restricts the search to a certain number of nodes based on bounding information.

One relatively recent line of research deals with the incorporation of algorithmic components originating from deterministic B&B derivatives such as beam search into construction-based metaheuristics. Examples are the so-called *Beam-ACO* algorithms [16, 18] and approximate and nondeterministic tree search (ANTS) procedures [62, 63]. Note that Beam-ACO can be seen as a generalization of ANTS. In Beam-ACO, artificial ants perform a probabilistic beam search in which the extension of partial solutions is done in the ACO fashion rather than deterministically. The existence of an accurate—and computationally inexpensive—lower bound for the guidance of the ACO’s search process is crucial for the success of Beam-ACO.

Another successful example concerns the use of CP techniques for restricting the search performed by an ACO algorithm to promising regions of the search space. The motivation for this type of hybridization is as follows: Generally, ACO algorithms are competitive with other optimization techniques when applied to problems that are not overly constrained. However, when highly constrained problems such as scheduling or timetabling are considered, the performance of ACO algorithms generally degrades. Note that this is usually also the case for other metaheuristics. The reason is to be found in the structure of the search space: When a problem is not overly constrained, it is usually not difficult to find feasible solutions. The difficulty rather lies in the optimization part, namely the search for good feasible solutions. On the other side, when a problem is highly constrained the difficulty is rather in finding any feasible solution. This is where CP comes into play, because these problems are the target problems for CP applications. Meyer and Ernst [67] introduced the incorporation of CP into ACO in an application to the single machine job scheduling problem.

## 16.8 Strategic Guidance of Other Techniques by Metaheuristics

Many metaheuristics are based on the principle of local search, i.e., starting from an initial solution, a certain neighborhood around it is investigated, and if a better solution can be identified, it becomes the new incumbent solution; this process is repeated. Thus, the central idea is to focus the search for better solutions on regions of the search space nearby already identified good solutions.

In comparison, most B&B algorithms choose the next B&B tree node to be processed by a *best-first* strategy: assuming minimization, a node with smallest lower bound is always selected, since it is considered to be most promising for leading to an optimal solution. This approach is often the best strategy for minimizing the total number of nodes that need to be explored until finding an optimum and proving its optimality. However, good complete solutions—and thus also tight upper bounds—are often found late during this search. The best-first node selection strategy typically “hops around” in the search tree and in the search space and does not stay focused on subregions. When no strong primal heuristic is applied for determining promising complete solutions, the best-first strategy is often combined with an initial *diving*, in which a depth-first strategy is followed at the beginning until some feasible solution is obtained. In depth-first search, the next node to be processed is always the one that has been most recently created by branching.

In the last years, several more sophisticated concepts have been proposed with the aim to intensify B&B search in an initial phase to neighborhoods of promising incumbents in order to quickly identify high-quality approximate solutions. In some sense, we can consider these strategies to “virtually” execute a metaheuristic.

Danna et al. [31] describe *guided dives*, which are a minor, but effective modification of the already mentioned simple diving by temporarily switching to depth-first search. The branch to be processed next in case of guided dives is always the one in which the branching variable is allowed to take the value it has in an incumbent solution. Diving is therefore biased toward the neighborhood of this solution. Instead of performing only a single dive at the beginning, guided dives are repeatedly applied at regular intervals during the whole optimization process. This strategy is trivial to implement, and experimental results indicate significant advantages over standard node selection strategies.

Fischetti and Lodi [42] propose *local branching*, an exact approach introducing the spirit of classical  $k$ -OPT local search in a generic branch-and-cut based MIP solver. The whole problem is partitioned into a  $k$ -OPT neighborhood of an initial solution and the remaining part of the search space by applying a *local branching constraint* and its inverse, respectively. The MIP solver is then forced to completely solve the  $k$ -OPT neighborhood before considering the remainder of the problem. If an improved solution has been found in the  $k$ -OPT neighborhood, a new subproblem corresponding to the  $k$ -OPT neighborhood of this new incumbent is split off and solved in the same way; otherwise, a larger  $k$  may be tried. The process is repeated until no further improvement can be achieved. Finally, the remaining problem corresponding to all parts of the search space not yet considered is processed in a standard way.

Hansen et al. [55] present a variant of local branching in which they follow the classical VNS strategy, especially for adapting the neighborhood parameter  $k$ . Improved results are reported. Another variant of the original local branching scheme is described by Fischetti et al. in [43]. They consider problems in which the set of variables can be naturally partitioned into two levels and fixing the values of the first-level variables yields substantially easier subproblems; cf. Section 16.4.

Danna et al. [31] further suggest an approach called *relaxation-induced neighborhood search* (RINS) for exploring the neighborhoods of promising MIP solutions more intensively. The main idea is to occasionally devise a sub-MIP at a node of the B&B tree that corresponds to a special neighborhood of an incumbent solution: First, variables having the same values in the incumbent and in the current solution of the LP relaxation are fixed. Second, an objective cutoff based on the objective value of the incumbent is set. Third, a sub-MIP is solved with the remaining variables. The time for solving this sub-MIP is limited. If a better incumbent could be found during this process, it is passed to the global MIP search which is resumed after the sub-MIP's termination. In the authors' experiments, CPLEX is the MIP solver, and RINS is compared to standard CPLEX, local branching, combinations of RINS and local branching, and guided dives. Results indicate that RINS often performs best. CPLEX includes RINS as a standard strategy for quickly obtaining good heuristic solutions since version 10.

The *nested partitioning* method proposed by Shi and Ólafsson [92] is another example where a metaheuristic provides strategic guidance to another technique. At each iteration the search focuses on a part of the search space called the most promising region. The remaining part of the search space is called the surrounding region. The most promising region may, for example, be characterized by a number of fixed variables. At each step, the most promising region is divided into a fixed number of subregions. This may be done, for example, by choosing one of the free variables and creating a subregion for each of the variable's possible domain value. Each of the subregions as well as the surrounding region is then sampled. The best objective function value obtained for each region is called the promising index. The region with the best index becomes the most promising region of the next iteration. The next most promising region is thus nested within the last one. When the surrounding region is found to be the best, the method backtracks to a larger region. The approach may be divided into four main steps: partitioning, sampling, selecting a promising region, and backtracking. Each of these steps may be implemented in a generic fashion, but can also be defined in a problem-specific way. In particular the sampling phase may benefit from the use of metaheuristics instead of performing a naive random sampling. In a sense, metaheuristics can be seen as enhancements for guiding the search process of the method. In [5], for example, ant colony optimization is applied for sampling, whereas in [93] local search is used for this purpose.

A very different paradigm is followed in *constraint-based local search* [98]. It combines the flexibility of CP concepts such as rich modeling, global constraints, and search abstractions with the efficiency of local search. The *Comet* programming language allows the modeling of combinatorial optimization problems in a relatively natural way. It also provides the necessary abstractions for specifying metaheuristics

and nondeterministic, local, and hybrid search. The concept of differential objects is an important aspect of Comet: Constraints maintain their violation and objective functions their evaluation. They can both return information on the possible consequences of local search moves by automatically propagating changes using so-called invariants. The separation of the model and search allows the specification of generic search procedures such as tabu search, variable neighborhood search, and hybrid evolutionary search [98]. The authors describe applications of various hybrid metaheuristics to problems ranging from car sequencing and graph coloring to scheduling. For example, a tabu search algorithm for the job-shop scheduling problem is presented, combining local search with complete enumeration as well as limited backtracking search. Several subsequent publications show the strong research interests in this direction and address issues such as distributed search [68] and visualization [33].

## 16.9 Decomposition Approaches

Problem decomposition approaches are another category of powerful techniques for combining different optimization techniques. Usually, a very hard-to-solve problem is decomposed into parts which can be dealt with more effectively. Some of the multi-stage approaches which we discussed in Section 16.4 already follow this basic idea. Large neighborhood search, heuristic cut and column generation in mixed integer programming, and constraint propagation by means of metaheuristics are three other prominent instances of successful decomposition techniques, which we consider in the following in more detail.

### 16.9.1 Exploring Large Neighborhoods

A common approach in more sophisticated local search-based metaheuristics is to search neighborhoods not by naive enumeration but by clever, more efficient algorithms. If the neighborhoods are chosen appropriately, they can be quite large and nevertheless an efficient search for a best neighbor is still possible in short time. Such techniques are known as *very large-scale neighborhood* (VLSN) search [2]; see also [24] for a recent survey. Many of today's combinations of local search-based metaheuristics with dynamic programming or MIP techniques follow this scheme. In the following, we present some examples.

In *Dynasearch* [27, 28] exponentially large neighborhoods are explored by dynamic programming. A neighborhood where the search is performed consists of all possible combinations of mutually independent simple search steps, and one Dynasearch move corresponds to a set of such simple steps that are executed in parallel in a single local search iteration. The required independence in the context of Dynasearch means that the individual simple moves do not interfere with each other; in this case, dynamic programming can be used to find a best combination. Ergun and Orlin [39] investigated several such neighborhoods in particular for the traveling salesman problem.

Particular types of large neighborhoods that can also be efficiently searched by dynamic programming are *cyclic and path exchange neighborhoods* [2, 3]. They are often applied in the context of problems where items need to be partitioned into disjoint sets. Examples of such problems are vehicle routing, capacitated minimum spanning tree, and parallel machine scheduling. In these neighborhoods, a series of items is exchanged between an arbitrary number of sets in a cyclic or path-like fashion, and a best move is determined by a shortest path-like algorithm.

Pesant and Gendreau [75] describe a generic framework for combining CP and local search. They view and model the original problem as well as the (large) neighborhoods as CP problems. Each of the neighborhoods is solved via a CP-based B&B that preserves solution feasibility. The framework allows for a relatively generic problem modeling while providing the advantages of local search. The authors solve a physician scheduling problem as well as the traveling salesman problem with time windows, and they approach them by tabu search in which large neighborhoods are searched by means of the CP-based B&B.

Puchinger et al. [84] describe a hybrid GA for a real-world glass cutting problem in which large neighborhoods are searched by means of B&B. The GA uses an order-based representation which is decoded using a greedy heuristic. B&B is applied with a certain probability, enhancing the decoding phase by generating locally optimal subpatterns. Reported results indicate that occasionally solving subpatterns to optimality often increases the overall solution quality.

Quite often, large neighborhoods are described in the form of MIPs and a MIP solver is applied for finding a good—or the best—neighbor. For example, Büdenbender et al. [23] present a tabu search hybrid for solving a real-world direct flight network design problem. Neighborhoods are created by fixing a large subset of the integer variables corresponding to the performed flights and allowing the other variables to be changed. CPLEX is used to solve the reduced problems corresponding to these neighborhoods. Diversification is achieved by closing flights frequently occurring in previously devised solutions. Other examples for MIP-based large neighborhood search can be found in Duarte et al. [35], where an iterated local search framework is applied to a real-world referee assignment problem, and in Prandstetter and Raidl [79] where several different MIP-based neighborhoods are searched within a VNS framework for a car sequencing problem.

Hu et al. [57] propose a VNS for the generalized minimum spanning tree problem. The approach uses two dual types of representations and exponentially large neighborhood structures. Best neighbors are identified by means of dynamic programming algorithms and—in case of the so-called global subtree optimization neighborhood—by solving an ILP formulation with CPLEX. Experimental results indicate that each considered neighborhood structure contributes to the overall excellent performance.

*Variable neighborhood decomposition search* (VNDS) [54] is a variant of VNS obtained by selecting the neighborhoods so as to obtain a problem decomposition. VNDS follows the usual VNS scheme, but the neighborhood structures and the local search are defined on subproblems rather than on the original problem. Given a

solution, all but  $k$  attributes (usually variables) are kept fixed. For each  $k$ , a neighborhood structure  $\mathcal{N}_k$  is defined. Local search only regards changes on the variables belonging to the subproblem it is applied to. Successful applications of VNDS include the edge-weighted  $k$ -cardinality tree problem [97] and supply chain management planning problems [61].

### 16.9.2 Cut and Column Generation by Metaheuristics

*Cutting plane algorithms* [105] are a powerful tool for solving complex MIPs. They start with a relaxation of the original problem in which most of the constraints—especially the integrality constraints—are omitted. This relaxation is solved, and then a *separation algorithm* is applied for finding further constraints that are fulfilled by an optimal solution to the original problem but are violated by the current solution to the relaxed problem. If such constraints, called *cuts*, could be identified, they are added to the relaxed LP, which is then solved again. This process is iterated until no further cuts can be found. If the final solution is infeasible, either a heuristic repair procedure may be applied, or the cutting plane algorithm is embedded in a B&B framework yielding an exact *branch-and-cut* algorithm.

Often, the subproblem of separating a cut (i.e., finding a valid inequality violated by the current LP solution) is difficult to solve by itself. In such cases, heuristics are often applied, and also fast metaheuristics have already been successfully used.

One example is the work from Augerat et al. [10], which uses a hierarchy consisting of a simple constructive heuristic, a randomized greedy method, and a tabu search for separating capacity constraints within a branch-and-cut algorithm for a capacitated vehicle routing problem. Another more recent example is the branch-and-cut algorithm for the diameter bounded minimum spanning tree problem by Gruber and Raidl [52], in which local search and tabu search techniques are used for separating so-called jump cuts.

One more example concerns the acceleration of Benders decomposition by local branching, as described by Rei et al. [89]. The basic principle of Benders decomposition is to project a MIP into the space of complicating integer variables only; real variables and the constraints involving them are replaced by corresponding constraints on the integer variables. These constraints, however, are not directly available but need to be dynamically generated. According to the classical method, an optimal solution to the relaxed master problem (including only the already separated cuts) is needed and an LP involving this solution must be solved in order to separate a single new cut. Rei et al. [89] improved this method by introducing phases of local branching on the original problem in order to obtain multiple feasible heuristic solutions. These solutions provide improved upper bounds on one hand, but also allow the derivation of multiple additional cuts before the relaxed master problem needs to be solved again.

*Column generation algorithms* can be seen as dual to cutting plane algorithms. Instead of starting with a reduced set of constraints and dynamically extending it,

the set of variables (which correspond to columns in the matrix notation of the MIP) is restricted, and further variables are iteratively added. Hereby, the essential subproblem, called *pricing problem*, is to identify variables whose inclusion will yield an improvement. Again, the pricing problem is often difficult by itself, and applying fast (meta-)heuristics is sometimes a meaningful option. If column generation is performed within an exact LP-based B&B framework, the approach is called *branch-and-price*.

Filho and Lorena [41] apply a heuristic column generation approach to graph coloring. A GA is used to generate initial columns and to solve the pricing problem at every iteration. Column generation is performed as long as the GA finds columns with negative reduced costs. The master problem is solved using CPLEX. Puchinger and Raidl [80, 82] describe an exact branch-and-price approach for the three-stage two-dimensional bin packing problem. Fast column generation is performed by applying a hierarchy of four methods: (a) a greedy heuristic, (b) an evolutionary algorithm, (c) solving a restricted form of the pricing problem using CPLEX, and finally (d) solving the complete pricing problem using CPLEX.

### 16.9.3 Using Metaheuristics for Constraint Propagation

In CP the mechanism of constraint propagation is used to reduce the domains of the variables at each node of the B&B search tree. Similarly to cut generation in mixed integer programming, the search space is reduced by deducing consequences from the current state of the search. Usually specialized and standard combinatorial algorithms are used [65].

Galinier et al. [46] present a tabu search procedure to speed up filtering for generalized all-different constraints. That is,

$$\text{SomeDifferent}(X, D, G) = \{(a_1, \dots, a_n) \mid a_i \in D_i \wedge a_i \neq a_j \forall (i, j) \in E(G)\}$$

is defined over a set of variables  $X = \{x_1, \dots, x_n\}$  with domains  $D = \{D_1, \dots, D_n\}$  and an underlying graph  $G = (X, E)$  [90]. The satisfiability of the constraint can be tested by solving a special graph coloring problem. Tabu search is first applied to see if it can color the graph. If it does not find a solution, an exact method is applied. In a second step a similar tabu search procedure is used to determine a large set of variable/value combinations that are feasible. Finally an exact filtering is applied to the remaining variable/value pairs checking if some of them can be excluded from the variable domains. Computational experiments show that the hybrid approach is comparable to the state of the art on data from a real-world work-force management problem and is significantly faster on random graph instances for the SomeDifferent constraint. The authors suppose that the idea of combining fast metaheuristics with exact procedures can speed up filtering procedures for other NP-hard constraints as well.

## 16.10 Summary and Conclusions

We have reviewed a large number of different possibilities for combining traditional metaheuristic strategies with each other or with algorithmic techniques coming from other fields. All these possibilities have their individual pros and cons, but the common underlying motivation is to exploit the advantages of the individual techniques in order to obtain a more effective hybrid system, benefiting from synergy. In fact, history clearly shows that the concentration on a single metaheuristic is rather restrictive for advancing the state of the art when tackling difficult optimization problems. Thus, designing hybrid systems for complex optimization problems is nowadays a natural process.

On the downside, metaheuristic hybrids are usually significantly more complex than classical “pure” strategies. The necessary development and tuning effort may be substantially higher than when using a straightforward out-of-the-box strategy. One should further keep in mind that a more complex hybrid algorithm does not automatically perform better—an adequate design and appropriate tuning is always mandatory, and the effort increases with the system’s complexity. Einstein’s advice of *keeping things as simple as possible, but not simpler* therefore is especially true also for metaheuristic hybrids.

We started by presenting a classification of metaheuristic hybrids in which we pointed out the different basic characteristics. Then we discussed several commonly used design templates. Note that these templates are not meant as a clear categorization of existing hybrid approaches: Many of the referenced examples from the literature can be argued to follow more than one design template, and occasionally the boundaries are fuzzy.

Finding initial or improved solutions by embedded methods might be the most commonly applied approach. Multi-stage combinations are sometimes straightforward for problems that naturally decompose into multiple levels and are also otherwise popular as they are typically easier to tune than more intertwined hybrids. The concept of decoder-based metaheuristics is also quite popular, as they can often be implemented quickly, once an appropriate construction heuristic is available. Solution merging was the next design template we discussed and for which numerous successful examples exist. Then we considered cases where metaheuristics are strategically guided by other techniques. In particular, solutions to relaxations of the original problem are frequently exploited in various ways. The reverse, strategic guidance of other techniques by metaheuristics, has been particularly successful in the field of mixed integer programming, where such strategies can help to find good approximate solutions early within an exact B&B-based method. Last but not least, there are several different decomposition approaches: Exploring large neighborhoods by specialized algorithms has become particularly popular over the last years, and occasionally metaheuristics are applied to solve separation or pricing problems in more complex MIP approaches and propagation subproblems in CP.

As an important final advice for the development of well-performing metaheuristic hybrids, the authors would like to recommend (1) the careful search of the literature for the most successful optimization approaches for the problem at hand or for

similar problems and (2) the study of clever ways of combining the most interesting features of the identified approaches. We hope this chapter provides a starting point and some useful references for this purpose.

**Acknowledgments** Günther R. Raidl is supported by the Austrian Science Fund (FWF) under grant 811378 and by the Austrian Exchange Service (Acciones Integradas, grant 13/2006). NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program. Christian Blum is supported by grants TIN2005-08818 (OPLINK) and TIN2007-66523 (FORMALISM) of the Spanish government, and by the EU project FRONTS (FP7-ICT-2007-1). He also acknowledges support from the *Ramón y Cajal* program of the Spanish Ministry of Science and Technology.

## References

1. Aggarwal, C., Orlin, J., Tai, R.: Optimized crossover for the independent set problem. *Oper. Res.* **45**, 226–234 (1997)
2. Ahuja, R.K., Ergun, Ö., Orlin, J.B., Punnen, A.P.: A survey of very large-scale neighborhood search techniques. *Discrete Appl. Math.* **123**(1–3), 75–102 (2002)
3. Ahuja, R.K., Orlin, J., Sharma, D.: Multi-exchange neighborhood search algorithms for the capacitated minimum spanning tree problem. *Math. Programming* **91**(1), 71–97 (2001)
4. Ahuja, R., Orlin, J., Tiwari, A.: A greedy genetic algorithm for the quadratic assignment problem. *Comput. Oper. Res.* **27**, 917–934 (2000)
5. Al-Shihabi, S.: Ants for sampling in the nested partition algorithm. In: Blum et al. [22], pp. 11–18.
6. Alba, E. (ed.): *Parallel Metaheuristics: A New Class of Algorithms*. Wiley, (2005)
7. Almeida, F., Blesa Aguilera, M.J., Blum, C., Moreno Vega, J.M., Pérez Pérez, M., Roli, A., Sampels, M. (eds.) *Proceedings of HM 2006 – Third International Workshop on Hybrid Metaheuristics*, Lecture Notes in Computer Science, vol. 4030 Springer, Berlin (2006)
8. Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J.: On the solution of the traveling salesman problem. *Documenta Mathematica, Extra Volume ICM III*, 645–656 (1998)
9. Applegate, D.L., Bixby, R.E., Chvátal, V., Cook, W.J.: *The Traveling Salesman Problem: A Computational Study*. Princeton Series in Applied Mathematics. Princeton University Press, Princeton, NJ (2007)
10. Augerat, P., Belenguer, J.M., Benavent, E., Corberan, A., Naddef, D.: Separating capacity constraints in the CVRP using tabu search. *Eur. J. Oper. Res.* **106**(2), 546–557 (1999)
11. Barahona, F., Anbil, R.: The volume algorithm: Producing primal solutions with a subgradient method. *Math. Programming, Series A*, **87**(3), 385–399 (2000)
12. Bartz-Beielstein, T., Blesa Aguilera, M.J., Blum, C., Naujoks, B., Roli, A., Rudolph, G., Sampels, M. (eds.): *Proceedings of HM 2007 – Fourth International Workshop on Hybrid Metaheuristics*, Lecture Notes in Computer Science. vol. 4771 Springer, Berlin (2007)
13. Beck J. Christopher: Solution-guided multi-point constructive search for job shop scheduling. *J. Artif. Intell. Res.* **29**, 49–77 (2007)
14. Binato, S., Hery, W.J., Loewenstern, D., Resende, M.G.C.: A GRASP for job shop scheduling. In: Ribeiro, C.C., Hansen, P. (eds.) *Essays and Surveys on Metaheuristics*, pp. 59–79. Kluwer Academic Publishers, Dordrecht (2001)
15. Blesa Aguilera, M.J., Blum, C., Roli, A., Sampels, M., (eds.): *Proceedings of HM 2005 – Second International Workshop on Hybrid Metaheuristics*. Lecture Notes in Computer Science. vol. 3636 Springer, Berlin (2005)

16. Blum, C.: Beam-ACO: Hybridizing ant colony optimization with beam search: An application to open shop scheduling. *Comput. Oper. Res.* **32**(6):1565–1591, 2005.
17. Blum, C.: A new hybrid evolutionary algorithm for the  $k$ -cardinality tree problem. In: Proceedings of the Genetic and Evolutionary Computation Conference 2006, pp. 515–522. ACM Press, New York, NY, July 8–12 (2006)
18. Blum, C.: Beam-ACO for simple assembly line balancing. *INFORMS J. Comput.* **20**(4), 618–627 (2008)
19. Blum, C., Blesa, M.: Combining ant colony optimization with dynamic programming for solving the  $k$ -cardinality tree problem. In: Proceedings of IWANN 2005 – 8th International Work-Conference on Artificial Neural Networks, Computational Intelligence and Bio-inspired Systems, number 3512 in Lecture Notes in Computer Science, pp. 25–33, Springer, Berlin (2005)
20. Blum, C., Blesa Aguilera, M.J., Roli, A., Sampels, M.: (eds.) *Hybrid Metaheuristics – An Emerging Approach to Optimization*, volume 114 of Studies in Computational Intelligence. Springer, Berlin (2008)
21. Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surveys* **35**(3), 268–308 (2003)
22. Blum, C., Roli, A., Sampels, M. (eds.): *Proceedings of HM 2004 – First International Workshop on Hybrid Metaheuristics*, Valencia, Spain (2004)
23. Büdenbender, K., Grünert, T., Sebastian, H.-J.: A hybrid tabu search/branch-and-bound algorithm for the direct flight network design problem. *Transportation Sci.* **34**(4), 364–380 (2000)
24. Chiarandini, M., Dumitrescu, I., Stützle, T.: Very large-scale neighborhood search: Overview and case studies on coloring problems. In: Blum et al. [20], pp. 117–150
25. Chu, P.C., Beasley, J.E.: A genetic algorithm for the multidimensional knapsack problem. *J. Heuristics* **4**, 63–86 (1998)
26. Cohoon, J., Hegde, S., artin, W., Richards, D.: Punctuated equilibria: A parallel genetic algorithm. In: Grefenstette, J. (ed.) *Proceedings of the Second International Conference on Genetic Algorithms*, pp. 148–154. Lawrence Erlbaum Associates, Mahwah, NJ 1987
27. Congram, R.K.: Polynomially searchable exponential neighbourhoods for sequencing problems in combinatorial optimisation. PhD thesis, University of Southampton, Faculty of Mathematical Studies, UK (2000)
28. Congram, R.K., Potts, C.N., van de Velde, S.L.: An iterated Dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS J. Comput.* **14**(1), 52–67 (2002)
29. Cotta, C.: A study of hybridisation techniques and their application to the design of evolutionary algorithms. *AI Commun.* **11**(3–4), 223–224 (1998)
30. Cotta, C., Troya, J.M.: Embedding branch and bound within evolutionary algorithms. *Appl. Intell.* **18**, 137–153 (2003)
31. Danna, E., Rothberg, E., Le Pape, C.: Exploring relaxation induced neighborhoods to improve MIP solutions. *Math. Programming, Series A*, **102**, 71–90 (2005)
32. Denzinger, J., Offermann, T.: On cooperation between evolutionary algorithms and other search paradigms. In: Porto, W. et al. (eds.) *Proceedings of the 1999 Congress on Evolutionary Computation (CEC)*, vol. 3, pp. 2317–2324. IEEE Press, Piscataway, NJ (1999)
33. Dooms, G., Van Hentenryck, P., Michel, L.: Model-driven visualizations of constraint-based local search. In: Bessiere, C. (ed.) *Principles and Practice of Constraint Programming – CP 2007, 13th International Conference*, Lecture Notes in Computer Science, vol. 4741, pp. 271–285. Springer, Berlin (2007)
34. Dowland, K.A., Herbert, E.A., Kendall, G., Burke, E.: Using tree search bounds to enhance a genetic algorithm approach to two rectangle packing problems. *Eur. J. Oper. Res.* **168**(2), 390–402 (2006)
35. Duarte, A.R., Ribeiro, C.C., Urrutia, S.: A hybrid ILS heuristic to the referee assignment problem with an embedded MIP strategy. In: Bartz-Beielstein et al. [12], pp. 82–95

36. Dumitrescu, I., Stuetzle, T.: Combinations of local search and exact algorithms. In: Raidl, G.R. et al. (eds.) *Applications of Evolutionary Computation*, Lecture Notes in Computer Science, vol. 2611, pp. 211–223. Springer, Berlin (2003)
37. El-Abd, M., Kamel, M.: A taxonomy of cooperative search algorithms. In: Blesa Aguilera et al. [15], pp. 32–41
38. Eremeev, A.V.: On complexity of optimal recombination for binary representations of solutions. *Evol. Comput.* **16**(1), 127–147 (2008)
39. Ergun, Ö., Orlin, J.B.: A dynamic programming methodology in very large scale neighborhood search applied to the traveling salesman problem. *Discrete Optimization*, **3**(1), 78–85 (2006)
40. Feo, T.A., Resende, M.G.C.: Greedy randomized adaptive search procedures. *J. Global Optimization* **6**, 109–133 (1995)
41. Filho, G.R., Lorena, L.A.N.: Constructive genetic algorithm and column generation: An application to graph coloring. In: Chuen, L.P. (ed.) *Proceedings of APORS 2000*, the Fifth Conference of the Association of Asian-Pacific Operations Research Societies Within IFORS, Singapore (2000)
42. Fischetti, M., Lodi, A.: Local branching. *Math. Programming, Series B* **98**, 23–47 (2003)
43. Fischetti, M., Polo, C., Scantamburlo, M.: Local branching heuristic for mixed-integer programs with 2-level variables, with an application to a telecommunication network design problem. *Networks* **44**(2), 61–72 (2004)
44. Fleurent, C., Glover, F.: Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory. *INFORMS J. Comput.* **11**, 198–204 (1999)
45. Focacci, F., Laburthe, F., Lodi, A.: Local search and constraint programming: LS and CP illustrated on a transportation problem. In: Milano, M. (ed.) *Constraint and Integer Programming. Towards a Unified Methodology*, pp. 293–329. Kluwer Academic Publishers, 2004.
46. Galinier, P., Hertz, A., Paroz, S., Pesant, G.: Using local search to speed up filtering algorithms for some NP-hard constraints. In: Perron and Trick [74], p. 298–302
47. Gilmour, S., Dras, M.: Kernelization as heuristic structure for the vertex cover problem. In: Dorigo, M., et al., editors, *Proceedings of ANTS 2006 – 5th International Workshop on Ant Colony Optimization and Swarm Intelligence*. Lecture Notes in Computer Science, vol. 4150, pp. 452–459. Springer, Berlin (2006)
48. Glover, F.: Surrogate constraints. *Operations Research* **16**(4), 741–749 (1968)
49. Glover, F.: Parametric tabu-search for mixed integer programming. *Computers and Operations Research*, **33**(9), 2449–2494 (2006)
50. Glover, F., Laguna, M., Martí, R.: Fundamentals of scatter search and path relinking. *Control and Cybernetics* **39**(3), 653–684 (2000)
51. Goldberg, D.E.: *Genetic Algorithms in Search, Optimization, and Learning*. Addison-Wesley, Reading, MA (1989)
52. Gruber, M., Raidl, G.R.: (Meta-)heuristic separation of jump cuts for the bounded diameter minimum spanning tree problem. In: Hansen et al. [53].
53. Hansen, P., Maniezzo, V., Fischetti, M., Stuetzle, T. (eds.) *Proceedings of Matheuristics 2008: Second International Workshop on Model Based Metaheuristics*, Bertinoro, Italy (2008)
54. Hansen, P., Mladenovic, N., Perez-Britos, D.: Variable neighborhood decomposition search. *J. Heuristics* **7**(4), 335–350 (2001)
55. Hansen, P., Mladenović, N., Urosević, D.: Variable neighborhood search and local branching. *Comput. Oper. Res.* **33**(10), 3034–3045 (2006)
56. Haouari, M., Siala, J.C.: A hybrid Lagrangian genetic algorithm for the prize collecting Steiner tree problem. *Comput. Oper. Res.* **33**(5), 1274–1288 (2006)
57. Hu, B., Leitner, M., Raidl, G.R.: Combining variable neighborhood search with integer linear programming for the generalized minimum spanning tree problem. *J. Heuristics* **14**(5), 473–479 (2008)

58. Hu, B., Raidl, G.R.: Effective neighborhood structures for the generalized traveling salesman problem. In: van Hemert, J., Cotta, C. (eds.) Evolutionary Computation in Combinatorial Optimisation – EvoCOP 2008, Lecture Notes in Computer Science, vol. 4972 pp. 36–47. Springer, Berlin (2008)
59. Kellerer, H., Pferschy, U., Pisinger, D.: Knapsack Problems. Springer, Berlin (2004)
60. Klau, G.W., Lesh, N., Marks, J., Mitzenmacher, M.: Human-guided search: Survey and recent results. *Journal of Heuristics* (2007, submitted)
61. Lejeune, M.A.: A variable neighborhood decomposition search method for supply chain management planning problems. *Eur. J. Oper. Res.* **175**(2), 959–976 (2006)
62. Maniezzo, V.: Exact and approximate nondeterministic tree-search procedures for the quadratic assignment problem. *INFORMS J. Comput.* **11**(4), 358–369 (1999)
63. Maniezzo, V., Carbonaro, A.: An ANTS heuristic for the frequency assignment problem. *Future Generation Comput. Syst.* **16**, 927–935 (2000)
64. Maniezzo, V., Hansen, P., Voss, S. (eds.): Proceedings of Matheuristics 2006: First International Workshop on Mathematical Contributions to Metaheuristics, Bertinoro, Italy (2006)
65. Marriott, K., Stuckey, P.J.: Introduction to Constraint Logic Programming. MIT Press, Cambridge, MA, (1998)
66. Martin, O., Otto, S.W., Felten, E.W.: Large-step Markov chains for the traveling salesman problem. *Complex Systems* **5**, 299–326 (1991)
67. Meyer, B., Ernst, A.: Integrating ACO and constraint propagation. In: Dorigo, M., et al. (eds.) Proceedings of ANTS 2004 – Fourth International Workshop on Ant Colony Optimization and Swarm Intelligence, Lecture Notes in Computer Science, vol. 3172, pp. 166–177. Springer, Berlin (2004)
68. Michel, L., See, A., Van Hentenryck, P.: Distributed constraint-based local search. In: Benhamou, F. (ed.) Principles and Practice of Constraint Programming – CP 2006, 12th International Conference, Lecture Notes in Computer Science, vol. 4204, pp. 344–358. Springer, Berlin (2006)
69. Moscato, P.: Memetic algorithms: A short introduction. In: Corne, D., et al. (eds.) New Ideas in Optimization, p. 219–234. McGraw Hill, New York, NY (1999)
70. Nagar, A., Heragu, S.S., Haddock, J.: A meta-heuristic algorithm for a bi-criteria scheduling problem. *Ann. Oper. Res.* **63**, 397–414 (1995)
71. Neto, T., Pedroso, J.P.: GRASP for linear integer programming. In: Sousa, J.P., Resende, M.G.C. (eds.) Metaheuristics: Computer Decision Making, Combinatorial Optimization Book Series, pp. 545–574. Kluwer Academic Publishers, Dordrecht (2003)
72. Ow, P.S., Morton, T.E.: Filtered beam search in scheduling. *Int. J. Prod. Res.* **26**, 297–307 (1988)
73. Pedroso, J.P.: Tabu search for mixed integer programming. In: Rego, C., Alidaee, B. (eds.) Metaheuristic Optimization via Memory and Evolution, Operations Research/Computer Science Interfaces Series, vol. 30, pp. 247–261. Springer, Berlin (2005)
74. Perron, L., Trick, M.A. (eds.): Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems – CPAIOR 2008, 5th International Conference, vol. 5015 Lecture Notes in Computer Science. Springer, Berlin (2008)
75. Pesant, G., Gendreau, M.: A constraint programming framework for local search methods. *J. Heuristics* **5**(3), 255–279 (1999)
76. Pirkwieser, S., Raidl, G.R., Puchinger, J.: Combining Lagrangian decomposition with an evolutionary algorithm for the knapsack constrained maximum spanning tree problem. In: Cotta, C., van Hemert, J. (eds.) Evolutionary Computation in Combinatorial Optimization – EvoCOP 2007, Lecture Notes in Computer Science, vol. 4446, pp. 176–187. Springer, Berlin (2007)
77. Pisinger, D.: Core problems in knapsack algorithms. *Operations Research* **47**, 570–575 (1999)
78. Plateau, A., Tachat, D., Tolla, P.: A hybrid search combining interior point methods and metaheuristics for 0–1 programming. *Int. Trans. Oper. Res.* **9**, 731–746 (2002)

79. Prandstetter, M., Raidl, G.R.: An integer linear programming approach and a hybrid variable neighborhood search for the car sequencing problem. *Eur. J. Oper. Res.* **191**(3), 1004–1022 (2008)
80. Puchinger, J., Raidl, G.R.: An evolutionary algorithm for column generation in integer programming: An effective approach for 2D bin packing. In: Yao, X., et al. (eds.) *Parallel Problem Solving from Nature – PPSN VIII*, vol. 3242 Lecture Notes in Computer Science, pp. 642–651. Springer, Berlin (2004)
81. Puchinger, J., Raidl, G.R.: Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In: Proceedings of the First International Work-Conference on the Interplay Between Natural and Artificial Computation, Part II, Lecture Notes in Computer Science, vol. 3562, pp. 41–53. Springer, Berlin (2005)
82. Puchinger, J., Raidl, G.R.: Models and algorithms for three-stage two-dimensional bin packing. *Eur. J. Oper. Res.* **183**, 1304–1327 (2007)
83. Puchinger, J., Raidl, G.R.: Bringing order into the neighborhoods: Relaxation guided variable neighborhood search. *J. Heuristics* **14**(5), 457–472 (2008)
84. Puchinger, J., Raidl, G.R., Koller, G.: Solving a real-world glass cutting problem. In: Gottlieb, J., Raidl, G.R. (eds.) *Evolutionary Computation in Combinatorial Optimization – EvoCOP 2004*, Lecture Notes in Computer Science, vol. 3004, pp. 162–173. Springer, Berlin (2004)
85. Puchinger, J., Raidl, G.R., Pferschy, U.: The core concept for the multidimensional knapsack problem. In: Gottlieb, J., Raidl, G.R. (eds.) *Evolutionary Computation in Combinatorial Optimization – EvoCOP 2006*, Lecture Notes in Computer Science, vol. 3906, pp. 195–208. Springer (2006)
86. Raidl, G.R.: An improved genetic algorithm for the multiconstrained 0–1 knapsack problem. In: Fogel, D.B., et al. (eds.) *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, pp. 207–211. IEEE Press, Piscataway, NJ (1998)
87. Raidl, G.R.: A unified view on hybrid metaheuristics. In: Almeida et al. [7], pp. 1–12
88. Raidl, G.R., Puchinger, J.: Combining (integer) linear programming techniques and metaheuristics for combinatorial optimization. In: Blum et al. [20], pp. 31–62 (2008)
89. Rei, W., Cordeau, J.-F., Gendreau, M., Sorianio, P.: Accelerating Benders decomposition by local branching. *INFORMS J. Comput.* **21**, 333–345 (2009)
90. Richter, Y., Freund, A., Naveh, Y.: Generalizing AllDifferent: The SomeDifferent constraint. In: Benhamou, F. (ed.) *Principles and Practice of Constraint Programming*, 12th International Conference, CP 2006, volume 4204 of Lecture Notes in Computer Science, pp. 468–483. Springer, Berlin (2006)
91. Rothberg, E.: An evolutionary algorithm for polishing mixed integer programming solutions. *INFORMS J. Comput.* **19**(4), 534–541 (2007)
92. Shi, L., Ólafsson, S.: Nested partitions method for global optimization. *Oper. Res.* **48**(3), 390–407 (2000)
93. Shi, L., Ólafsson, S., Chen, Q.: An optimization framework for product design. *Manage. Sci.* **47**(12), 1681–1692 (2001)
94. Talbi, E.: A taxonomy of hybrid metaheuristics. *J. Heuristics* **8**(5), 541–565 (2002)
95. Talukdar, S., Baeretzen, L., Gove, A., de Souza, P.: Asynchronous teams: Cooperation schemes for autonomous agents. *J. Heuristics* **4**, 295–321 (1998)
96. Tamura, H., Hirahara, A., Hatono, I., Umano, M.: An approximate solution method for combinatorial optimisation. *Trans. Soc. Instrum. Control Engineers* **130**, 329–336 (1994)
97. Urosevic, D., Brimberg, J., Mladenovic, N.: Variable neighborhood decomposition search for the edge weighted  $k$ -cardinality tree problem. *Comput. Oper. Res.* **31**(8), 1205–1213 (2004)
98. Van Hentenryck, P., Michel, L.: *Constraint-Based Local Search*. MIT Press, Cambridge, MA (2005)
99. Vasquez, M., Hao, J.-K.: A hybrid approach for the 0–1 multidimensional knapsack problem. In: Nebel, B. (ed.) *Proceedings of the 17th International Joint Conference on Artificial Intelligence, IJCAI 2001*, pp. 328–333, Seattle, Washington (2001) Morgan Kaufman

100. Vasquez, M., Vimont, Y.: Improved results on the 0–1 multidimensional knapsack problem. *Eur. J. Oper. Res.* **165**(1), 70–81 (2005)
101. Walshaw, C.: Multilevel refinement for combinatorial optimisation: Boosting metaheuristic performance. In: Blum et al. [20], pp. 261–289 (2008)
102. Watson, J.-P., Beck, J.C.: A hybrid constraint programming/local search approach to the job-shop scheduling problem. In: Perron and Trick [74], pp. 263–277.
103. Watson, J.-P., Howe, A.E., Darrell Whitley, L.: Deconstructing Nowicki and Smutnicki’s i-TSAB tabu search algorithm for the job-shop scheduling problem. *Comput. Oper. Res.* **33**(9), 2623–2644 (2006)
104. Wolpert, D., Macready, W.: No free lunch theorems for optimization. *IEEE Trans. Evol. Comput.* **1**(1), 67–82 (1997)
105. Wolsey, L.A.: Integer Programming. Wiley-Interscience, New York, NY 1998.

# Chapter 17

## Parallel Meta-heuristics

Teodor Gabriel Crainic and Michel Toulouse

**Abstract** We present a state-of-the-art survey of parallel meta-heuristic strategies, developments, and results. We discuss general design and implementation principles that apply to most meta-heuristic classes and instantiate these principles for neighborhood and population-based meta-heuristics. We also identify a number of trends and promising research directions.

### 17.1 Introduction

Meta-heuristics are widely acknowledged as essential tools in addressing difficult problems in numerous and diverse fields. Meta-heuristics actually often offer the only practical approach to solving complex problems of realistic dimensions.

Even using meta-heuristics, however, the limits of what may be solved in “reasonable” computing times are still reached rapidly for many problem settings, at least much too rapidly for the growing needs of research and industry alike. Heuristics do not, in general, guaranty optimality. Moreover, the performance often depends on the particular problem setting and instance characteristics. Consequently, a major issue in meta-heuristic design and calibration is not only how to build them for maximum performance but also how to make them *robust*, in the sense

---

Teodor Gabriel Crainic

Département de management et technologie, École des sciences de la gestion, and  
Interuniversity Research Centre on Enterprise Networks, Logistics, and Transportation  
(CIRRELT), Université du Québec à Montréal, Montréal, QC, Canada  
e-mail: teodorgabriel.crainic@cirrelt.ca

Michel Toulouse

Interuniversity Research Centre on Enterprise Networks, Logistics, and Transportation  
(CIRRELT), Université de Montréal, Montréal, QC, Canada  
e-mail: michel.toulouse@cirrelt.ca

of offering a consistently high level of performance over a wide variety of problem settings and characteristics.

*Parallel meta-heuristics* aim to address both issues. Of course, the first goal is to solve larger problem instances in reasonable computing times. In appropriate settings, such as cooperative multi-search strategies, parallel meta-heuristics also prove to be much more robust than sequential versions in dealing with differences in problem types and characteristics. They also require less extensive, and expensive, parameter calibration efforts.

The objective of this chapter is to paint a general picture of the parallel meta-heuristic field. More specifically, we aim to present a state-of-the-art survey of the main parallel meta-heuristic ideas and strategies and discuss general design and implementation principles that apply to most meta-heuristic classes, to instantiate these principles for neighborhood- and population-based meta-heuristics, and to identify a number of trends and promising research directions.

The parallel meta-heuristic field is very broad, while the space available for this chapter imposes hard choices and limits the presentation. In addition to the references provided in the following sections, the reader may consult a number of surveys, taxonomies, and syntheses of parallel meta-heuristics, some addressing methods based on particular methodologies, while others address the field in more comprehensive terms. Methodology dedicated syntheses may be found in [4, 74–76, 119] for parallel simulated annealing, [2, 16, 17, 93, 104, 132] for genetic-based evolutionary methods, [26, 34, 41, 72, 151] for tabu search, [59] for scatter search, [14, 52, 81] for ant-colony methods, and [100] for Variable Neighborhood Search (VNS). Surveys and syntheses that address more than one methodology may be found in [27, 36–38, 42, 79, 84, 113, 150].

The chapter is organized as follows. Section 17.2 is dedicated to a general discussion of basic meta-heuristics design principles, the corresponding potential for parallel computing, and the taxonomy we use to structure the presentation. Section 17.3 addresses strategies focusing on accelerating computing-intensive tasks without modifying the basic algorithmic design. Methods based on the decomposition of the search space are treated in Section 17.4, while strategies based on the simultaneous exploration of the search space by several independent meta-heuristics constitutes the topic of Section 17.5. The different cooperation principles and strategies are the subject of Section 17.6 and we conclude in Section 17.7.

## 17.2 Meta-heuristics and Parallelism

This section is dedicated to an overview of the main classes of meta-heuristics and the associated potential for parallel computing. The latter is completed by a discussion of performance indicators for parallel meta-heuristics. The section concludes with the criteria used in this chapter to describe and classify parallelization strategies for meta-heuristics.

### 17.2.1 Heuristics and Meta-heuristics

Given a set of objects, each with an associated contribution, an objective function computing the value of a subset of objects out of their respective contributions, and the feasibility rules specifying how subsets may be built, combinatorial optimization problems aim to select a subset of objects satisfying these rules and such that the value of the function is the highest/lowest among all possible combinations. Many problems of interest may be represented through this framework, including design, routing, and scheduling. Combinatorial optimization problems are usually formulated as (mixed) integer optimization programs. To define notation, assume that one desires to minimize an objective function  $f(x)$ , linear or not, subject to  $x \in \mathcal{X} \subseteq \mathbb{R}^n$ . The set  $\mathcal{X}$  collects constraints on the *decision variables*  $x$  and defines the *feasible domain*. Decision variables are generally non-negative and all or some may be compelled to take discrete values. One seeks a globally *optimal solution*  $x^* \in \mathcal{X}$  such that  $f(x^*) \leq f(x)$  for all  $x \in \mathcal{X}$ .

In most cases, such formulations are difficult to solve for realistically sized problem instances, the main issue being the number of feasible solutions – subsets of objects – that grows exponentially with the number of objects in the initial set. Once various methods have been applied to re-formulate the problem and bound the region where the optimal solution is to be found, most solution methods are based on some form of exploration of the set of feasible (and sometimes, infeasible) solutions. Explicit enumeration is normally out of the question and the search for the optimal solution proceeds by implicit enumeration. Branch-and-bound (and price, and cut, and ...) methods are typical of such approaches and make up one of the strategies of choice used in the search for optimal solutions to combinatorial problems. Unfortunately, these methods fail for many instances, even when parallel implementations are used. Thus, heuristics have been, and continue to be, an essential methodology in addressing combinatorial optimization formulations, often offering the only practical alternative when dealing with problem instances of realistic dimensions and characteristics.

A *heuristic* is any procedure that aims to identify a “good” feasible solution  $\tilde{x} \in \mathcal{X}$ . Of course, one would like  $\tilde{x}$  to be identical to  $x^*$  (if the latter is unique) or  $f(\tilde{x})$  to be equal to  $f(x^*)$ . For most heuristics, however, one can only hope (and for some, prove) that  $f(\tilde{x})$  is “close” to  $f(x^*)$ . At the core of many heuristics, one finds an *improving* iterative procedure that *moves* from a given solution to a solution in its *neighborhood*, which is better in terms of the objective function value or some other measure based on the solution characteristics. Thus, at each iteration, such a *local search (LS)* procedure identifies and evaluates solutions in the neighborhood of the current solution, selects the best one relative to given criteria, and implements the transformations required to establish the selected solution as the current one. The procedure iterates until no further improvement is possible.

Figure 17.1 illustrates the local search template (for a minimization problem). The input value  $x$  is usually a feasible initial solution. The expression  $\mathcal{N}(x)$  denotes the *neighborhood* of  $x$ , that is, the set of solutions, called *neighbors*, that can

---

**input:**  $x \in \mathcal{X}$  an initial solution  
 $y \leftarrow Neighbor(\mathcal{N}(x))$   
**while** ( $c(y) < c(x)$ )  
     $x \leftarrow y$   
     $y \leftarrow Neighbor(\mathcal{N}(x))$   
**return**  $x$

---

**Fig. 17.1** Local search template.

be reached from  $x$  through a predefined transformation of  $x$  called *move*. Such a transformation may be simple, e.g., complement the value of an integer-valued variable, or more complex made up of a sequence of operations, e.g.,  $\lambda$ -opt modifications of routes in vehicle routing problems (VRPs). The value of any solution  $x$  is given by  $c(x)$ , which may be  $f(x)$ , or more or less loosely related to it or an entirely different function. Figure 17.2 illustrates the procedure *Neighbor*, which identifies, evaluates, and selects solutions in the neighborhood  $\mathcal{N}(x)$  of the current solution  $x$ ,  $m(x)$  standing for a neighbor of  $x$  obtained by applying the move  $m$ . The parameter *neighbor\_selection* indicates whether the procedure returns the best solution in  $\mathcal{N}(x)$  with respect to  $c(x)$  or the first solution better than  $x$  found while exploring  $\mathcal{N}(x)$ . One denotes these two cases as *best-improvement* (also called *steepest descent* or *ascent* when maximizing) and *first-improvement*, respectively.

---

**input:**  $x$ ; *neighbor\_selection*  
 $NewCurrentSolution \leftarrow x$   
**if** (*neighbor\_selection* = first improvement) **then**  
     $y \leftarrow m(x); \mathcal{N}(x) \leftarrow \mathcal{N}(x) \setminus y$   
    **while** ( $c(y) \geq c(x)$  or  $\mathcal{N}(x) \neq \emptyset$ )  $y \leftarrow m(x); \mathcal{N}(x) \leftarrow \mathcal{N}(x) \setminus y$   
     $NewCurrentSolution \leftarrow y$   
**if** (*neighbor\_selection* = best improvement) **then**  
    **while** ( $\mathcal{N}(x) \neq \emptyset$ )  
         $y \leftarrow m(x); \mathcal{N}(x) \leftarrow \mathcal{N}(x) \setminus y$   
        **if** ( $c(y) < c(NewCurrentSolution)$ ) **then**  $NewCurrentSolution \leftarrow y$   
**return**  $NewCurrentSolution$

---

**Fig. 17.2** Procedure *Neighbor*.

Local search heuristics update the current solution  $x$  only when it can be improved and never backtrack to a previous solution. They therefore stop as soon as a local optimum solution is found. This inability to continue past the first encountered local optimum is a major drawback of classical heuristic schemes. Moreover, such procedures are unable to react and adapt to particular problem instances. Re-starting and randomization strategies, as well as combinations of simple heuristics offer only partial and largely unsatisfactory answers to these issues. The class of modern heuristics known as *meta-heuristics* aims to address these challenges.

Meta-heuristics have been defined as master strategies (heuristics) that guide and modify other heuristics to produce solutions beyond those normally identified

by heuristics such as local search [69, 71]. Compared to exact search methods, such as branch-and-bound, meta-heuristics cannot generally guarantee a systematic exploration of the entire solution space. Instead, they attempt to examine only parts thereof where, according to certain criteria, one believes good solutions may be found. Well-designed meta-heuristics avoid getting trapped in local optima or sequences of visited solutions (*cycling*) and provide reasonable assurance that the search has not overlooked promising regions.

Meta-heuristics are iterative procedures, which move at each iteration toward “good” solutions in the neighborhood of the current solution or of a suitably selected subset. Unlike local search heuristics, however, meta-heuristics may move to not-necessarily improving solutions, which constitutes the main mechanism to avoid stopping at local optima. Additional mechanisms control the evolution of the meta-heuristic to avoid cycling, learn from previous moves and encountered solutions, and provide for a thorough search. Meta-heuristics explore a *search space* that may be the feasible domain of the problem at hand or only loosely based on it (e.g., the search space may include infeasible solutions or may be restricted to a subset of variables only). Many meta-heuristics have been proposed. From the point of view of parallel-strategy design, however, it is convenient to discuss them according to whether their main search mechanism is based on neighborhoods or populations.

Neighborhood-based meta-heuristics implement explicitly moves to solutions selected within given neighborhoods and generally proceed following a single trajectory in the search space. Tabu search, simulated annealing, guided local search, variable neighborhood search, greedy randomized adaptive search, and iterated local search, belong to this category of meta-heuristics, and define high-level mechanisms to guide local search explorations of the search space. The high-level mechanisms monitor the status of the search, determine when particular phases (e.g., diversification and intensification) should start, select neighborhoods, and local-search procedures, etc. Figure 17.3 displays the general design idea for neighborhood-based meta-heuristics, emphasizing the two major nested loops of their usual implementation: An outer loop implements the high-level meta-heuristic controlling (guiding) the global search and the selected local search procedure, while the inner loop executes the local search. Notice that the  $\text{Neighbor}(\mathcal{N}(x))$  procedure called within the local search implicitly adds a third-level nested loop.

---

```

Identify an initial solution
while (Termination criterion not satisfied)
    Monitor and update global search status
    Update meta-heuristic guidance information including:
        selection of LS procedures
        selection of neighborhoods
    while (LS termination criterion not satisfied)
        Apply local move to current solution  $x$ :  $y \leftarrow \text{Neighbor}(\mathcal{N}(x))$ 

```

---

**Fig. 17.3** The neighborhood-based meta-heuristic idea.

Population-based meta-heuristics use a set of solutions to concurrently sample different regions of the solution space. The search moves to new solutions by recombining elements from different solutions in the current population. We find in this group evolutionary methods (genetic algorithms), scatter search, and path relinking. For the purposes of this chapter, we also include in the group ant-based methods and other swarm-based algorithms. Figure 17.4 displays the general algorithmic idea of population-based meta-heuristics. There are several loops in an implementation of a population-based method: the generation of the initial population, the computation of solution values, which could involve the entire population, and of the global information, the generation of new individuals, which may involve local search either as an individual improvement mechanism (e.g., genetic algorithms and scatter search) or as a trajectory between two individuals (path relinking).

---

Generate an initial population  $P$  of size  $n$   
*while* (Termination criterion not satisfied)  
  Monitor and update global search status and  
  Update meta-heuristic guidance information including  
    selection of elite sub-populations and  
    computation of corresponding solution values  
  Compute global information (average fitness, pheromone matrix)  
  Update population (including local search)

---

**Fig. 17.4** The population-based metaheuristic idea.

### 17.2.2 Sources of Parallelism

Parallel/distributed computing means that several processes work simultaneously on several processors solving a given problem instance. Parallelism thus follows from a decomposition of the total computational load and the distribution of the resulting tasks to available processors. The decomposition may concern the algorithm, the problem-instance data, or the problem structure (e.g., mathematical or attribute-based [28, 29] decomposition). In the first case, denoted *functional parallelism*, different tasks, possibly working on the “same” data, are allocated to different processors and run in parallel, possibly exchanging information. The second is denoted *data parallelism* or *domain decomposition* and refers to the case where the problem domain, or the associated search space, is decomposed and a particular solution methodology is used to address the problem on each of the resulting components of the search space. The third case is quite recent and generates tasks by decomposing the problem along sets of attributes. The decomposition could be performed through mathematical programming techniques or heuristically. Then, some tasks work on subproblems corresponding to particular sets of attributes (i.e., part of the original search space), while others combine subproblem solutions into whole solutions to the original problem. According to how “small” or “large” are the tasks

in terms of algorithm work or search space, the parallelization is denoted *fine-* or *coarse-grained*, respectively.

From an algorithmic point of view, the main source of parallelism for meta-heuristics is the concurrent execution of their inner-loop iterations: evaluating neighbors, computing the fitness of individuals, or having ants forage concurrently. Unfortunately, this is often also the only source of readily available parallelism in meta-heuristics, most other steps being time dependent and requiring the computation of the previous steps to be completed. Even when parallelism is available, synchronization enforcing the time dependency of the meta-heuristic steps yields significant delays, which makes parallel computation nonrelevant.

A significant amount of parallelism may be found, on the other hand, in the domain of the problem addressed or in the corresponding search space. Indeed, there are no data dependencies between the cost or evaluation functions of different solutions and, thus, these may be computed in parallel. Furthermore, theoretically, the parallelism in the solution or search space is as large as the space itself. There are considerable limitations to an efficient exploitation of this parallelism, however. For obvious reasons, one cannot assign a processor to each solution evaluation. The solution or search space must therefore be partitioned among processors, thus serializing the evaluation of solutions assigned to the same processor. The resulting partitions are generally still too large for explicit enumeration and, thus, an exact or heuristic search method is still required for implicitly exploring it. Partitioning then raises two issues with respect to an overall meta-heuristic search strategy. First, the control of an overall search conducted separately on several partitions of the original space and the comprehensiveness of the solution finally reached. Second, the allocation of the computing resources for an efficient exploration avoiding, for example, searching regions with poor-quality solutions. Nonetheless, besides the inner-loop computations, this is the only other relevant source of parallelism for meta-heuristics and is exploited in many of the strategies described in this chapter.

Two main approaches are used to partition the search space: *domain decomposition* and *multi-search* (the name *multiple walks* is also found in the literature). The former explicitly partitions it (see Section 17.4), while the latter implicitly divides it through concurrent explorations by several methods, denoted in the following “search threads.” Using different search strategies contributes toward a non-overlapping exploration of the search space, but does not guarantee it and, thus, a multi-search parallelization rarely provides a proper partition of the search space. Multi-search strategies, particularly those based on cooperation principles, make up the bulk of the successful parallel meta-heuristics, however. They are the object of most recent publications in the field and are addressed in Sections 17.5 and 17.6.

### 17.2.3 Parallel Meta-heuristics Strategies

We adopt the classification of Crainic and Nourredine [36], generalizing that of Crainic, Toulouse, and Gendreau [41] (see also [26, 37, 38]; [150] and [42] present classifications that proceed of the same spirit), to describe the different parallel

strategies for meta-heuristics. This classification reflects the previous discussion and is sufficiently general to encompass all meta-heuristic classes, while avoiding a level of detail incompatible with the scope and dimension limits of the chapter.

The three dimensions of the classification indicate how the global problem-solving process is controlled, how information is exchanged among processes, and the variety of solution methods involved in the search for solutions, respectively. The first dimension, *Search Control Cardinality*, thus specifies whether the global search is controlled by a single process or by several processes that may collaborate or not. The two alternatives are identified as *1-control* (1C) and *p-control* (pC), respectively.

The second dimension, relative to the type of *Search Control and Communications*, addresses the issue of information exchanges. In parallel computing, one generally refers to *synchronous* and *asynchronous* communications. In the former case, all concerned processes stop and engage in some form of communication and information exchange at moments (number of iterations, time intervals, specified algorithmic stages, etc.) exogenously determined, either hard-coded or determined by a control (master) process. In the latter case, each process is in charge of its own search, as well as of establishing communications with other processes, and the global search terminates once each individual search stops. To reflect more adequately the quantity and quality of the information exchanged and shared, as well as the additional knowledge derived from these exchanges (if any), we refine these notions and define four classes: *Rigid (RS)* and *Knowledge Synchronization (KS)* and, symmetrically, *Collegial (C)*, and *Knowledge Collegial (KC)*.

Because more than one solution method or variant (e.g., different parameter settings) may be involved in a parallel meta-heuristic, the third dimension indicates the *Search Differentiation*: do search threads start from the same or different solutions and do they make use of the same or different search strategies? The four cases considered are *SPSS, Same initial Point/Population, Same search Strategy*; *SPDS, Same initial Point/Population, Different search Strategies*; *MPSS, Multiple initial Points/Populations, Same search Strategies*; *MPDS, Multiple initial Points/Populations, Different search Strategies*. Obviously, one uses “point” for neighborhood-based methods, while “population” is used for genetic-based evolutionary methods, scatter search, and ant-colony methods.

Based on this classification and the sources of parallelism in meta-heuristics identified at Section 17.2.2, we address the parallel meta-heuristic strategies in four groups and sections: 1-control strategies exploiting the intrinsic parallelism offered by the basic, inner-loop, computations of meta-heuristics in Section 17.3, and strategies based on explicit domain decomposition in Section 17.4, while Sections 17.5 and 17.6 are dedicated to independent and cooperative multi-search strategies, respectively.

We complete this section with a few notes on measures to evaluate the performances of parallel meta-heuristics. The traditional goal when designing parallel solution methods is to reduce the time required to “solve,” exactly or heuristically, given problem instances or to address larger instances without increasing

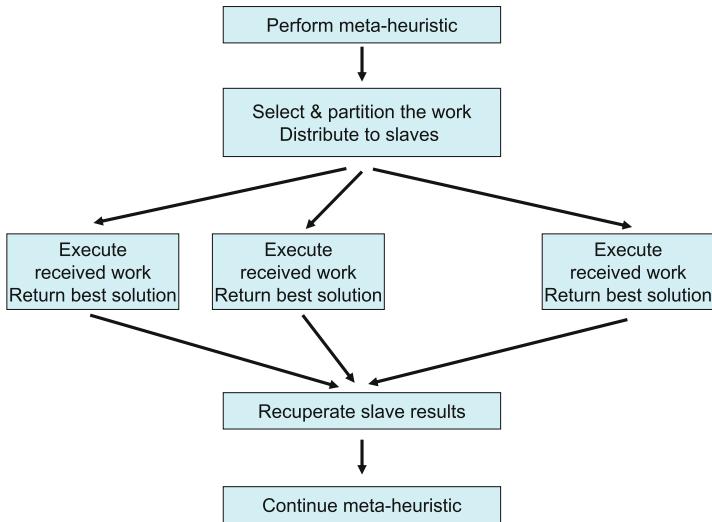
the computational effort. For exact solution methods that run until the optimal solution is obtained, this translates into the well-known *speedup* performance measure, computed as the ratio between the wall-clock time required to solve the problem instance in parallel with  $p$  processors and the corresponding solution time of the best-known sequential algorithm. A somewhat less restrictive measure replaces the latter with the time of the parallel algorithm run on a single processor. See [8] for a detailed discussion of this issue, including additional performance measures.

Speedup measures are more difficult to define when the optimal solution is not guaranteed or the exact method is stopped before optimality is reached. Indeed, for most parallelization strategies, the sequential and parallel versions of a heuristic yield solutions that are different in value, composition, or both. Thus, an equally important objective when parallel heuristics are contemplated is to design methods that outperform their sequential counterparts in terms of solution quality and, ideally, computational efficiency, i.e., the parallel method should not require a higher overall computation effort than the sequential method or should justify the effort by higher quality solutions. Search robustness is another characteristic increasingly expected of parallel heuristics. Robustness with respect to a problem variant is meant here in the sense of providing “equally” good solutions to a large and varied set of problem instances, without excessive calibration, neither during initial development nor when addressing new problem instances. See [37, 38] for a discussion of these issues.

## 17.3 Low-Level 1-Control Parallelization Strategies

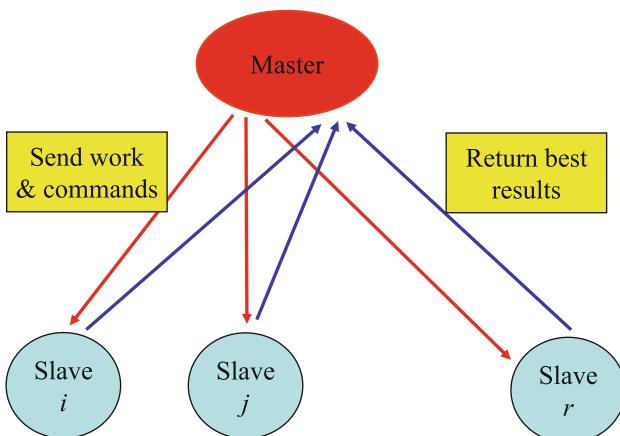
Parallel strategies that exploit the potential for task decomposition within the inner-loop computations of meta-heuristics are often labeled “low level” because they modify neither the algorithmic logic nor the search space. They aim solely to accelerate the search and generally do not modify the search behavior of the sequential meta-heuristic. Typically, the exploration is initialized from a single initial solution or population, and the search proceeds according to a single meta-heuristic strategy, only the inner-loop computations being decomposed and simultaneously performed by several processors.

Most low-level parallel strategies belong to the 1C/RS/SPSS class and are usually implemented according to the classical master-slave parallel programming model. A “master” program executes the 1-control sequential meta-heuristic but dispatches computation-intensive tasks to be executed in parallel by “slave” programs, as illustrated in Figure 17.5. The master program receives and processes the information resulting from the slave operations, selects and implements moves or, for population-based methods, selects parents and generates children, updates the memories (if any) or the population, and decides whether to activate different search strategies or stop the search. The *slave* programs perform evaluations



**Fig. 17.5** Low-level decomposition strategy.

and return the results to the master which, once all the results are in, resumes the normal logic of the sequential meta-heuristic. The complete control on the algorithm execution rests with the master, which decides the work allocation for all other processors and initiates most communications. No communications take place among slave programs. Figure 17.6 illustrates this control and communication scheme. Its instantiations for neighborhood and population-based methods are presented in Sections 17.3.1 and 17.3.2.



**Fig. 17.6** Master-slave configuration.

### 17.3.1 Neighborhood-Based 1C/RS/SPSS Meta-heuristics

This is the parallelization of the neighborhood evaluation procedure, depicted in Figure 17.2, called to compute the next current solution of a local search heuristic embedded in a neighborhood or population-based (e.g., evolutionary procedures implementing advanced “schooling” for offspring) meta-heuristic. Each iteration of the appropriate **while** loop of the procedure generates and evaluates one neighbor of the current solution and may be executed independently of the other iterations since no data dependency exists between iterations. The computations of these iterations may then be distributed over the available  $p$  processors as illustrated in Figure 17.7. The master groups the neighbors into the appropriate number of tasks, which are then sent to slaves. Each slave then executes the *Neighbor* procedure on its respective part of the neighborhood and sends back the best one found. The master waits for all slaves to terminate their computations and, then, selects the best move and proceeds with the search. See [58] for an application of this strategy to a tabu search meta-heuristic for the vehicle routing problem with time-window constraints (VRPTW).

---

**input:**  $x$  a current solution;  $task\_size$  the number of neighbors in a task

```

 $c(Best) \leftarrow \infty$ ;  $number\_of\_tasks \leftarrow \frac{|\mathcal{N}(x)|}{task\_size}$ ;  $j \leftarrow i \leftarrow 0$ 
while ( $j < number\_of\_tasks$ )
    send_to_a_slave_process( $task_j$ );  $j++$ 
while ( $i < number\_of\_tasks$ )
     $y \leftarrow receive\_from\_slave(neighbor)$ ;  $i++$ 
    if ( $c(y) < c(Best)$ ) then  $Best \leftarrow y$ 
return  $Best$ 
```

---

**Fig. 17.7** 1C/RS strategy: master neighborhood evaluation.

There is no predefined optimal size for the parallel tasks, the granularity  $|\mathcal{N}(x)|/p$  of the decomposition depending upon the number  $p$  of available processors, as well as preoccupations with inter-processor communication times and balancing work loads among processors, given the computer architecture on which computations are being performed. Thus, for example, defining each neighbor evaluation as a single task and dynamically dispatching these on a first-available, first-served basis to slave processors as they complete their tasks provide maximum flexibility and good load balancing when the evaluation of neighbors is of uneven length. For most parallel computer architectures, however, this fine-grained parallelism may come at too high an overhead cost for creating and exchanging tasks. When neighbor evaluations are sensibly the same, an often used strategy is to partition the elements defining the neighborhood into as many groups as available processors.

Notice that, when the local search procedure returns the first-best neighbor (e.g., the simple-serializable-set approach for parallel simulated annealing [74, 82]), the implemented move will often be different from that of the sequential version and,

thus, the two algorithms will behave differently. Moreover, the speedup performance of this strategy will be poor when many “good” neighbors are readily available (e.g., when the temperature parameter of simulated annealing is high and most neighbors are acceptable).

### 17.3.2 Population-Based 1C/RS/SPSS Meta-heuristics

1C/RS/SPSS parallelism in genetic algorithms is to be found in the loops that implement the selection, crossover, mutation, and fitness-evaluation operators, the resulting methods being variably identified in the literature as *global parallelization*, *master-slave parallelization*, and *distributed fitness evaluation*.

In theory, the degree of parallelism for each of these four operators is equal to the population size, but overhead costs may significantly decrease the degree of achievable parallelism. Actually, to be worth parallelizing, the computation must be significant. It is of little worth, for example, to parallelize a computationally simple operation like mutation. In other cases, an efficient parallelization can only be implemented on shared memory systems. Thus, for example, the selection operator must be able to access randomly any individual in the population and its parallelization on a distributed-memory computer, where individuals are distributed across several processors, is too costly and inefficient. In practice, only the fitness evaluation can satisfy these requirements and is often the only practical source of 1C/RS/SPSS parallelism for genetic–evolutionary methods.

The 1C/RS/SPSS parallel fitness evaluation of a population can be implemented using the master-slave model. The master partitions the individuals among slaves, which compute and return the fitness of each individual, as well as aggregate figures to facilitate the average population fitness to be computed by the master once all slaves have reported in. Similarly to other 1-control low-level parallelizations, the execution of a 1C/RS/SPSS genetic–evolutionary algorithm performs the same search as the sequential program, only faster.

The 1C/RS/SPSS parallelism for ant-colony methods lies at the level of the individual ants. Ants share information indirectly through the pheromone matrix. Furthermore, the pheromone matrix is updated once all solutions have been constructed, and there are no modifications of the pheromone matrix during a construction cycle. Consequently, the construction procedure performed by each individual ant is performed without data dependencies on the progress of the other ants.

Currently, most parallel ant-colony methods implement some form of 1C/RS/SPSS strategy according to the master-slave model, including [14, 49, 118, 120, 141]. The master builds tasks consisting of one or several ants (which can be assimilated to a “small” colony) and distributes them to the available processors. Slaves perform their construction heuristic and return their solution(s) to the master, which updates the pheromone matrix, returns it to the slaves, and so on. To further speed up computation, the pheromone update can be computed at the level of each slave, which computes the update associated with its solutions as well as the

best solution and sends the aggregated pheromone update and its best solution to the master. The fine-grained version with central matrix update has been the topic of most contributions so far and, in general, it outperformed the sequential version of the algorithm. It is acknowledged, however, that it does not scale well and, similarly to other meta-heuristics, this strategy is outperformed by more advanced multi-search methods.

Scatter search and path relinking implement different evolution strategies, where a restricted number of elite solutions are combined, the result being enhanced through a local search or a full-fledged meta-heuristic, usually neighborhood-based. Consequently, the 1C/RS/SPSS strategies discussed previously regarding the parallelization of local-search exploration apply straightforwardly to the present context, as in [59, 60, 62] for the  $p$ -median and the feature-selection problems.

A different 1C/RS/SPSS strategy for scatter search may be obtained by running concurrently the combination and improvement operators on several subsets of the reference set. Here, the master generates tasks by extracting a number of solution subsets and sending them to slaves. Each slave then combines and improves its solutions, returning its results to the master for the update of the reference set. Each subset sent to a slave may contain the exact number of solutions required by the combination operator or a higher number. In the former case [59, 60, 62], the corresponding slave performs an “iteration” of the scatter search algorithm. In the latter case, several combination-improvement sequences could be executed and solutions could be returned to the master as they are found or all together at the end of all sequences. This heavy load for slaves may conduct to very different computation times and, thus, load-balancing capabilities should be added to the master.

### 17.3.3 Remarks

We complete the low-level parallelization discussion with two remarks.

A second class of low-level parallelization approaches was defined in the literature, the so-called probing or look-ahead strategies, parallelizing the *sequential fan candidate list* strategy first proposed for tabu search [72, 73]. Probing strategies belong to the 1C/KS class with any of the search-differentiation models identified previously. For neighborhood-based methods, probing may allow slaves to perform a number of iterations before synchronization and the selection of the best neighbor solution from which to proceed (one may move directly to the last solution identified by the slave or not). For population-based methods, the method may allow each slave to generate child solutions, “educate” them through a hill climbing or local-search procedure, and play out a tournament to decide who of the parents and children survive and are passed back to the master. To the best of our knowledge, [39] is the only paper to ever report results on a parallel implementation of the sequential fan candidate list strategy. The authors realized a comparative study of several synchronous tabu search parallelizations for the location-allocation problem with balancing requirements, including a straightforward 1C/RS/SPSS approach and a

1C/KS/SPSS method following the model just described (as well as a few p-control approaches). Both the 1C/KS/SPSS and the 1C/RS/SPSS heuristics yielded better solutions than sequential tabu search on the tested instances, the former being consistently superior to the latter.

We notice that a rather limited impact of low-level, 1-control parallel strategies was observed in most cases. Of course, when neighborhoods are large or neighbor-evaluation procedures are costly, the corresponding gain in computing time may prove interesting, e.g., the parallel tabu searches of [19, 21, 135] for the Quadratic Assignment Problem (QAP), [20] for the Traveling Salesman Problem (TSP), and [115–117] for the task-scheduling problem. Then, when a sufficiently large number of processors is available, it might prove worthy to combine a 1C/RS/SPSS approach with more sophisticated strategies into hierarchical solution schemes (e.g., [122] were low-level parallelism accelerated the move evaluations of the individual searches engaged into an independent multi-search procedure for the VRP). More advanced multi-search strategies generally outperform low-level strategies, however.

## 17.4 Domain Decomposition

Domain or *search-space decomposition* constitutes another major parallelization strategy, one that is intuitively simple and appealing: divide the search space into smaller, usually disjoint but not necessarily exhaustive sets, solve the resulting subproblems by applying the sequential meta-heuristic on each set, collect the respective partial solutions, and reconstruct an entire one.

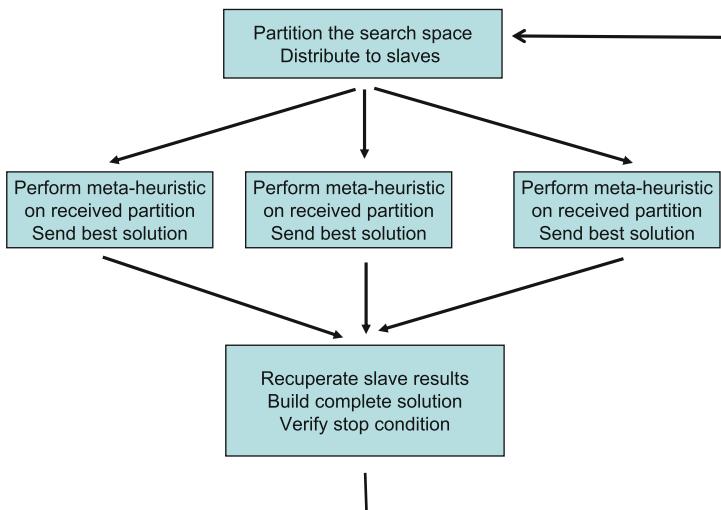
This apparently simple idea may take several forms, however. The most straightforward approach consists in partitioning the solution vector, each resulting subset defining a subproblem. Thus, for example, the arc-design variables of a VRP may be separated into customer subsets (including the depot in each subset). A number of factors must then be specified to completely define the search-space decomposition. First, whether the partition is strict or subsets are allowed to overlap (e.g., “close by” customers may appear in two subsets in the previous VRP example). Second, whether search threads consider complete or partial solutions to the problem (in both cases, search processes access only a restricted portion of the search space). In the latter case, a complete solution has to be reconstructed at some point. Third, whether the “moves” performed on a subproblem are restricted to the corresponding search-space subset or may involve variables in neighboring subspaces creating an indirect overlapping of subsets.

Strict partitioning restricts the meta-heuristic threads to their subsets and forbids moves involving solutions belonging to two or more subsets (e.g., arc swaps involving customers in different subsets). This obviously results in part of the search space being unreachable and the parallel meta-heuristic being non-optimal. Explicit or implicit overlapping aims to address this issue. But not completely and not without cost. Thus, the only way to guarantee that all potential solutions are reachable

is to make overlapping cover the entire search space. This corresponds to “no decomposition” in the case of explicit overlapping and is thus not relevant. For implicit overlapping, it may also deny any gain resulting from decomposition in the first place or, in the best case, require significant overhead costs to keep most of the subproblem threads within their own subspaces.

Consequently, strict partitioning or very limited overlapping are the preferred approaches and a re-decomposition feature is included to increase the thoroughness of the search and allow all potential solutions to be examined: the decomposition is modified at regular intervals and the search is restarted using this new decomposition. This feature provides also the opportunity to define non-exhaustive decompositions, i.e., where the union of the subsets is smaller than the complete search space. A complete-solution reconstruction feature is almost always part of the procedure.

This strategy is naturally implemented using 1C/KS schemes, with a MPSS or MPDS search-differentiation strategy, according to the master-slave programming model illustrated in Figure 17.8. The master process determines the partition and sends subsets to slaves, synchronizes their work and collects their solutions, reconstructs solutions (if required), modifies the partitions, and determines stopping conditions. Slaves concurrently and independently perform the search on their assigned search-space subsets. The master is illustrated in Figure 17.9, where it is assumed that each slave performs a local search or a meta-heuristic on its subproblem and returns its current best solution when the master synchronizes activities.



**Fig. 17.8** Domain decomposition – master-slave logic.

For neighborhood-based meta-heuristics, as well as for evolutionary methods embedding such meta-heuristics or local search procedures, one may implement the search-space decomposition approach by replacing the corresponding local search with the master-slave strategy of Figure 17.9. A few modifications to the original

---

**input:**  $x \in \mathcal{X}$  an initial solution;  $j \leftarrow i \leftarrow 0$   
 Decompose problem instance into  $p$  subproblems  $S_1, S_2, \dots, S_p$   
**while** (Stopping criteria not reached)  
  **while** ( $j < p$ ) send\_to\_a\_slave\_process( $S_j, x$ );  $j++$   
  Synchronize slaves  
  **while** ( $i < p$ )  $x_i \leftarrow$  receive\_from\_slave();  $i++$   
   $x \leftarrow$  construct( $x_1, x_2, \dots, x_p$ )  
   $S_1, S_2, \dots, S_p \leftarrow$  Modify partition  
**return**  $x$

---

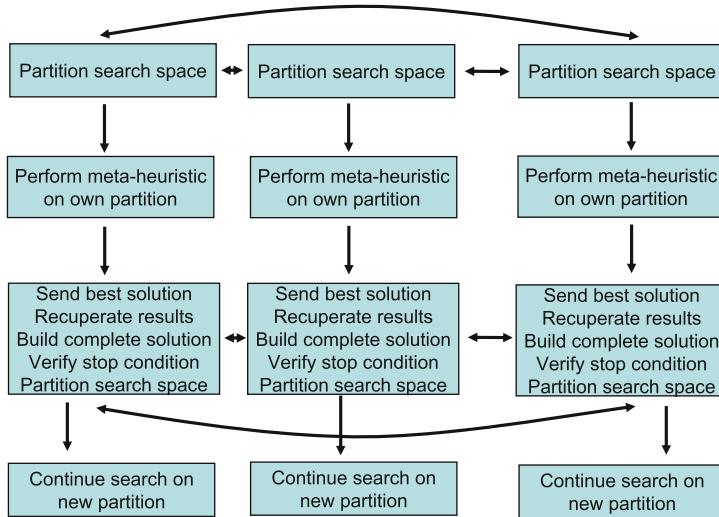
**Fig. 17.9** Domain decomposition – master procedure.

meta-heuristic may be required though. Thus, for simulated annealing, the length of the local search (cooling schedule) of each slave is reduced by a factor equal to  $p$ . The main issue for tabu search is the global tabu list the master has to reconstruct out of the local memories of the slaves simultaneously with the reconstruction of a complete solution prior to continuing the search.

A different approach for implementing the 1C/KS scheme of Figures 17.8 and 17.9 is to execute a full meta-heuristic on each subset of the search space, periodically modifying the partition and re-starting the search. Such an approach has been used for tabu search and proved quite successful for problems for which a large number of iterations can be performed in a relatively short time and restarting the method with a new decomposition does not require an unreasonable computational effort (e.g., [54] for the TSP, [83] for image filtering, and [68] to solve efficiently in real-time several variants of the same ambulance fleet management problem instance).

We are not aware of any application of the previous approach to meta-heuristics other than tabu search, although such applications would be straightforward in most cases. A similar remark applies to the pC/KS version of search-space decomposition, a MPSS or MPDS search-differentiation strategy, illustrated in Figure 17.10. Such an approach was proposed in [136] for the VRP, where the customer set was partitioned, vehicles were allocated to the resulting regions, and each subproblem was solved by an independent tabu search. All processors stopped after a number of iterations that varied according to the total number of iterations already performed. The partition was then modified by an information exchange phase, during which tours, undelivered cities, and empty vehicles were exchanged between adjacent processors (corresponding to neighboring regions). At the time, this approach did allow to address successfully a number of problem instances, but the synchronization inherent in the design of the strategy hindered its performance. A parallel ant-colony approach combining this decomposition idea with a master-slave implementation was presented in [51] (parallelizing the algorithm presented in [123]), where the master generates an initial solution, defines the partition, and updates the global pheromone matrix, while slaves execute a savings-based ant colony algorithm [124] for the resulting restricted VRP.

To complete the presentation, notice that the search behavior and the computational work performed by the sequential and parallel versions of meta-heuristics,



**Fig. 17.10** Domain decomposition – collegial logic.

as well as the quality of their respective solutions are not the same in most cases (again, enforcing similar behaviors would require efforts that would take away any possible benefit from parallelization). Search-space decomposition methods appear increasingly needed as the dimensions of contemplated problem instances continue to grow. Clearly, more work is required on how to best combine domain decomposition and the other parallelization strategies, cooperation in particular.

## 17.5 Independent Multi-search

*Independent multi-search* is among the earliest parallelization strategies. It is also the most simple and straightforward p-control parallelization strategy and generally offers very interesting performances.

The strategy consists in performing several searches simultaneously on the entire search space, starting from the same or from different initial solutions, and selecting at the end the best among the best solutions obtained by all searches. It is thus a straightforward parallelization of the well-known multi-start heuristic.

Independent multi-search methods belong to the pC/RS class of the taxonomy. No attempt is made to take advantage of the multiple search threads running in parallel other than to identify the best overall solution once all programs stop. This earns independent search strategies their rigid synchronization classification.

Independent multi-search methods turn out to be effective, simply because of the sheer quantity of computing power they allow one to apply to a given problem. This was established empirically by several papers, including the tabu searches in [10] for the QAP and [137] for the job shop scheduling problems, in which excellent

results were obtained when compared to the best existing heuristics at the time. Both studies also attempted to establish some theoretical justifications for the efficiency of independent search. Battiti and Tecchiolli [10] derived models that showed that the probability of “success” increased and the corresponding average time to “success” decreased with the number of processors (provided the tabu procedure did not cycle). On the other hand, Taillard [137] showed that the conditions required for the parallel method to be “better” than the sequential one are rather strong, where “better” was defined as “the probability the parallel algorithm achieves success by time  $t$  with respect to some condition (in terms of optimality or near-optimality), is higher than the corresponding probability of the sequential algorithm by time  $pt$ .” However, the author also mentioned that, in many cases, the empirical probability function of iterative algorithms was not very different from an exponential one, implying that independent multi-thread parallelization is an efficient strategy. The results for the job shop problem seemed to justify this claim. Similar results may also be found in [142].

This combination of simplicity of implementation and relatively good performances explains the popularity of the pC/RS/MPSS strategy for the parallelization of neighborhood-based meta-heuristics, e.g., tabu search for the VRP [122, 140] and production planning [12]; GRASP for the QAP [92, 112, 114], the Steiner problem [95, 96], and the 2-path telecommunication network design [125–127]; simulated annealing for graph partitioning [6, 7, 43, 91] (in the first two contributions the simulated annealing threads were enhanced with a simple tabu search to avoid cycling and were part of a multi-level implementation) and the TSP [99] (where each search thread was a simulated annealing procedure with an adaptive temperature schedule controlled by a genetic algorithm); and variable neighborhood search for the  $p$ -median problem [61].

Independent multi-search pC/RS/MPSS applications to non-genetic evolutionary methods have also been proposed for scatter search [60, 62], as well as for ant-colony optimization for set covering [118], the TSP [134], and the VRP [50]. Stützle [134] also presented a mathematical analysis and empirical results that suggest the behavior of pC/RS ant-colony parallel algorithms is similar to that of neighborhood-based meta-heuristics, i.e., assuming that the probability of finding the best solution is exponentially distributed with respect to time, independent multi-colony strategies are likely to find better solutions than a sequential implementation for properly selected performance targets.

In theory, pC/RS independent multi-search may be as easily adapted to parallel evolutionary meta-heuristics by running the same (MPSS) or different (MPDS) evolutionary method on disjoint populations. In practice, however, most evolutionary-genetic pC/RS parallelizations used small-sized populations, an “initial” population of size  $n$  being separated into  $n/p$  groups for the  $p$  available processors [77, 130], a strategy that did not perform well when compared to sequential algorithms with optimized population size for the particular problem instance. Indeed, while small populations speed up the computation of genetic operators such as fitness evaluation and crossover, they also display well-documented adverse impacts on the diversity of the genetic material, leading to premature convergence of the search. Running

several full-sized population genetic methods in parallel [23, 24] avoids this issue and offers performances similar to those observed for the other meta-heuristics: a computation effort multiplied by the number of independent search threads and generally being outperformed by cooperative strategies.

## 17.6 Cooperative Search Strategies

Independent multi-search strategies seek to accelerate the exploration of the search space toward a better solution (compared to sequential search) by initiating simultaneous search threads from different initial points (with or without different search strategies). Cooperative search strategies go one step further and integrate mechanisms to share, *while the search is in progress*, the information obtained from this diversified exploration. The sharing and, eventually, creation of new information yields in many cases a collective output with better solutions than a parallel independent search.

Cooperative multi-search methods launch several independent *search threads*, each defining a trajectory in the search space from a possibly different initial point or population by using a possibly different meta-heuristic or search strategy. The information-sharing cooperation mechanism specifies how these independent meta-heuristics interact the global search behavior of the cooperative parallel meta-heuristic emerging from the local interactions among them. Such similarities with systems where decisions emerge from interactions among autonomous and equal “colleagues” have inspired the name *collegial control* for the classes of strategies described in this section.

Cooperative search may be viewed as a bottom-up meta-heuristic specifying the components and their interactions, and it may thus become a “new” meta-heuristic in its own right. The key challenge of cooperation is to ensure that *meaningful* information is exchanged in a *timely* manner yielding a global parallel search that achieves a better performance than the simple concatenation of the results of the individual threads, where performance is measured in terms of computing time and solution quality. Toulouse, Crainic, and Gendreau [144] have proposed a list of fundamental issues to be addressed when designing cooperative parallel strategies for meta-heuristics: What information is exchanged? Between what processes is it exchanged? When is information exchanged? How is it exchanged? How is the imported data used? Implicit in their taxonomy and explicitly stated in later papers, the issue of whether the information is modified during exchanges or whether new information is created completes this list.

These decisions are more than implementation details, they constitute the core design parameters of a cooperative meta-heuristic. For example, a cooperative strategy could have a set of independent meta-heuristics re-start periodically from the current-best overall solution. The specification that all independent search threads are *re-started periodically* from the current-best solution of all the independent programs makes up the cooperation mechanism. It tells when programs

interact (periodically; the period length is usually clearly stated), what information is exchanged (the best solutions and the overall best), between what search threads (all), what to do with the exchanged information (re-start from the imported solution).

The information to be shared among cooperating search threads should aim to improve the performance of the receiving programs and create a global, “complete” image of the status of the search. “Good” solutions are the most often exchanged type of information. In many cases, this takes the form of the current-best solution a search thread sends to the others or, as in the previous example, the overall best being sent to all. Not all such strategies are profitable, however.

It has been observed that sending out all current-best solutions is often counter productive, particularly when the meta-heuristic starts on a series of improving moves or generations, as solutions are generally “similar” (particularly for neighborhood-based procedures) and the receiving threads have no chance to actually act on the in-coming information. It has also been observed that always sending the overall best solution to all cooperating threads is generally bad as it rapidly decreases the diversity of the parts of the search space explored and, thus, increases the amount of worthless computational work (many threads will search in the same region) and brings an early “convergence” to a not-so-good solution. Sending out local optima only, exchanging groups of solutions, and implementing random selection procedures for the solutions to send out, the latter generally biased toward good or good-and-different solutions, are among the strategies aimed at addressing these issues. (A different strategy was proposed in [3], where the negative impact of best solution broadcasts followed by search re-initialization was countered by having half the tabu searches regularly apply a diversification procedure, while the other half engaged in an intensification phase.)

So-called context information may also be exchanged. Context information refers to data collected by a meta-heuristic during its own exploration, such as the statistical information relative to the presence of particular solution elements in improving solutions (e.g., the medium and long-term memories of tabu search). Such exchanges show great promise as part of guiding heuristics for the overall search (see Section 17.6.3), but are not much used yet and significant research is needed to define and qualify them.

Search threads may exchange information directly or indirectly. Direct exchanges of information between two or more threads often occur when the concerned programs agree on a meeting point in time to share information. But not always. Thus, a search thread may send – broadcast – its information to one or several other threads without prior mutual agreement. Receiving search threads must then include capabilities to store such information without disturbing their own search trajectories until ready to consider it. Failure to implement such mechanisms may result in bad performances, as has been observed for strategies combining uncontrolled broadcasting of information and immediate acceptance of received data.

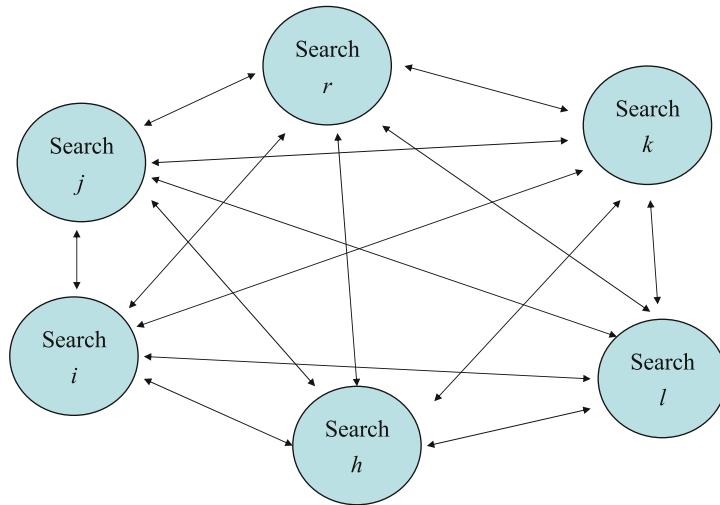
Indirect exchanges of information are performed through independent data structures that become “central” sources of information search threads may access asynchronously to post information and read information already posted. Such a

data structure is denoted *blackboard* in computer science and artificial-intelligence vocabulary; *memory*, *pool*, and *data warehouse* are equivalent terms found in the parallel meta-heuristic literature (due to the role assigned to the elements it contains, the terms *reference* and *elite set* are also sometimes used; in the following, we use “blackboard” for general discussions and an appropriate one among the others when addressing specific topics). Blackboards could be centralized or distributed. Centralized blackboards have been used in most parallel meta-heuristic contributions. They post information generated by all the search threads, which, in turn, may read this information independently. The distributed approach has several blackboards located on the sites of the search threads, which thus become *hosts*, and only a subset of threads may post and access information stored on a given local blackboard. Note that a blackboard which only posts information generated by its host can support direct asynchronous interactions between the host and a subset of “adjacent” search threads. The number of blackboards in such a distributed implementation could thus be as large as the number of search threads. More complex, hierarchical structures may be contemplated, in particular for grids or loosely coupled distributed systems, but have yet to be studied.

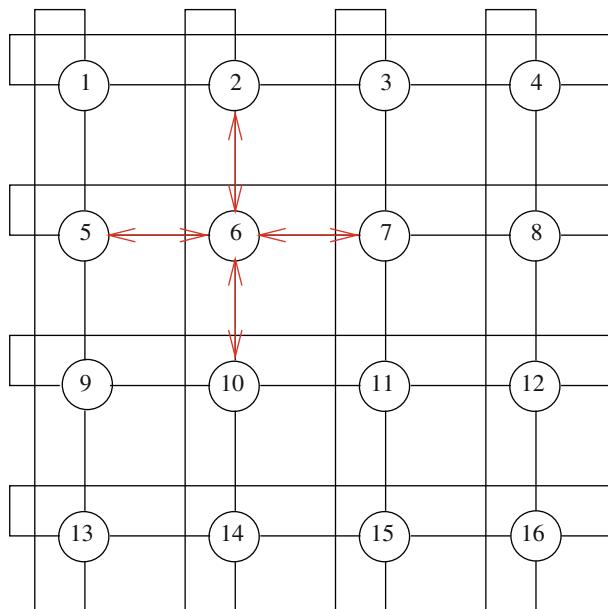
Communications proceed according to an *interaction topology* represented by a *communication graph* specifying the processes that may engage in direct exchanges. Each node of the graph represents a search thread or a blackboard. Edges define pairs of search threads or of a search thread and a blackboard that may communicate directly. They therefore specify the direct flow of information in the cooperative system. Communication graphs may mirror the physical interconnection topology of the parallel computer executing the parallel program. Often, however, the communication graph is logically defined to suit the requirements of the cooperation strategy. Typical interaction topologies found in the parallel meta-heuristic literature are complete graphs (Figure 17.11), rings, grids (Figure 17.12), toruses, and stars (Figure 17.13).

When and how information is shared specifies how frequently cooperation activities are initiated and whether, in order to engage in these activities, concerned search threads must synchronize, i.e., each stopping its activities and waiting for all others to be ready, or not. One identifies these two cases as *synchronous* and *asynchronous* communications, respectively. The accumulated knowledge of the field indicates for both cases that exchanges should not be too frequent to avoid excessive communication overheads as well as premature “convergence” to local optima [145–148].

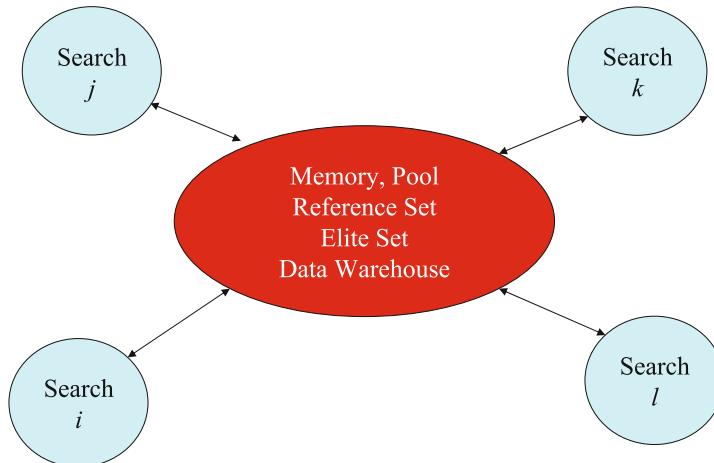
Synchronous cooperation activities are initiated based on conditions external to all, or to all but one of the individual programs. In the example above, where programs interact periodically, the cooperative search strategy is synchronous, exchanges being initiated when exogenously specified conditions, e.g., time or the number of iterations, are reached. These conditions are applied in the same way to all search threads, and communications cannot start until all have reached the designated status. The goal of synchronous cooperative strategies is to re-create a state of complete knowledge at particular points in the global search and, thus, to hopefully guide the global search into a coordinated evolution toward the desired solution to



**Fig. 17.11** Complete-graph communication scheme.



**Fig. 17.12** Grid communication scheme – diffusion-based cooperation.



**Fig. 17.13** Star communication scheme – memory-based cooperation.

the problem. As we will see in the following sections, this goal is rarely attained. Moreover, synchronization results in significant time inefficiencies as communications are initiated only when the slowest search thread is ready to start. We refer to such strategies as p-control, knowledge synchronous, pC/KS, with any of the SPDS, MPSS, or MPDS search differentiation approaches (appropriately applied).

A cooperation strategy is asynchronous when programs initiate cooperation activities according to their own internal state only, without coordination with other programs. Thus, for example, a search thread may make available its current best solution by posting it on a blackboard or may ask for an external solution after it failed to improve the quality on its best solution for a certain number of iterations. Asynchronous communications provide the means to build cooperation and information sharing among search threads without incurring the overheads associated with synchronization. As we will see in Sections 17.6.2 and 17.6.3, they also bring adaptivity to cooperation strategies, to the extend that the parallel cooperative meta-heuristic may react and dynamically adapt to the exploration of the search space of the particular problem instance being addressed. This is more likely to yield a globally emergent exploration strategy of the search space than synchronous approaches.

Asynchronous cooperation is fully distributed, cooperative activities being instantiated independently and concurrently by search threads, and are referred to as p-control collegial, pC/C, strategies (Section 17.6.2). Shared information may not just be exchanged; however, it may also be modified or used to infer knowledge. Thus, for example, statistical information may be gathered regarding configurations of solution elements in the best exchanged solutions, or the solutions gathered in a particular blackboard may form an elite population yielding new individuals to be shared among cooperating search threads. We refer to such settings as p-control knowledge collegial, pC/KC, strategies (Section 17.6.3).

It is worth noticing that cooperation is somewhat biased toward intensifying the search in regions of the solution space that have already been explored and where interesting solutions have been identified. This is particularly true for simple cooperation mechanisms based on synchronization or only exchanging current best solutions. It is thus important to equip the cooperation mechanisms with diversification capabilities. The introduction of probabilistic selection of exchanged solutions constitutes an example of such a mechanism. Advanced pC/KC cooperation strategies go further through creation of new solutions and guidance information as described in the following subsections.

The main principles of cooperative p-control parallelization are the same for neighborhood- and population-based meta-heuristics, even though denominations and implementation approaches may differ. One thus finds, for example, *coarse- and fine-grained island* models for genetic-based evolutionary methods, the differentiation following from the cardinality of the population of each participating meta-heuristic, few (even down to 1 in some implementations) and many individuals, respectively. Similarly, *multi-colony* is the term generally used in the ant-colony meta-heuristic community. The presentation that follows identifies these differences without dedicating subsections to each meta-heuristic class.

### **17.6.1 pC/KS Synchronous Cooperative Strategies**

According to the cooperative pC/KS scheme, the independent cooperating meta-heuristics enter into an information exchange phase at pre-determined intervals, phase that must be completed before any program can restart its exploration from that synchronization point.

Many proposed pC/KS cooperative search meta-heuristics follow a strategy where all threads synchronize at each point using a complete-graph communication model (Figure 17.11) and use a master-slave implementation. In this setting, a master process, which may or not also include one of the participating meta-heuristics, initiates the other threads, stops all threads at synchronization points, gathers the sharable information, updates the global data, decides on the termination of the search and, either effectively terminates it or distributes the shared information (a good solution, generally, the overall best solution in many cases) and the continue-search signal to the other meta-heuristic threads.

The pC/KS implementation of VNS for the  $p$ -median problem proposed in [61] followed this idea, as well as the tabu search-based implementations proposed for the TSP [94], the VRP (using ejection chains) [121, 122], the QAP [46] and the task mapping problem [45], the last two contributions attempting to overcome the limitations of the master-slave setting by allowing processes, on terminating their local search phases, to synchronize and exchange best solutions with processes running on neighboring processors. A more sophisticated pC/KS approach was proposed in [108], where the master dynamically adjusted the search-strategy parameters of cooperating tabu searches according to the results each thread obtained so far.

Computational results reported for the 0-1 Multi-dimensional Knapsack Problem showed that this dynamic adjustment of search parameters was indeed beneficial.

The master-slave implementation model has been applied to evolutionary methods as well. In coarse-grained island implementations of cooperating genetic methods [43, 133], the master stops the cooperating meta-heuristics and initiates the *migration* operator to exchange among the independent populations the best or a small group of some of the best individuals in each. Applied to ant-colony systems [53] (also for a satisfiability problem), this strategy divided the colony into several sub-colonies, each assigned to a different processor. Each slave sent to the master its best solution once its ants finished searching. The master then updated the pheromone matrix and started a new search phase.

Alternatively, pC/KS cooperative schemes can be implemented by having each search thread empowered to initiate synchronization once it reaches a pre-determined status. It then broadcasts its sharable data, current best solution or group of solutions, followed by similar broadcasts performed by the other search threads. Once all information is shared, each thread performs its own import procedures on the received data and proceeds with its exploration of the search space until the next synchronization event. Such an approach was proposed for simulated annealing [48], where the search threads transmitted their best solutions every  $n$  steps and re-started the search after updating their respective best solutions. This cooperative method outperformed an independent multi-thread search approach, both obtaining better results than the sequential version in terms of solution quality.

Most synchronous coarse-grained island parallelizations of genetic-based evolutionary methods fall under this category, where migration operators are applied at regular intervals, e.g., [152] for satisfiability problems, where the best individual of each population migrated to replace the worst of the receiving population, [55] for multi-objective telecommunication network design with migration following each generation, and [22–24, 78, 93] for graph-partitioning, the latter implementing a hierarchical method, where the fitness computation was performed at the second level (through a master-slave implementation; the overhead due to the parallelization of the fitness became significant for larger numbers of processors). A similar strategy was proposed for the multi-ant-colony algorithms [97, 98]. Each colony has its own pheromone matrix and may (homogeneous) or may not (heterogeneous) use the same update rule. Colonies synchronize after a fixed number of iterations to exchange elite solutions that are used to update the pheromone matrix of the receiving colony.

Several of these studies [22–24, 93] compared different implementations of coarse-grained parallel genetic methods and contributed to show that synchronous pC/KS strategies outperform independent search approaches. They also showed the superiority of dynamically determined synchronization points, as well as that of asynchronous communications. Similar conclusions were obtained with parallel simulated annealing for the graph partitioning problem [88–91] and with tabu search with location problems with balancing requirements [39, 40].

The previous strategies are based on global exchanges of information, gathered at synchronization points during the computation and distributed to all search threads.

The interest of these strategies follows from the fact that they use the best knowledge available at the synchronization points to attempt to guide the exploration. Global information sharing has obvious drawbacks, however. When each independent program is guided by the same set of best solutions, the global search lacks diversity and, eventually, all threads will focus on the same regions of the search space. Designing globally efficient synchronous cooperative strategies has proved to be difficult in most cases.

Synchronized cooperation may alternatively be based on direct local exchanges of information, global information sharing taking place through *diffusion*. To design such a cooperative strategy, the complete communication graph used previously is replaced by a less densely connected communication topology such as ring, torus, or grid graphs. The topology restricts to a few neighbors the direct communications a search thread may engage in, as illustrated in Figure 17.12 for a grid communication graph where, for example, the search thread on node 6 can share information only with threads on nodes 2, 5, 7, and 10. Following synchronization, each thread continues its search based on information obtained from programs adjacent in the communication graph. Even though no direct global exchanges take place, information is still shared through diffusion. Thus, for example, assuming the meta-heuristic at node 6 received and accepted the best solution of node 2 at synchronization point  $i$  and the program at node 10 receives and accepts the solution of node 6 at synchronization point  $i + 1$ , information from search thread on node 2 passed to the method on node 10. Through diffusion, the search threads on nodes 2 and 10 shared some information even though they are not adjacent.

This idea has not been as broadly explored as the global-exchange strategy, even though synchronous cooperative mechanisms based on local exchanges and diffusion have a less negative impact on the diversity of the search-space exploration. A number of applications were proposed for coarse-grained [15, 143] and fine-grained [56, 57, 101, 103] genetic-based evolutionary methods with good results. (It is interesting to recall that [101, 103] was part of a larger body of contributions [102, 104–107] where hill-climbing heuristics were embedded into genetic algorithms to improve—“educate”—individuals and the impact of this hybridization on the behavior and performance of genetic methods was studied.) An application to ant-colony optimization methods was also proposed with similar results [98].

Cooperation based on asynchronous information sharing generally outperforms synchronous methods, however, and are the topic of Section 17.6.2.

## 17.6.2 *pC/C Asynchronous Cooperative Strategies*

Historically, independent and synchronous cooperative methods were the first multi-search approaches to be developed. However, because of the shortcomings of these methods, discussed at length in the previous subsections, attention has increasingly been turned toward asynchronous strategies, which now largely define the “state-of-the-art” in parallel multi-search meta-heuristics. These asynchronous procedures

all follow the same general pattern: starting from possibly different initial solutions and using possibly different search strategies,  $p$  threads explore simultaneously the search space, exchanging and, eventually, creating information according to a mechanism moved by the internal logic of each participating search thread and the state of the search. Asynchronous cooperative strategies belong either to the pC/C or to the pC/KC class of the taxonomy, the main difference between the two being whether or not any “new” knowledge is inferred on the basis of the information exchanged between the search threads; pC/KS strategies are addressed in Section 17.6.3.

Most genetic-based evolutionary asynchronous cooperative meta-heuristics belong to the pC/C class. They generally implement a coarse-grained island model, where migration is triggered within individual populations, selected migrant individuals being directed toward either all other populations or a dynamically selected subset. An early comparative study of coarse-grained parallel genetic methods for the graph-partitioning problem numerically showed the superiority of the pC/C strategy (with migration toward a subset of populations) over synchronous approaches [93]. Currently, this is the most popular strategy for multi-population parallel genetic methods [17].

The sharing of information in most asynchronous cooperative search strategies outside the genetic-evolutionary community is based on some form of centralized blackboard model, most often denoted *central memory* [26, 40, 41] and illustrated in Figure 17.13. According to this pC/C cooperation model, whenever a search thread identifies sharable information it sends it to the central memory, from where, when needed, it also retrieves information sent by the other cooperating search threads. (Information retrieval on distributed-memory systems passes through a request to the processor running the central-memory processes.) The sharable information corresponds to a locally improving solution, the most successful implementations sending new local optima only, according to the already mentioned information-sharing parsimony principle.

The need for cooperation is particular to each type of meta-heuristic involved in the cooperation. Following again the principle of parsimonious communications, such activities are often initiated at algorithmic steps involving a choice of solutions or a modification of the current solution from where the next local search will proceed, e.g., diversification moves in tabu search or neighborhood changes in variable neighborhood search. It may also be triggered by a priori rules, e.g., a fixed number of iterations, particularly when no such algorithmic steps exist, e.g., the tabu searches based on continuous diversification strategies. Central-memory-based asynchronous cooperative algorithms thus provide the environment of an indirect exchange of information between cooperating search threads, in particular among the one that stored the sharable information in the central memory and the one that accessed this information through a request for knowledge to the central memory.

Most current implementations of this approach manage the central-memory information following an algorithmic template similar to the one illustrated in Figure 17.14. The input variable *memory\_size* specifies the capacity of the central memory, i.e., the number of solutions that can be stored in the memory. Incoming solutions are automatically accepted when the memory is not full. Acceptance is

---

```

input: memory_size number of sharable information instances
 $c(rWorst) \leftarrow \infty$ ;  $fill \leftarrow 0$ 
while (not finished)
   $x', message, pid \leftarrow \text{receive\_from\_coop\_search\_thread}()$ 
  if (message = put_in_memory) then
    if ( $c(x') < c(rWorst)$ ) then
      if ( $fill \leq \text{memory\_size}$ ) then
         $\text{memory} \leftarrow \text{memory} \cup x'$ ;  $fill \leftarrow fill + 1$ 
      else
         $\text{memory} \leftarrow \text{memory} \cup x'$ 
         $\text{memory} \leftarrow \text{memory} \setminus rWorst$ 
         $rWorst \leftarrow \text{compute\_rWorst}()$ 
    if (message = get_from_memory) then
       $rBest \leftarrow \text{select\_randomly}(\text{memory})$ 
       $\text{send\_to\_search\_thread}(rBest, pid)$ 

```

---

**Fig. 17.14** Memory template.

conditional to the relative interest of the incoming solution compared to the “worst” solution in the memory, otherwise. In most cases, the comparison is based on the evaluation function for the corresponding search space or the value of the objective function of the original problem. The *rWorst* solution of a full memory is then replaced with the incoming solution when the value of the latter is better (lower, for minimization problems) than that of the former. Diversity measures may modify this choice in more advanced strategies (which may also delete a larger part, half, usually, of the population in memory). The template also illustrates the random solution-extraction process, which follows the request of a cooperating search thread for a solution from memory. The random selection may be uniform or biased to favor solutions with the best ranking based on, for example, solution values.

Several central-memory-based cooperative search strategies are described in the literature, including simulated annealing applications to graph partitioning [88–91] and the TSP [129], and the master-slave implementation of a pC/C/MPSS cooperation mechanism for VNS applied to the *p*-median problem [33]. In the latter work, individual VNS processes communicated exclusively with a master process, which kept, updated, and communicated the current overall best solution (it also initiated and terminated the algorithm). Solution updates and communications were performed following messages from the individual VNS threads, which proceeded with the “normal” VNS exploration for as long as the solution was improved. When the solution was not improved, it was communicated to the master (if better than the one at the last communication) and the overall best solution was requested from the master. The search was then continued starting from the best overall solution in the current neighborhood. Computational results on TSPLIB problem instances with up to 11849 customers showed that the cooperative strategy yielded significant gains in terms of computation time without loosing on solution quality, which was comparable to that of the best results in the literature (when available).

To the best of our knowledge, Crainic, Toulouse, and Gendreau were the first to propose a central-memory approach for asynchronous tabu search in their

comparative study for multi-commodity location with balancing requirements [40]. Their method, where individual tabu searches sent to the memory their local best solutions when improved and imported a probabilistically selected (rank-biased) solution from the memory before engaging in a diversification phase, outperformed in terms of solution quality the sequential version (which was also bested in terms of wall-clock computing time) as well as pC/RS/MPDS, pC/KS (varying the synchronization mechanisms and the Search-Differentiation strategies), and broadcast-based asynchronous pC/C cooperative strategies. The same approach was applied to the fixed cost, capacitated, multicommodity network design problem with similar results [32]. Over the last few years, several other authors have implemented fairly similar approaches to a variety of problems, including the partitioning of integrated circuits for logical testing [1], two-dimensional cutting [11], the loading of containers [13], labor-constrained scheduling [18], and VRPTW [86].

The same broad strategy was also followed when meta-heuristics belonging to different types were sequentially applied to a given problem. The two-phase approach of Gehring and Homberger for the VRPTW [63–65, 80] is a typical example of such a method, where each search thread first applies an evolution strategy to reduce the number of vehicles, followed by a tabu search to minimize the total distance traveled. A somewhat different two-phase pC/C parallel strategy was proposed in [9] for the Steiner problem, where each phase, using reactive tabu search and path relinking, respectively, implemented the pC/C asynchronous central-memory strategy, all processes switching from the first to the second phase simultaneously.

The central-memory pC/C approach has proved efficient in handling the problem of premature “convergence” in cooperative search. The memory contains a large set of different solutions and cooperating search threads may import different solutions even when their cooperation activities are taking place in a short time span. Furthermore, the probabilistic solution-extraction strategies used by the central-memory program yield a cooperation that continuously evolves with respect to the search threads indirectly exchanging information.

The central-memory approach also allows for more flexibility in terms of the different meta-heuristic (and exact, eventually) methods that can be combined in a same set of cooperating programs. One can thus have methods that heuristically construct new solutions, execute neighborhood-based improving meta-heuristics, evolve populations of solutions, or perform post-optimization procedures on solutions in the memory. One can thus select cooperating methods that complement each other, as illustrated in the study of Crainic and Gendreau [31], where a genetic-method thread was added to an asynchronous multi-thread tabu search for multi-commodity location-allocation with balancing requirements [40]. The tabu searches were aggressively exploring the search space, while the genetic method contributed toward increasing the diversity of solutions exchanged among the cooperating methods. The genetic method was launched once a certain number of elite solutions identified by the tabu searches were recorded in the central memory, using this memory as initial population. Asynchronous migration subsequently transferred the best solution of the genetic pool to the central memory, as well as solutions of the central memory toward the genetic population. This strategy did perform well, especially on

larger instances. It also yielded an interesting observation: the best overall solution was never found by the genetic thread, but its inclusion allowed the tabu search threads to find better solutions through what appears as a more effective diversification of the global search.

Memory-based pC/C cooperative search is also computationally efficient as no costs are incurred for inter-program synchronizations. No broadcasting is taking place and there is no need for complex mechanisms to select the threads that will receive or send information and to control the cooperation. The central memory is thus an efficient implementation device that allows for a strict asynchronous mode of exchange, with no predetermined connection pattern, where no process is interrupted by another for communication purposes, but where any thread may access at all times the data previously sent out by any other search thread. These positive qualities of the central-memory cooperation concept have naturally opened the way to the development of more advanced pC/KC mechanisms where new information is generated based on the data exchanged among cooperating threads. This is the topic of Section 17.6.3 but, first, we look at a different asynchronous cooperative search strategy based on direct information exchanges among search threads.

As mentioned earlier, one can implement direct asynchronous exchanges through local memories (blackboards). Hosted by each search thread and storing information that the host makes available for sharing, such memories can be read directly and asynchronously by adjacent search threads according to their own internal logic. Information is thus shared globally through diffusion processes.

A pC/C cooperative strategy based on these ideas and denoted *multi-level cooperative search* was proposed by Toulouse, Thulasiraman, and Glover [149]. The mechanism may be instantiated with any search differentiation strategy (the authors used MPSS) and enforces the principle of controlled diffusion of information. Each search thread works at a different level of aggregation of the original problem (one processor works on the original problem) and communicates exclusively with the threads working on the immediate higher and lower aggregation levels. Improved solutions are exchanged asynchronously at various moments dynamically determined by each thread according to its own logic, status, and search history. Received solutions are used to modify the search at the receiving level. An incoming solution will not be transmitted further until a number of iterations have been performed, thus avoiding the uncontrolled diffusion of information. Excellent results have been obtained for graph and hypergraph partitioning problems [110, 111], network design [35], feature selection in biomedical data [109], and covering design [44]. In all these cases, the proposed method is either the current best or is on the par with the best meta-heuristics for the problem.

### 17.6.3 pC/KC Asynchronous Cooperative Strategies

The exchanges performed by cooperating search threads constitute a rich source of data for constructing an approximate image of the status of the global search. It has been thus widely observed that globally optimal solutions are often similar,

e.g., in the values taken by a large number of variables. Then, because the solutions exchanged are generally locally good, being local optima in many cases, one may assume that the statistical properties of the best solutions exchanged are likely to approximate the statistical properties of the set of globally optimal solutions, and that this likelihood should increase in time with the evolution of the cooperative search. It then appears interesting to process the exchanged solutions to extract knowledge about these characteristics, and then use this knowledge to guide the search performed by the cooperating threads.

A particular form of knowledge-creation mechanism follows from the observation that the exchanged solutions form an elite population. New solutions may then be created by applying any available evolutionary meta-heuristic, solutions that improve upon their parents and thus directly enhance the global search. Moreover, these new solutions contribute to diversify the sharable information and may thus lead one or several cooperating threads to explore new regions of the search space.

Cooperative strategies including mechanisms to create new information and solutions based on the solutions exchanged belong to the p-control knowledge collegial (pC/KC) class.

Historically, two main classes of pC/KC cooperative mechanisms are found in the literature, both based on the idea of exploiting a set of elite solutions exchanged by cooperating search threads, but differing in the information that is kept in memory: *adaptive-memory* methods [128] store partial elements of good solutions and combine them to create new complete solutions that are then improved by the cooperating threads; while *central-memory* methods exchange complete elite solutions among neighborhood and population-based meta-heuristics and use them to create new solutions and knowledge to guide the cooperating threads [26, 38, 40]. The differences between the two approaches are becoming increasingly blurred, however, as the latter approach generalizes the former.

The adaptive-memory terminology was coined by Rochat and Taillard in a paper [128] proposing tabu search-based heuristics for the VRP and the VRPTW that are still among the most effective ones for both problems. (For more on adaptive-memory concepts, see [70, 138, 139].) The main idea is to keep in the memory the individual components (in routing problems, the vehicle routes) making up the elite solutions as they are found by the cooperating threads, together with memories counting the frequency of each element in the best solutions encountered so far. The elements are ranked according to the attribute values of their respective solutions, the objective value, in particular. When a cooperating thread completes its current search, it sends its best solution to the adaptive memory and, then, probabilistically selects tours in the memory to construct an initial solution for its next search. In almost all cases, the new solution will be made up of routes from different elite solutions, thus inducing a powerful diversification effect.

The adaptive-memory approach has been applied very successfully to the VRPTW [5] and [131], the latter proposing a set-covering heuristic to select the elements that will generate the new initial solution of a cooperating thread and to real-time vehicle routing and dispatching [66], within a two-level parallelization scheme: a pC/KC/MPSS cooperating adaptive-memory scheme was implemented at the first

level while, at the second level, each individual tabu search thread implemented the route decomposition of Taillard [136] with the help of several slave processors.

Badeau et al. [5] also reported a number of interesting findings for the development of asynchronous multi-thread procedures, whether they used adaptive memory or not. First, the performance of their method with respect to the quality of the solution was almost independent of the number of search processes (as long as this number remained within reasonable bounds) for a fixed computational effort (measured in terms of the overall number of calls to the adaptive memory by all search threads). Second, while traditional parallelization schemes rely on a one-to-one relationship between actual processors and search processes, it turned out that their method did run significantly faster when using more search processes than the number of available processors, because this allowed to overcome the bottlenecks created when several threads were trying to access simultaneously the processor on which the adaptive memory was located. Furthermore, computational evidence showed that it is not, in general, a good idea to run a search thread concurrently with the adaptive-memory-management procedure on the same processor.

Central-memory mechanisms keep full solutions, as well as attributes and context information sent by the search threads involved in cooperation. They include adaptive-memory concepts as special cases and are thus offering increased generality and flexibility. Cooperating methods may construct new solutions, execute a neighborhood-based improving meta-heuristic, implement a population-based meta-heuristic, or perform post-optimization procedures on solutions in the memory. Improving meta-heuristics aggressively explore the search space, while population-based methods (e.g., genetic algorithms [31, 86, 87] and path relinking [30]) contribute toward increasing the diversity of shared information (solutions) among the cooperating methods. Exact solution methods may participate to the cooperation either to build solutions or to seek out optimal ones (on restricted versions of the problem, eventually). Moreover, once complete solutions are stored in the central memory, statistics and information-extraction and creation mechanisms may be built based on any individual element of these solutions or combinations thereof. Memories recording the performance of individual solutions, solution components, or search threads may be added to the central memory and statistics, learning schemes, and guidance mechanisms may be gradually built.

Population-based methods, genetic algorithms, in particular, are often used to create new solutions. As described in Section 17.6.3, Crainic and Gendreau [31] proposed in an early study to use the “young” set of elite solutions in the central memory to populate a genetic method and, then, to asynchronously migrate elite solutions between the two populations. The concept evolved and the set of elite solutions in the central memory is currently viewed as the population to be simultaneously evolved by the cooperating threads, including one or more evolutionary methods [30, 47, 85–87].

The cooperative meta-heuristic proposed by Le Bouthiller and Crainic [86] (also used in [85, 87]) for the VRPTW had thus two simple genetic algorithms with order and edge recombination crossovers, respectively, evolving the same population with two tabu search methods that perform well sequentially, the Unified

Tabu of Cordeau, Laporte, and Mercier [25] and Taburoute of Gendreau, Hertz, and Laporte [67]. The cooperating threads shared information about their respective good solutions identified so far. When a thread improved its best solution, it sent it to the central memory, where they were considered “in-training” until they went through the post-optimization process (2-opt, 3-opt, Or-opt, and ejection-chain procedures used to reduce the number of vehicles and the total traveled distance) and become “adults.” The pool of adults in the central memory formed the elite population for the genetic operators and the tabu search procedures, which required solutions when needed (at regular intervals for the Unified Tabu and at diversification time for Taburoute). This algorithm, without any calibration or tailoring, proved to be competitive with the best meta-heuristics of its day in linear speedups.

The goal of Le Bouthiller, Crainic, and Kropf [85, 87] was to improve upon this pC/C cooperative scheme by extracting new knowledge from the information exchanged, in order to guide the individual threads and, hopefully, yield a more efficient global search. The authors also aimed for a guidance mechanism independent of particular features of the problem class at hand, e.g., routes in vehicle routing problems, and thus selected to work with one of the atomic elements of the problem: the arc.

The basic idea was that an arc that appears often in good solutions and less frequently in bad solutions may be worthy of inclusion in a tentative solution and vice versa. To implement this idea, the authors considered the frequency of inclusion of arcs in three subsets of solutions in the pool, the elite (e.g., the 10% best), average (between the 10 and 90% best), and worst (the last 10%) groups of solutions. An arc with a high frequency in a given group signals that the meta-heuristics participating to the cooperation have often produced solutions that include that arc. Patterns of arcs were then defined, representing subsets of arcs with similar frequencies of inclusion or not in particular population groups. Guidance was obtained by transmitting arc patterns to the individual threads indicating whether the arcs in the pattern should be “fixed” or “prohibited” to intensify or diversify the search, respectively (“fix” and “prohibit” were performed by using the patterns to bias the selection of arcs during moves or reproduction). The computing time allocated to the cooperative method was divided into four phases: two phases of diversification at the beginning to broaden the search, followed by two intensification phases to focus the search around promising regions. (A dynamic version of this mechanism where phases are triggered by the evolution of the population diversity and best-solution value is presented in [85].) Excellent performances in terms of solution quality and computing efficiency were observed when this pC/KC method was compared to the best-performing methods of the day.

The versatility and flexibility of the central-memory concept is also seen in the methods that start to be proposed to address so-called rich (combinatorial optimization) problems displaying multiple “attributes” characterizing their feasibility and optimality structures. The general approach when addressing such multi-attribute problems is to either simplify them or to sequentially solve a series of particular cases, where part of the overall problem is fixed or ignored, or both. It is

well-known that this leads to suboptimal solutions, however, and thus methods are being proposed to comprehensively address rich combinatorial problems, accounting for “all” their attributes simultaneously.

According to the best knowledge of the authors, Crainic et al. [30] (see also [47]) were the first to propose such a methodology in the context of designing wireless networks, where seven attributes were considered simultaneously. The authors proposed a pC/KC/MPDS parallel cooperative meta-heuristic that had tabu search solvers work on limited subsets of attributes only, while a genetic method amalgamated the partial solutions attained by the tabu search procedures into complete solution to the initial problem. The global search proceeded in three phases. To initiate the search, tabu searches started from randomly generated solutions and sent their improving solutions to the central memory to build the starting population for the evolutionary method. In the second phase, tabu search threads requested solutions at diversification time, solutions that were extracted probabilistically biased toward the best (by objective value). Finally, the third phase was activated when the global search stalled and made use of a guidance mechanism based on solution attributes different from the objective value to direct the searches of the cooperating threads toward regions of the search space where those attributes displayed desired values.

A generalization of this approach, denoted *Integrative Cooperative Search (ICS)* was introduced recently [28, 29]. In ICS, independent exact or meta-heuristic solution methods, the *Partial Solvers*, work on different subsets of attributes of the problem, while other algorithms, the *Integrators* combine the resulting partial solutions and improve them. Each Partial Solver is designed to investigate a subset of the problem attribute set and construct the set of corresponding elite *Partial Solutions* (more than one solver may be assigned to the same attribute subset, in which case, they are organized according to the central memory cooperation model). Each Integrator selects solutions from the different Partial Solution sets and constructs, and possibly improves, complete solutions to the original problem, which become part of the central memory. It is noteworthy that this is a particular case of evolution, where parents and offspring are in different populations and are conceptually different. Consequently, the method is not evolving the populations of elite partial and complete solutions in the strict sense of the term, by replacing individuals. It is rather a continuous process yielding, for each Integrator thread, one or several individuals, which enrich the set of complete solutions. Partial Solvers and Integrators cooperate through the central memory and an adaptive Global Search Coordinator. The latter is a device that monitors the central memory and the information exchanged to maintain an image of the context of the global search and of the performance and evolution of each individual search thread. Guidance mechanisms are built based on this information, in particular to steer Partial Solvers toward different regions of their search subspaces.

The last two contributions belong to a development trend still in its infancy, requiring work to fully describe its behavior and characterize its performance. The preliminary results are very promising, however, and also illustrate the interest of the pC/KC central-memory asynchronous cooperation idea.

## 17.7 Perspectives

We presented a state-of-the-art survey of the main parallel meta-heuristic ideas and strategies, discussed general design and implementation principles, and instantiated these principles for neighborhood- and population-based meta-heuristics. The survey was structured along the lines of a taxonomy of parallel meta-heuristics, which provides a rich framework for analyzing these design principles and strategies, reviewing the literature, and identifying trends and promising research directions.

To sum up, four main classes of strategies are found in the parallel meta-heuristics field: low-level decomposition of computing-intensive tasks with no modification to the original algorithm, direct decomposition of the search space, independent multi-search, and cooperative multi-search. Historically, this series corresponds to the development sequence of parallel strategies, which, initially, was proposed mainly for genetic methods, simulated annealing, and tabu search. The range of targeted meta-heuristics has broadened in recent years, multi-search strategies taking center stage. A number of studies identifying and characterizing general strategies were also proposed (see references in the Introduction) and successfully applied to various meta-heuristics and problem classes.

This is not to say that the research on parallel meta-heuristics is over. Far from it. As pointed out in the paper and summarized in the following, many open questions and challenges still face the community, both in terms of general methodology and its instantiation to the particular context of given meta-heuristics and problem classes. Indeed, such instantiations lead not only to well-adapted and performing implementations for the meta-heuristic and problem class considered, but also to a broader understanding of the methodology and its implications. This is certainly true for the more recent methods, e.g., ant-colony and other swarm-based methods, but also for the more “traditional” meta-heuristic classes where one still observes a lack of comprehensive studies focusing on these issues.

Returning to the four classes of strategies mentioned above, one should emphasize that each addresses a particular need and, therefore, they are all part of the parallel-meta-heuristic toolbox. Low-level parallelization strategies accelerate computing-intensive tasks, particularly the evaluation of a population or neighborhood, and should prove effective in addressing similar needs for swarm-inspired methods. Consequently, while the research issues are less challenging, the impact of these strategies in actual implementations may be significant, particularly as part of hierarchical strategies.

A similar case can be made for domain decomposition. Following a short period of intensive work, this strategy has been less studied in recent times. The increase in memory and computing power of contemporary computers might explain this fact. The dimensions of the problem instances one faces keep increasing, however, and strategies that provide the means to efficiently address them are certainly needed. The decomposition of the search space of the problem at hand is a “natural” approach to attacking such problems. Research is required, however, on advanced ways to dynamically perform this decomposition and the reconstruction of whole solutions. The relations to the implicit-decomposition (attribute-based) methods and

the integration with cooperative-search strategies also constitute a challenging and promising research field.

Independent multi-search offers an easy access to parallel meta-heuristic computation. The straightforward parallelization of the multi-start strategy resulting in the simultaneous exploration of the search space by (possibly different) search threads starting from different initial solutions has proved its worth in numerous studies. When one looks for a “good” solution without investment in methodological development or actual coding, independent multi-search is the appropriate tool. More refined techniques, cooperative search in particular, are needed for better results, however.

Asynchronous cooperation provides a powerful, flexible, and adaptable framework for parallel meta-heuristics that consistently achieved good results in terms of computing efficiency and solution quality for many meta-heuristic and problem classes. It is increasingly acknowledged as a meta-heuristic class in its own right and constitutes a rich and challenging research field. Among the many research issues of significant interest, we mention four.

First, the exchange and utilization of context data, in particular the memories local to the search threads, to construct an image of the status of the global search. Such a process may prove of great interest in better identifying the regions of the search space already visited and the ones where it would be interesting to direct the search (promising or not yet visited regions). It could also be extended to investigate the relative performance of the individual searches participating to the cooperation and either dynamically adjust the search parameters of some or even replacing a poorly performing method with a better performing one.

The second issue we identify is that of learning and extracting of information from the shared data. One could, in fact, see the previous issue as a particular case of this field aimed at creating new solutions and new information, e.g., patterns of attributes in given subsets of solutions in the central memory. The goal is to (1) enrich the population of elite solutions, or parts thereof, that are shared among cooperating search threads and (2) build guidance mechanisms that bring in a more consistent (and, sometimes, more directive) way the status of the global search to the search decisions of individual search threads. Research in this direction is still at the very beginning but has proved its worth. Of particular interest in this context are the studies aimed at the integrative cooperative methods, where the search space is indirectly partitioned according to subsets of problem attributes, and search threads address either the resulting subproblems or the integrative processes constructing and improving complete solutions.

More research is also warranted in mixing particular meta-heuristics and strategies. With respect to the latter issue, there is interest in the possible linkages between central-memory and multi-level strategies. Indeed, while it is important to preserve the multi-level data-exchange mechanisms that provide the means to control the diffusion of information, the introduction of a central memory, with guidance, eventually, could enhance the global search by, on one hand, more rapidly making available pertinent information and, on the other hand, creating new sharable data. With

respect to methods, we should study not only how the parallel strategies apply to the newer meta-heuristics (e.g., swarm-based) but also how these behave while part of a cooperative algorithm and to what extent and how the behavior of the latter is modified. Of equally significant interest is the role of exact methods in cooperation.

It is noteworthy, finally, that cooperation in cooperative search methods takes place at two different levels. The first is the direct and explicit information sharing specified by the cooperation mechanism, that is, by the algorithmic design of the cooperation. In this sense, it is a top-down and “local” process, exchanges taking place between particular search threads or searches and the memory, at moments determined by the algorithmic logic of search initiating communications. The second level is that of the implicit cooperation, where information spreads across all the cooperating methods through a diffusion process and correlated interactions. Implicit cooperation is bottom up and global. It is not specified algorithmically. It is rather an emergent phenomenon produced by the correlated local interactions among searches. Many research issues are related to indirect cooperation and how to harness it to enhance the optimization capabilities of cooperative meta-heuristics. The main issue is thus how to design such systems (essentially through the selection of search algorithms and direct cooperation strategies) to obtain a system-wide emergent behavior that fulfills some specific requirements, e.g., an efficient exploration of the solution space. This area of research is close to similar efforts in other fields focusing on systems that display emergent behavior through self-organization and complex adaptive behaviors, e.g., decentralized autonomic computing and social robotics. A number of concepts have been proposed in these contexts (e.g., nonlinear dynamical systems, chaos theory, attractors and system equilibrium) but have not yet resulted in significant advances for cooperative search. Research in this area will thus probably continue to be mostly empirical for some time in the future, while theoretical models are being built and put to the test. The global effort in these directions will provide us with the design tool for more powerful parallel meta-heuristics.

**Acknowledgments** The authors wish to acknowledge the contributions of their colleagues and students, in particular professor Michel Gendreau, Université de Montréal, who collaborated over the years to their work on parallel meta-heuristics for combinatorial optimization. While working on this project, the first author was the NSERC Industrial Research Chair on Logistics Management, ESG UQAM, and adjunct professor with the Department of Computer Science and Operations Research, Université de Montréal, and the Department of Economics and Business Administration, Molde University College, Norway. The second author was adjunct professor with the Department of Computer Science and Operations Research, Université de Montréal, and visiting associate professor with the Department of Computer Science at Oklahoma State University. Partial funding for this project has been provided by the Natural Sciences and Engineering Council of Canada (NSERC), through its Industrial Research Chair and Discovery Grants programs, by the partners of the Chair, CN, Rona, Alimentation Couche-Tard and the Ministry of Transportation of Québec, and by the Fonds québécois de recherche sur la nature et les technologies (FQRNT Québec) through its Team Research grants program.

## References

1. Aiex, R.M., Martins, S.L., Ribeiro, C.C., Rodriguez, N.R.: Cooperative multi-thread parallel tabu search with an application to circuit partitioning. In: Proceedings of IRREGULAR'98 - 5th International Symposium on Solving Irregularly Structured Problems in Parallel, Lecture Notes in Computer Science, vol. 1457 pp. 310–331. Springer (1998)
2. Alba, E. (ed.): Parallel Metaheuristics: A New Class of Algorithms. Wiley, Hoboken, NJ, (2005)
3. Attanasio, A., Cordeau, J.F., Ghiani, G., Laporte, G.: Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem. *Parallel Comput.* **30**, 377–387, (2004)
4. Azencott, R.: Simulated Annealing Parallelization Techniques. Wiley, New York, NY, (1992)
5. Badeau, P., Gendreau, M., Guertin, F., Potvin, J.-Y., Taillard, É.D.: A parallel tabu search heuristic for the vehicle routing problem with time windows. *Transp. Res. Part C: Emerg. Technol.* **5**(2), 109–122, (1997)
6. Banos, R., Gil, C., Ortega, J., and Montoya, F.G.: A parallel multilevel metaheuristic for graph partitioning. *J. Heuristics* **10**(4), 315–336, (2004)
7. Banos, R., Gil, C., Ortega, J., and Montoya, F.G.: Parallel heuristic search in multilevel graph partitioning. In: Proceedings of the 12th Euromicro Conference on Parallel, Distributed and Network-Based Processing, pp. 88–95, (2004)
8. Barr, R.S., Hickman, B.L.: Reporting computational experiments with parallel algorithms: issues, measures, and experts opinions. *ORSA J. Comput.* **5**(1), 2–18, (1993)
9. Bastos, M.P., Ribeiro, C.C.: Reactive tabu search with path-relinking for the Steiner problem in graphs. In: Voß, S., Martello, S., Roucairol, C., Osman, I.H., (eds.) *Meta-Heuristics 98: Theory & Applications*, pp. 31–36. Kluwer, Norwell, MA, (1999)
10. Battiti, R., Tecchiolli, G.: Parallel based search for combinatorial optimization: genetic algorithms and TABU. *Microproc. Microsy.* **16**(7), 351–367, (1992)
11. Blazewicz, J., Moret-Salvador, A., Walkowiak, R.: Parallel tabu search approaches for two-dimensional cutting. *Parallel Process. Lett.* **14**(1), 23–32, (2004)
12. Bock, S., Rosenberg O.: A New parallel breadth first tabu search technique for solving production planning problems. *Int. Trans. Oper. Res.* **7**(6), 625–635, (2000)
13. Bortfeldt, A., Gehring, H., Mack, D.: A Parallel tabu search algorithm for solving the container loading problem. *Parallel Comput.* **29**, 641–662, (2003)
14. Bullnheimer, B., Kotsis, G., Strauß, C.: Parallelization strategies for the ant system. In: De Leone, R., Murli, A., Pardalos, P., Toraldo, G. (eds.) *High Performance Algorithms and Software in Nonlinear Optimization*, volume 24 of *Applied Optimization*, pages 87–100. Kluwer, Dordrecht, (1999)
15. Calégari, P., Guidec, F., Kuonen, P., Kuonen, D.: Parallel island-based genetic algorithm for radio network design. *J. Parallel Distrib. Comput.* **47**(1), 86–90, (1997)
16. Cantú-Paz, E.: A survey of parallel genetic algorithms. *Calculateurs Parallèles, Réseaux et Systèmes répartis* **10**(2), 141–170, (1998)
17. Cantú-Paz, E.: Theory of parallel genetic algorithms. In: Alba, E., (ed.) *Parallel Metaheuristics: A New Class of Algorithms*, pp. 425–445. Wiley, Hoboken, (2005)
18. Cavalcante, C.B.C., Cavalcante, V.F., Ribeiro, C.C., de Souza, C.C.: Parallel cooperative approaches for the labor constrained scheduling problem. In: Ribeiro C.C., Hansen, P., (eds.) *Essays and Surveys in Metaheuristics*, pp. 201–225. Kluwer, Norwell, MA, (2002)
19. Chakrapani, J., Skorin-Kapov, J.: A connectionist approach to the quadratic assignment problem. *Comput. Oper. Res.* **19**(3/4), 287–295, (1992)
20. Chakrapani, J., Skorin-Kapov, J.: Connection machine implementation of a tabu search algorithm for the traveling salesman problem. *J. Comput. Inf. Technol.* **1**(1), 29–36, (1993)
21. Chakrapani, J., Skorin-Kapov, J.: Massively parallel tabu search for the quadratic assignment problem. *Ann. Oper. Res.* **41**, 327–341, (1993)

22. Cohoon, J., Hedge, S., Martin, W., Richards, D.: Punctuated equilibria: a parallel genetic algorithm. In: Grefenstette, J.J., (ed.) Proceedings of the 2nd International Conference on Genetic Algorithms and their Applications, pp. 148–154. Lawrence Erlbaum Associates, Hillsdale, NJ, (1987)
23. Cohoon, J., Martin, W., Richards, D.: Genetic algorithm and punctuated equilibria in VLSI. In: Schewefel, H.-P., Männer, R. (eds.) Parallel Problem Solving from Nature. Lecture Notes in Computer Science, vol. 496 pp. 134–144. Springer, Berlin, (1991a)
24. Cohoon, J., Martin, W., Richards, D.: A multi-population genetic algorithm for solving the k-partition problem on hyper-cubes. In Belew, R.K., Booker, L.B. (eds.) Proceedings of the 4th International Conference on Genetic Algorithms, pp. 134–144. Morgan Kaufmann, San Mateo, CA, (1991b)
25. Cordeau, J.-F., Laporte, G., Mercier, A.: A unified tabu search heuristic for vehicle routing problems with time windows. *J. Oper. Res. Soc.* **52**, 928–936, (2001)
26. Crainic, T.G.: Parallel computation, co-operation, tabu Search. In: Rego, C., Alidaee, B. (eds.) Metaheuristic Optimization Via Memory and Evolution: Tabu Search and Scatter Search, pp. 283–302. Kluwer, Norwell, MA, (2005)
27. Crainic, T.G.: Parallel solution methods for vehicle routing problems. In: Golden, B., Raghavan, S., Wasil, E. (eds.) The Vehicle Routing Problem: Latest Advances and New Challenges, pp. 171–198. Springer, New York, (2008)
28. Crainic, T.G., Crisan, C.C., Gendreau, M., Lahrichi, N., Rei, W.: A concurrent evolutionary approach for cooperative rich combinatorial optimization. In: Proceedings of Genetic and Evolutionary Computation Conference - GECCO 2009, July 8-12, Montréal, Canada. CD-ROM, ACM, Order no. 910092, (2009)
29. Crainic, T.G., Crisan, C.C., Gendreau, M., Lahrichi, N., Rei, W.: Multi-thread integrative cooperative optimization for rich combinatorial problems. In: Proceedings of The 12th International Workshop on Nature Inspired Distributed Computing - NIDISC'09, 25-29 May, Rome, CD-ROM (2009)
30. Crainic, T.G., Di Chiara, B., Nonato, M., Tarricone, L.: Tackling electrosmog in completely configured 3G networks by parallel cooperative meta-Heuristics. *IEEE Wireless Commun.* **13**(6), 34–41, (2006)
31. Crainic, T.G., Gendreau, M.: Towards an evolutionary method - cooperating multi-thread parallel tabu search hybrid. In: Voß, S., Martello, S., Roucairol, C., Osman, I.H. (eds.) Meta-Heuristics 98: Theory & Applications, pp. 331–344. Kluwer, Norwell, MA, (1999)
32. Crainic, T.G., Gendreau, M.: Cooperative parallel tabu search for capacitated network design. *J. Heuristics* **8**(6), 601–627, (2002)
33. Crainic, T.G., Gendreau, M., Hansen, P., Mladenović, N.: Cooperative parallel variable neighborhood search for the  $p$ -median. *J. Heuristics* **10**(3), 293–314, (2004)
34. Crainic, T.G., Gendreau, M., Potvin, J.-Y.: Parallel tabu search. In: Alba, E. (ed.) Parallel Metaheuristics, pp. 298–313. Wiley, Hoboken, NJ, (2005)
35. Crainic, T.G., Li, Y., Toulouse, M.: A first multilevel cooperative algorithm for the capacitated multicommodity network design. *Comput. Oper. Res.* **33**(9), 2602–2622, (2006)
36. Crainic, T.G., Nourredine, H.: Parallel meta-heuristics applications. In: Alba, E. (ed.) Parallel Metaheuristics: A New Class of Algorithms, pp. 447–494. Wiley, Hoboken, NJ, (2005)
37. Crainic, T.G., Toulouse, M.: Parallel metaheuristics. In: Crainic, T.G., Laporte, G. (eds.) Fleet Management and Logistics, pp. 205–251. Kluwer, Norwell, MA, (1998)
38. Crainic, T.G., Toulouse, M.: Parallel strategies for meta-heuristics. In: Glover, F., Kochenberger, G. (eds.) Handbook in Metaheuristics, pp. 475–513. Kluwer, Norwell, MA, (2003)
39. Crainic, T.G., Toulouse, M., Gendreau, M.: Synchronous tabu search parallelization strategies for multicommodity location-allocation with balancing requirements. *OR Spektrum* **17**(2/3), 113–123, (1995)

40. Crainic, T.G., Toulouse, M., Gendreau, M.: Parallel asynchronous tabu search for multicommodity location-allocation with balancing requirements. *Annals Oper. Res.* **63**, 277–299, (1996)
41. Crainic, T.G., Toulouse, M., Gendreau, M.: Towards a taxonomy of parallel tabu search algorithms. *INFORMS J. Comput.* **9**(1), 61–72, (1997)
42. Cung, V.-D., Martins, S.L., Ribeiro, C.C., Roucairol, C.: Strategies for the parallel implementations of metaheuristics. In: Ribeiro, C.C., Hansen, P. (eds.) *Essays and Surveys in Metaheuristics*, pp. 263–308. Kluwer, Norwell, MA, (2002)
43. Czech, Z.J.: A parallel genetic algorithm for the set partitioning problem. In: 8th Euromicro Workshop on Parallel and Distributed Processing, pp. 343–350, (2000)
44. Dai, C., Li, B., Toulouse, M.: A multilevel cooperative tabu search algorithm for the covering design problem. *J. Comb. Math. Comb. Comput.* **68**, 35–65, (2009)
45. De Falco, I., Del Balio, R., Tarantino, E.: Solving the mapping problem by parallel tabu search. Report, Istituto per la Ricerca sui Sistemi Informatici Parallel-CNR, (1995)
46. De Falco, I., Del Balio, R., Tarantino, E., Vaccaro, R.: Improving Search by Incorporating Evolution Principles in Parallel Tabu Search. In: Proceedings International Conference on Machine Learning, pp. 823–828, (1994)
47. Di Chiara, B.: Optimum planning of 3G cellular systems: radio propagation models and cooperative parallel meta-heuristics. PhD thesis, Dipartimento di Ingegneria dell'Innovazione, Università degli Studi di Lecce, Lecce, Italy, (2005–2006)
48. Diekmann, R., Lüling, R., Monien, B., Spräuer, C.: Combining helpful sets and parallel simulated annealing for the graph-partitioning problem. *Int. J. Parallel Programming* **8**, 61–84, (1996)
49. Doerner, K., Hartl, R.F., Kiechle, G., Lucka, M., Reimann, M.: Parallel ant systems for the capacitated vehicle routing problem. In: Gottlieb, J., Raidl, G.R. (eds.) *Evolutionary Computation in Combinatorial Optimization: 4th European Conference, EvoCOP 2004*, volume 3004 of *Lecture Notes in Computer Science*, pp. 72–83. Springer, Berlin, (2004)
50. Doerner, K.F., Hartl, R.F., Benkner, S., Lucka, M.: Cooperative savings based ant colony optimization - multiple search and decomposition approaches. *Parallel Processing Lett.* **16**(3), 351–369, (2006)
51. Doerner, K.F., Hartl, R.F., Lucka, M.: A parallel version of the D-ant algorithm for the vehicle routing problem. In: Vajtersic, M., Trobec, R., Zinterhof, P., Uhl, A. (eds.) *Parallel Numerics'05*, pp. 109–118. Springer, New York, NY, (2005)
52. Dorigo, M., Stuetzle, T.: The ant colony metaheuristic. algorithms, applications, and advances. In: Glover, F., Kochenberger, G. (eds.) *Handbook in Metaheuristics*, pp. 251–285. Kluwer, Norwell, MA, (2003)
53. Drias, H., Iibri, A.: Parallel ACS for weighted MAX-SAT. In: Mira, J., Álvarez, J. (eds.) *Artificial Neural Nets Problem Solving Methods - Proceedings of the 7th International Work-Conference on Artificial and Natural Neural Networks*, Lecture Notes in Computer Science, vol. 2686 pp. 414–421. Springer, Heidelberg, (2003)
54. Fiechter, C.-N.: A parallel tabu search algorithm for large travelling salesman problems. *Discrete Appl. Math.* **51**(3), 243–267, (1994)
55. Flores, S.D., Cegla, B.B., Caceres, D.B.: Telecommunication network design with parallel multi-objective evolutionary algorithms. In: IFIP/ACM Latin America Networking Conference 2003, (2003)
56. Folino, G., Pizzuti, C., Spezzano, G.: Combining cellular genetic algorithms and local search for solving satisfiability problems. In: Proceedings of the 10th IEEE International Conference on Tools with Artificial Intelligence, pp. 192–198. IEEE Computer Society Press, (1998)
57. Folino, G., Pizzuti, C., Spezzano, G.: Solving the satisfiability problem by a parallel cellular genetic algorithm. In Proceedings of the 24th EUROMICRO Conference, pp. 715–722. IEEE Computer Society Press, (1998)
58. Garcia, B.L., Potvin, J.-Y., Rousseau, J.M.: A parallel implementation of the tabu search heuristic for vehicle routing problems with time window constraints. *Comput. Oper. Res.* **21**(9), 1025–1033, (1994)

59. García-López, F., García Torres, M., Melián-Batista, B., Moreno-Pérez, J.A., Moreno-Vega, J.M.: Parallel scatter search. In: Alba, E. (ed.) *Parallel Metaheuristics: A New Class of Metaheuristics*, pp. 223–246. Wiley, Hoboken, NJ, (2005)
60. García-López, F., García Torres, M., Melián-Batista, B., Moreno-Pérez, J.A., Moreno-Vega, J.M.: Solving feature subset selection problem by a parallel scatter search. *Eur. J. Oper. Res.* **169**, 477–489, (2006)
61. García-López, F., Melián-Batista, B., Moreno-Pérez, J.A., Moreno-Vega, J.M.: The parallel variable neighborhood search for the  $p$ -median problem. *J. Heuristics* **8**(3), 375–388, (2002)
62. García-López, F., Melián-Batista, B., Moreno-Pérez, J.A., Moreno-Vega, J.M.: Parallelization of the scatter search for the  $p$ -median problem. *Parallel Comput.* **29**, 575–589, (2003)
63. Gehring, H., Homberger, J.: A parallel hybrid evolutionary metaHeuristic for the vehicle routing problem with time windows. In: Miettinen, K., Mäkelä, M.M., Toivanen, J. (eds.) *Proceedings of EUROGEN99 - Short Course on Evolutionary Algorithms in Engineering and Computer Science*, pp. 57–64. Jyväskylä, Finland, (1997)
64. Gehring, H., Homberger, J.: A parallel two-phase metaheuristic for routing problems with time windows. *Asia-Pacific J. Oper. Res.* **18**(1), 35–47, (2001)
65. Gehring, H., Homberger, J.: Parallelization of a two-phase metaheuristic for routing problems with time windows. *J. Heuristics* **8**, 251–276, (2002)
66. Gendreau, M., Guertin, F., Potvin, J.-Y., Taillard, É.D.: Tabu search for real-time vehicle routing and dispatching. *Transp. Sci.* **33**(4), 381–390, (1999)
67. Gendreau, M., Hertz, A., Laporte, G.: A tabu search heuristic for the vehicle routing problem. *Manage. Sci.* **40**, 1276–1290, (1994)
68. Gendreau, M., Laporte, G., Semet, F.: A dynamic model and parallel tabu search heuristic for real-time ambulance relocation. *Parallel Comput.* **27**(12), 1641–1653, (2001)
69. Glover, F.: Future paths for integer programming and links to artificial intelligence. *Comput. Oper. Res.* **1**(3), 533–549, (1986)
70. Glover, F.: Tabu search and adaptive memory programming – advances, applications and challenges. In: Barr, R.S., Helgason, R.V., Kennington, J. (eds.) *Interfaces in Computer Science and Operations Research*, pp. 1–75. Kluwer, Norwell, MA, (1996)
71. Glover, F., Laguna, M.: Tabu search. In: Reeves, C.R. (ed.) *Modern Heuristic Techniques for Combinatorial Problems*, pp. 70–150. Blackwell Scientific Publications, Oxford, (1993)
72. Glover, F., Laguna, M.: Tabu Search. Kluwer, Norwell, MA, (1997)
73. Glover, F., Taillard, É.D., de Werra, D.: A User’s Guide to Tabu Search. *Annals Oper. Res.* **41**, 3–28, (1993)
74. Greening, D.R.: A taxonomy of parallel simulated annealing techniques. Technical Report No. RC 14884. IBM, (1989)
75. Greening, D.R.: Asynchronous parallel simulated annealing. *Lect. Comp. Syst.* **3**, 497–505, (1990)
76. Greening, D.R.: Parallel simulated annealing techniques. *Physica D* **42**, 293–306, (1990)
77. Herdy, M.: Reproductive isolation as strategy parameter in hierarchical organized evolution strategies. In: Männer, R., Manderick, B. (eds.) *Parallel Problem Solving from Nature*, 2, pp. 207–217. North-Holland, Amsterdam, (1992)
78. Hidalgo, J.I., Prieto, M., Lanchares, J., Baraglia, R., Tirado, F., Garnica, O.: Hybrid parallelization of a compact genetic algorithm. In: Proceedings of the 11th Uromicro Conference on Parallel, Distributed and Network-Based Processing, pp. 449–455, (2003)
79. Holmqvist, K., Migdalas, A., Pardalos, P.M.: Parallelized heuristics for combinatorial search. In: Migdalas, A., Pardalos, P.M., Storoy, S. (eds.) *Parallel Computing in Optimization*, pp. 269–294. Kluwer, Norwell, MA, (1997)
80. Homberger, J., Gehring, H.: Two evolutionary metaheuristics for the vehicle routing problem with time windows. *INFOR* **37**, 297–318, (1999)
81. Janson, S., Merkle, D., Middendorf, M.: Parallel ant colony algorithms. In: Alba, E. (ed.) *Parallel Metaheuristics: A New Class of Metaheuristics*, pp. 171–201. Wiley, Hoboken, NJ, (2005)
82. Kravitz, S.A., Rutenbar, R.: Placement by simulated annealing on a multiprocessor. *IEEE Trans. Comput. Aid. Des.* **6**, 534–549, (1987)

83. Laganière, R. Mitiche, A.: Parallel tabu search for robust image filtering. In Proceedings of IEEE Workshop on Nonlinear Signal and Image Processing (NSIP'95), vol. 2, pp. 603–605, (1995)
84. Laursen, P.S.: Parallel heuristic search – introductions and a new approach. In: Ferreira, A., Pardalos, P.M. (eds.) Solving Combinatorial Optimization Problems in Parallel, Lecture Notes in Computer Science 1054, vol. 1054 pp. 248–274. Springer, Berlin, (1996)
85. Le Bouthillier, A.: Recherches coopératives pour la résolution de problèmes d'optimisation combinatoire. PhD thesis, Département d'informatique et de recherche opérationnelle, Université de Montréal, Montréal, QC, Canada, (2007)
86. Le Bouthillier, A. Crainic, T.G.: A cooperative parallel meta-heuristic for the vehicle routing problem with time windows. *Comput. Oper. Res.* **32**(7), 1685–1708, (2005)
87. Le Bouthillier, A., Crainic, T.G., Kropf, P.: A guided cooperative search for the vehicle routing problem with time windows. *IEEE Intell. Syst.* **20**(4), 36–42, (2005)
88. Lee, K.G. Lee, S.Y.: Efficient parallelization of simulated annealing using multiple markov chains: an application to graph partitioning. In: Mudge, T.N. (ed.) Proceedings of the International Conference on Parallel Processing, volume III: Algorithms and Applications, pp. 177–180. CRC Press, (1992)
89. Lee, K.G. Lee, S.Y.: Synchronous and asynchronous parallel simulated annealing with multiple Markov chains. In: Brandenburg, F.J. (ed.) Graph Drawing - Proceedings GD '95, Symposium on Graph Drawing, Passau, Germany, Lecture Notes in Computer Science, vol. 1027 pp. 396–408. Springer, Berlin, (1995)
90. Lee, S.Y. Lee, K.G.: Asynchronous communication of multiple markov chains in parallel simulated annealing. In: Mudge, T.N. (ed.) Proceedings of the International Conference on Parallel Processing, volume III: Algorithms and Applications, pp. 169–176. CRC Press, Boca Raton, FL, (1992)
91. Lee, S.Y. Lee, K.G.: Synchronous and Asynchronous Parallel Simulated Annealing with Multiple Markov Chains. *IEEE Trans. Parallel Distribut. Syst.* **7**(10), 993–1007, (1996)
92. Li, Y., Pardalos, P.M., Resende, M.G.C.: A greedy randomized adaptive search procedure for quadratic assignment problem. In DIMACS Implementation Challenge, DIMACS Series on Discrete Mathematics and Theoretical Computer Science, vol. 16, pp. 237–261. American Mathematical Society, (1994)
93. Lin, S.-C., Punch, W., Goodman, E.: Coarse-grain parallel genetic algorithms: categorization and new approach. In 6th IEEE Symposium on Parallel and Distributed Processing, pp. 28–37. IEEE Computer Society Press, (1994)
94. Malek, M., Guruswamy, M., Pandya, M., Owens, H.: Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem. *Annals Oper. Res.* **21**, 59–84, (1989)
95. Martins, S.L., Resende, M.G.C., Ribeiro, C.C., Parlados, P.M.: A parallel grasp for the steiner tree problem in graphs using a hybrid local search strategy. *J. Global Optimization* **17**, 267–283, (2000)
96. Martins, S.L., Ribeiro, C.C., Souza, M.C.: A parallel GRASP for the Steiner problem in graphs. In: Ferreira, A., Rolim, J. (eds.) Proceedings of IRREGULAR'98 – 5th International Symposium on Solving Irregularly Structured Problems in Parallel, Lecture Notes in Computer Science, vol. 1457 pp. 285–297. Springer, (1998)
97. Michels, R. Middendorf, M.: An ant system for the shortest common supersequence problem. In: Corne, D., Dorigo, M., Glover, F. (eds.) New Ideas in Optimization, pp. 51–61. McGraw-Hill, (1999)
98. Middendorf, M., Reischle, F., Schmeck, S.: Multi colony ant algorithms. *J. Heuristics* **8**(3), 305–320, (2002)
99. Miki, M., Hiroyasu, T., Wako, J., Yoshida, T.: Adaptive temperature schedule determined by genetic algorithm for parallel simulated annealing. In: CEC'03 - The 2003 Congress on Evolutionary Computation, vol. 1, pp. 459–466, (2003)

100. Moreno-Pérez, J.A., Hansen, P., Mladenović, N.: Parallel variable neighborhood search. In: Alba, E. (ed.) *Parallel Metaheuristics: A New Class of Metaheuristics*, pp. 247–266. Wiley, Hoboken, NJ, (2005)
101. Mühlenbein, H.: Parallel genetic algorithms, population genetics and combinatorial optimization. In: Schaffer, J.D., (ed.) *Proceedings of the 3rd International Conference on Genetic Algorithms*, pp. 416–421. Morgan Kaufmann, San Mateo, CA, (1989)
102. Mühlenbein, H.: Evolution in time and space - the parallel genetic algorithm. In: Rawlins, G.J.E. (ed.) *Foundations of Genetic Algorithm & Classifier Systems*, pp. 316–338. Morgan Kaufman, San Mateo, CA, (1991)
103. Mühlenbein, H.: Parallel genetic algorithms, population genetics, and combinatorial optimization. In: Becker, J.D., I. Eisele, Mündemann, F.W. (eds.) *Parallelism, Learning, Evolution: Workshop on Evolutionary Models and Strategies - WOPPLOT 89*, pp. 398–406. Springer, Berlin, (1991)
104. Mühlenbein, H.: Parallel genetic algorithms in combinatorial optimization. In: Balci, O., Sharda, R., Zenios, S. (eds.) *Computer Science and Operations Research: New Developments in their Interface*, pp. 441–456. Pergamon Press, New York, NY, (1992)
105. Mühlenbein, H.: How genetic algorithms really work: mutation and hill-climbing. In Männer, R., Manderick, B. (eds.) *Parallel Problem Solving from Nature*, 2, pp. 15–26. North-Holland, Amsterdam, (1992a)
106. Mühlenbein, H., Gorges-Schleuter, M., Krämer, O.: New solutions to the mapping problem of parallel systems - the evolution approach. *Parallel Comput.* **6**, 269–279, (1987)
107. Mühlenbein, H., Gorges-Schleuter, M., Krämer, O.: Evolution algorithms in combinatorial optimization. *Parallel Comput.* **7**(1), 65–85, (1988)
108. Niar, S. Fréville, A.: A parallel tabu search algorithm for the 0-1 multidimensional knapsack problem. In: 11th International Parallel Processing Symposium (IPPS '97), Geneva, Switzerland, pp. 512–516. IEEE, (1997)
109. Oduntan, I.O., Toulouse, M., Baumgartner, R., Bowman, C., Somorjai, R., Crainic, T.G.: A multilevel tabu search algorithm for the feature selection problem in biomedical data Sets. *Comput. Math. Appl.* **55**(5), 1019–1033, (2008)
110. Ouyang, M., Toulouse, M., Thulasiraman, K., Glover, F., Deogun, J.S.: Multi-level cooperative search: application to the netlist/hypergraph partitioning problem. In *Proceedings of International Symposium on Physical Design*, pp. 192–198. ACM Press, (2000)
111. Ouyang, M., Toulouse, M., Thulasiraman, K., Glover, F., Deogun, J.S.: Multilevel cooperative search for the circuit/hypergraph partitioning problem. *IEEE Trans. Comput.-Aid. Des.* **21**(6), 685–693, (2002)
112. Pardalos, P.M., Li, Y., Murthy, K.A.: Computational experience with parallel algorithms for solving the quadratic assignment problem. In: Balci, O., Sharda, R., Zenios, S. (eds.) *Computer Science Operations Research: New Developments in their Interface*, pp. 267–278. Pergamon Press, New York, NY, (1992)
113. Pardalos, P.M., L. Pitsoulis, T. Mavridou, Resende, M.G.C.: Parallel search for combinatorial optimization: genetic algorithms, simulated annealing, tabu Search and GRASP. In: Ferreira, A., Rolim, J. (eds.) *Proceedings of Workshop on Parallel Algorithms for Irregularly Structured Problems*, Lecture Notes in Computer Science, vol. 980, pp. 317–331. Springer, Berlin, (1995)
114. Pardalos, P.M., Pitsoulis, L., Resende, M.G.C.: A parallel GRASP implementation for the quadratic assignment problem. In: Ferreira, A., Rolim, J. (eds.) *Solving Irregular Problems in Parallel: State of the Art*, pp. 115–130. Kluwer, Norwell, MA, (1995)
115. Porto, S.C.S., Kitajima, J.P.F.W., Ribeiro, C.C.: Performance evaluation of a parallel tabu search task scheduling algorithm. *Parallel Comput.* **26**, 73–90, (2000)
116. Porto, S.C.S. Ribeiro, C.C.: A tabu search approach to task scheduling on heterogenous processors under precedence constraints. *Int. J. High-Speed Comput.* **7**, 45–71, (1995)
117. Porto, S.C.S. Ribeiro, C.C.: Parallel tabu search message-passing synchronous strategies for task scheduling under precedence constraints. *J. Heuristics* **1**(2), 207–223, (1996)

118. Rahoual, M., Hadji, R., Bachelet, V.: Parallel ant system for the set covering problem. In: Dorigo, M., Di Caro, G., Sampels, M. (eds.) *Ant Algorithms - Proceedings of the 3rd International Workshop, ANTS 2002*, Lecture Notes in Computer Science, vol. 2463 pp. 262–267. Springer, Berlin, (2002)
119. Ram, D.J., Sreenivas, T.H., Subramaniam, K.G.: Parallel simulated annealing algorithms. *J. Parallel Distribut. Comput.* **37**, 207–212, (1996)
120. Randall, M., Lewis, A.: A parallel implementation of ant colony optimisation. *J. Parallel Distribut. Comput.* **62**, 1421–1432, (2002)
121. Rego, C.: Node ejection chains for the vehicle routing problem: sequential and parallel algorithms. *Parallel Comput.* **27**, 201–222, (2001)
122. Rego, C., Roucairol, C.: A parallel tabu search algorithm using ejection chains for the VRP. In: Osman, I.H., Kelly, J.P. (eds.) *Meta-Heuristics: Theory & Applications*, pp. 253–295. Kluwer, Norwell, MA, (1996)
123. Reimann, M., Doerner, K., Hartl, R.: D-Ants: Savings based ants divide and conquer the vehicle routing problem. *Comput. Oper. Res.* **31**(4), 563–591, (2004)
124. Reimann, M., Stummer, M., Doerner, K.: A savings based ants system for the vehicle routing problem. In: Langton, C., Cantú-Paz, E., Mathias, K.E., Roy, R., Davis, L., Poli, R., Balakrishnan, K., Honavar, V., Rudolph, G., Wegener, J., Bull, L., Potter, M.A., Schultz, A.C., Miller, J.F., Burke, E.K., Jonoska, N. (eds.) *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, New York, USA, July 9-13, 2002, pp. 1317–1326. Morgan Kaufmann Publishers, San Francisco, CA, (2002)
125. Ribeiro, C.C., Rossetti, I.: A parallel GRASP heuristic for the 2-path network design problem. 4 journée ROADEF, Paris, February 20-22, (2002)
126. Ribeiro C.C., Rossetti, I.: A parallel GRASP heuristic for the 2-path network design problem. Third Meeting of the PAREO Euro Working Group, Guadeloupe (France), May, (2002)
127. Ribeiro C.C., Rossetti, I.: Parallel grasp with path-relinking heuristic for the 2-path network design problem. AIRO'2002, L'Aquila, Italy, September, (2002)
128. Rochat, Y., Taillard, É.D.: Probabilistic diversification and intensification in local search for vehicle routing. *J. Heuristics* **1**(1), 147–167, (1995)
129. Sanvicente-Sánchez, H., Frausto-Solís, J.: MPSA: a methodology to parallelize simulated annealing and its application to the traveling salesman problem. In: Coello Coello, C.A., de Albornoz, A., Sucar, L.E., Battistutti, O.C. (eds.) *MICAI 2002: Advances in Artificial Intelligence*, volume 2313 of *Lecture Notes in Computer Science*, pp. 89–97. Springer Heidelberg, (2002)
130. Schlierkamp-Voosen, D., Mühlenbein, H.: Strategy adaptation by competing subpopulations. In: Davidor, Y., Schwefel, H.-P., Männer, R. (eds.) *Parallel Problem Solving from Nature III*, Lecture Notes in Computer Science, vol. 866 pp. 199–208. Springer, Berlin, (1994)
131. Schulze, J., Fahle, T.: A parallel algorithm for the vehicle routing problem with time window constraints. *Annals Oper. Res.* **86**, 585–607, (1999)
132. Shonkwiler, R.: Parallel genetic algorithms. In: Forrest, S. (ed.) *Proceedings of the 5th International Conference on Genetic Algorithms*, pp. 199–205. Morgan Kaufmann, San Mateo, CA, (1993)
133. Solar, M., Parada, V., Urrutia, R.: A parallel genetic algorithm to solve the set-covering problem. *Comput. Oper. Res.* **29**(9), 1221–1235, (2002)
134. Stutzle, T.: Parallelization strategies for ant colony optimization. In: Eiben, A.E., Back, T., Schoenauer, M., Schwefel, H.-P. (eds.) *Proceedings of Parallel Problem Solving from Nature V*, Lecture Notes in Computer Science, vol. 1498 pp. 722–731. Springer, Heidelberg, (1998)
135. Taillard, É.D.: Robust taboo search for the quadratic assignment problem. *Parallel Comput.* **17**, 443–455, (1991)
136. Taillard, É.D.: Parallel iterative search methods for vehicle routing problems. *Networks* **23**, 661–673, (1993)
137. Taillard, É.D.: Parallel taboo search techniques for the job shop scheduling problem. *ORSA J. Comput.* **6**(2), 108–117, (1994)

138. Taillard, É.D., Gambardella, L.M., Gendreau, M., Potvin, J.-Y.: Adaptive memory programming: a unified view of metaheuristics. *Eur. J. Oper. Res.* **135**, 1–10, (1997)
139. Taillard, É.D., Gambardella, L.M., Gendreau, M., Potvin, J.-Y.: Programmation à mémoire adaptative. *Calculateurs Parallèles, Réseaux et Systèmes répartis* **10**, 117–140, (1998)
140. Talbi, E.-G., Hafidi, Z., Geib, J.-M.: Parallel adaptive tabu search approach. *Parallel Comput.* **24**, 2003–2019, (1998)
141. Talbi, E.-G., Roux, O., Fonlupt, C., Robillard, D.: Parallel ant colonies for combinatorial optimization problems. In: José Rolim et al. (ed.) 11th IPPS/SPDP'99 Workshops Held in Conjunction with the 13th International Parallel Processing Symposium and 10th Symposium on Parallel and Distributed Processing, April 12–16, San Juan, Puerto Rico, Lecture Notes in Computer Science, vol. 1586 pp. 239–247. Springer, Berlin, (1999)
142. ten Eikelder, H.M.M., Aarts, B.J.M., Verhoeven, M.G.A., Aarts, E.H.L.: Sequential and parallel local search for job shop scheduling. In: Voß, S., Martello, S., Roucairol, C., Osman, I.H. (eds.) *Meta-Heuristics 98: Theory & Applications*, pp. 359–371. Kluwer, Norwell, MA, (1999)
143. Tongcheng, G. Chundi, M.: Radio network design using coarse-grained parallel genetic algorithms with different neighbor topology. In: Proceedings of the 4th World Congress on Intelligent Control and Automation, vol. 3, pp. 1840–1843, (2002)
144. Toulouse, M., Crainic, T.G., Gendreau, M.: Communication issues in designing cooperative multi thread parallel searches. In: Osman I.H., Kelly, J.P. (eds.) *Meta-Heuristics: Theory & Applications*, pp. 501–522. Kluwer, Norwell, MA, (1996)
145. Toulouse, M., Crainic, T.G., Sansó, B.: An Experimental study of systemic behavior of cooperative search algorithms. In: Voß, S., Martello, S., Roucairol, C., Osman, I.H. (eds.) *Meta-Heuristics 98: Theory & Applications*, pp. 373–392. Kluwer, Norwell, MA, (1999)
146. Toulouse, M., Crainic, T.G., Sansó, B.: Systemic behavior of cooperative search algorithms. *Parallel Comput.* **30**(1), 57–79, (2004)
147. Toulouse, M., Crainic, T.G., Sansó, B., Thulasiraman, K.: Self-organization in cooperative search algorithms. In: Proceedings of the 1998 IEEE International Conference on Systems, Man, and Cybernetics, pp. 2379–2385. Omnipress, Madison, WI, (1998)
148. Toulouse, M., Crainic, T.G., Thulasiraman, K.: Global optimization properties of parallel cooperative search algorithms: a simulation study. *Parallel Comput.* **26**(1), 91–112, (2000)
149. Toulouse, M., Thulasiraman, K., Glover, F.: Multi-level cooperative search: a new paradigm for combinatorial optimization and an application to graph partitioning. In: Amestoy, P., Berger, P., Daydé, M., Duff, I., Frayssé, V., Giraud, L., Ruiz, D. (eds.) 5th International Euro-Par Parallel Processing Conference, Lecture Notes in Computer Science, vol. 1685 pp. 533–542. Springer, Heidelberg, (1999)
150. Verhoeven, M.G.A. Aarts, E.H.L.: Parallel local search. *J. Heuristics* **1**(1), 43–65, (1995)
151. Voß, S.: Tabu search: applications and prospects. In: Du, D.-Z., Pardalos, P.M. (eds.) *Network Optimization Problems*, pp. 333–353. World Scientific Publishing, Singapore, (1993)
152. Wilkerson, R., Nemer-Preece, N.: Parallel genetic algorithm to solve the satisfiability problem. In Proceedings of the 1998 ACM symposium on Applied Computing, pp. 23–28. ACM Press, (1998)



# Chapter 18

## Reactive Search Optimization: Learning While Optimizing

Roberto Battiti and Mauro Brunato

**Abstract** Reactive Search Optimization advocates the integration of sub-symbolic machine learning techniques into search heuristics for solving complex optimization problems. The word *reactive* hints at a ready response to events during the search through an internal online feedback loop for the self-tuning of critical parameters. Methodologies of interest for Reactive Search Optimization include machine learning and statistics, in particular reinforcement learning, active or query learning, neural networks, and meta-heuristics (although the boundary signalled by the “meta” prefix is not always clear).

### 18.1 Introduction

The final purpose of Reactive Search Optimization (RSO) is to simplify the life for the final user of optimization. While researchers enjoy designing algorithms, testing alternatives, tuning parameters, and choosing solution schemes—in fact this is part of their daily life—the final users’ interests are different: solving a problem in the most effective way without requiring a costly adaptation and learning curve.

Reactive Search Optimization has to do with *learning for optimizing*, with the insertion of a machine learning component into a solution process so that algorithm selection, adaptation, integration are done in an automated way and a comprehensive solution is delivered to the final user. The diversity of tasks, stochasticity, dynamicity which is intrinsic in real-world tasks can be dealt with in a seamless manner. The interaction with the final user is simplified and made human: no complex technical

---

Roberto Battiti  
LION Lab, Università di Trento, Italy  
e-mail: roberto.battiti@unitn.it

Mauro Brunato  
LION Lab, Università di Trento, Italy  
e-mail: mauro.brunato@unitn.it

questions are asked about parameters, but the focus is kept on the problem's detailed characteristics and user preferences. In fact, the user wants to maintain control of the problem definition, including of course hard and soft constraints, preferences, weights. This is the part which cannot be automated, while the user is happy to delegate issues related to algorithm choices and tuning.

Needless to say, studying and designing satisfactory solutions to the above final goal is a long-term enterprise with opportunities for PhD students and researchers of this century, but we feel that the road is clear and that preliminary results of interest abound.

Apart from the above concrete issues related to the final user, Reactive Search Optimization also addresses a scientific issue related to the reproducibility of results and to the objective evaluation of methods. In fact, if an intelligent user is actively in the loop between a parametric algorithm and the solution of a problem, judging about an algorithm in isolation from its user—in some cases its creator—becomes difficult if not impossible. Are the obtained results a merit of the algorithm or a merit of its intelligent user? In some cases the second case holds, which explains why even some naïve and simplistic techniques can obtain results of interest if adopted by a motivated person, not to say by a researcher in love with his pet algorithm and under pressure to get something published.

Now that the long-term vision is given, let us come to a more detailed definition.

Reactive Search Optimization (RSO) advocates the integration of machine learning techniques into search heuristics for solving complex optimization problems. The word *reactive* hints at a ready response to events while alternative solutions are tested, through an internal online feedback loop for the self-tuning of critical parameters. Its strength lies in the introduction of high-level skills often associated to the human brain, such as learning from the past experience, learning on the job, rapid analysis of alternatives, ability to cope with incomplete information, quick adaptation to new situations and events.

If one considers the dictionary definition of *reactive*, see the box below, the “ready response to some treatment, situation, or stimulus” is the part of interest to us. The contrary in our context is inactive, inert, unresponsive. For sure, its contrary is not proactive! In fact, when the level of automation increases, the final user wins, but the work becomes much more challenging for the researcher: he has to be fully proactive to anticipate the different adaptation needs of a Reactive Search Optimization algorithm.

**re·ac·tive**

1: of, relating to, or marked by reaction or reactance

2 a: readily responsive to a stimulus b: occurring as a result of stress or emotional upset

**re·ac·tion**

Date: circa 1611

1 a: the act or process or an instance of reacting b: resistance or opposition to a force, influence, or movement ...  
2: a response to some treatment, situation, or stimulus "her stunned reaction to the news" ...  
3: bodily response to or activity aroused by a stimulus: a: an action induced by vital resistance to another action ; especially : the response of tissues to a foreign substance (as an antigen or infective agent) ...  
4: the force that a body subjected to the action of a force from another body exerts in the opposite direction  
5 a (1): chemical transformation or change : the interaction of chemical entities (2): the state resulting from such a reaction b: a process involving change in atomic nuclei  
(derived from: *Merriam-Webster online dictionary*)

Before dwelling on the technical details, let us briefly mention some relevant characteristics of Reactive Search Optimization when applied in the context of local search-based processes.

**Learning on the job** Real-world problems have a rich structure. While many alternative solutions are tested in the exploration of a search space, patterns and regularities appear. The human brain quickly learns and drives future decisions based on previous observations. This is the main inspiration source for inserting online machine learning techniques into the optimization engine of RSO.

**Rapid generation and analysis of many alternatives** Often, to solve a problem one searches among a large number of alternatives, each requiring the analysis of what-if scenarios. The search speed is improved if alternatives are generated in a strategic manner, so that different solutions are chained along a trajectory in the search space exploring wide areas and rapidly exploiting the most promising solutions.

**Flexible decision support** Crucial decisions depend on several factors and priorities which are not always easy to describe before starting the solution process. Feedback from the user in the preliminary exploration phase can be incorporated so that a better tuning of the final solutions takes the end user preferences into account.

**Diversity of solutions** The final decision is up to the user, not the machine. The reason is that not all qualitative factors of a problem can be encoded into a computer program. Having a set of diverse near-best alternatives is often crucial for the decision maker.

**Anytime solutions** The user decides when to stop searching. A first complete solution is generated rapidly, better and better ones are produced in the following search phases. The more the program runs, the bigger the possibility to identify excellent solutions, but if you want a solution fast you are going to get it!

Methodologies of interest for Reactive Search Optimization include machine learning and statistics, in particular neural networks, artificial intelligence, reinforcement learning, active or query learning.

When one considers the *source* of information that is used for the algorithm selection and tuning process, it is important to stress that there are at least three different possibilities:

1. *Problem-dependent information.* This is related to characteristics of the specific problem. For example, a local search scheme for the Traveling Salesman Problem needs a different neighborhood definition w.r.t. a scheme for the network partitioning problem.
2. *Task-dependent information.* A single problem consists of a set of instances or tasks with characteristics which can be radically different. For example, a Traveling Salesman task for delivering pizza among a set locations in Los Angeles can be very different from a pizza delivery task in Trento, a small and pleasant town in the Alps.
3. *Local properties in configuration space.* When one considers a local search scheme based on perturbation one builds a trajectory in configuration space given by successive sample points generated by selecting and applying the local moves. In poetic terms, one travels along a fitness surface with peaks and valleys which can vary a lot during the trip. For example, the size and depth of the attractors around local minimizers can vary from a reasonably flat surface to one characterized by deep wells. If a scheme for escaping local minimizers is adapted also to the local characteristics, better results can be expected.

Now, the first possibility is the typical source of information for off-line algorithm selection and parameter tuning, while the last two possibilities are the starting point for the online schemes of RSO, where parameters are dynamically tuned based on the current optimization state and previous history of the search process while working on a specific instance.

*Intelligent optimization*, a superset of Reactive Search Optimization, refers to a more extended area of research, including online and off-line schemes based on the use of memory, adaptation, incremental development of models, experimental algorithmics applied to optimization, intelligent tuning, and design of heuristics.

The RSO approach of learning on the job is to be contrasted with off-line parameter tuning. This orthogonal approach is studied, for example in [79, 80], that proposes methods to predict per-instance and per-parameter run-times with reasonable accuracy. These predictive models are then used to predict which parameter settings result in the lowest run-time for a given instance, thus automatically tuning the parameter values of a stochastic local search (SLS) algorithm on a per-instance basis by simply picking the parameter configuration that is predicted to yield the lowest run-time. An iterated local search (ILS) algorithm for the algorithm configuration problem is proposed in [81]. The approach works for both deterministic and randomized algorithms and can be applied regardless of tuning scenario and optimization objective.

Online and off-line strategies are complementary: in fact, even RSO methods tend to have a number of parameters that remain fixed during the search and can hence be tuned by off-line approaches.

The following part of this chapter is organized as follows. First the different opportunities for RSO strategies are listed and briefly commented. Section 18.2

describes different RSO schemes that have been introduced in the literature. A much more extended presentation has been recently published in [14]. Then sample applications of Reactive Search Optimization principles are illustrated in Section 18.3.

The brevity of this chapter does not allow for a complete listing and examination: we ask the omitted authors for forgiveness and encourage authors of novel work to get in touch with us. The Reactive Search Optimization community<sup>1</sup> and software<sup>2</sup> web sites are two additional sources of information which can be mined for more detailed interests.

## 18.2 Different Reaction Possibilities

The design principles of many superficially different techniques for diversifying the search in a responsive manner according to the RSO principles of learning while optimizing are strongly related. The unifying principle is that of using online reactive learning schemes to increase the robustness and to permit more *hands-off* usage of software for optimization.

For brevity we concentrate this review chapter on reactive techniques applied to single local search streams. Other possibilities related to using more than one search stream, aka population-based methods, genetic algorithms, evolutionary techniques, range from adaptive portfolios to restart strategies, to racing techniques, to intelligent and reactive solver teams [14].

### 18.2.1 Reactive Prohibitions

It is part of commonsense that the discovery of radically new solutions which is associated to real creativity demands departing from the usual way of doing things, avoiding known solutions. The popular concepts of “lateral thinking” and “thinking outside the box” are related to shifting the point of view, observing an old problem with new eyes, discarding pet hypotheses.

Techniques that apply lateral thinking to problems are characterized by the shifting of thinking patterns, away from entrenched or predictable thinking to new or unexpected ideas. A new idea that is the result of lateral thinking is not always a helpful one, but when a good idea is discovered in this way it is usually obvious in hindsight, which is a feature lateral thinking shares with a joke.

There are a number of mental tools or methods that can be used to bring about lateral thinking. These include the following:

...

<sup>1</sup> <http://reactive-search.org/>

<sup>2</sup> <http://reactive-search.com/>

Provocation: Declare the usual perception out of bounds, or provide some provocative alternative to the usual situation under consideration. . . .

As an example see the provocation on cars having square wheels.

Challenge: Simply challenge the way things have always been done or seen, or the way they are. This is done not to show there is anything wrong with the existing situation but simply to direct your perceptions to exploring outside the current area.

For example you could challenge coffee cups being produced with a handle. There is nothing wrong with coffee cups having handles so the challenge is a direction to explore without defending the status quo. The reason for the handle seems to be that the cup is often too hot to hold directly. Perhaps coffee cups could be made with insulated finger grips . . .

There are many other techniques . . . All these tools are practical matters for circumstances where our normal automatic perceptions and pattern matching tend to keep us trapped “within the box”.

(derived from Wikipedia “lateral thinking” voice, Jan 2008)

When one reflects about the above connections, it is not surprising to see ideas related to using “prohibitions” to encourage diversification and exploration (the technical terms for true creativity in the context of optimization heuristics) in different contexts and different times. For example, they can be found in the *denial* strategy of [121]: once common features are detected in many suboptimal solutions, they are *forbidden*.

The full blossoming of “intelligent prohibition-based heuristics” starting from the late 1980s is greatly due to the role of F. Glover in the proposal and diffusion of a rich variety of meta-heuristic tools under the umbrella of Tabu Search (TS) [68, 69], but see also [73] for an independent seminal paper. It is evident that Glover’s ideas have been a source of inspiration for many approaches based on the intelligent use of memory in heuristics.

The main competitive advantage of TS with respect to alternative heuristics based on local search like Simulated Annealing (SA) lies in the intelligent use of the past history of the search to influence its future steps. Because TS includes now a wide variety of methods, we prefer the term *prohibition-based search* when the investigation is focussed onto the use of prohibition to encourage diversification.

Let us assume that the feasible search space is the set of binary strings with a given length  $L$ :  $\mathcal{X} = \{0, 1\}^L$ .  $X^{(t)}$  is the current configuration and  $N(X^{(t)})$  the set of its neighbors, i.e., configurations that can be explored in the following step (Section 18.2.2 is mainly focused on neighborhoods). In prohibition-based search some of the neighbors are *prohibited*, a subset  $N_A(X^{(t)}) \subset N(X^{(t)})$  contains the *allowed* ones. The general way of generating the search trajectory is given by

$$X^{(t+1)} = \text{BEST-NEIGHBOR}(N_A(X^{(t)})), \quad (18.1)$$

$$N_A(X^{(t+1)}) = \text{ALLOW}(N(X^{(t+1)}), X^{(0)}, \dots, X^{(t+1)}). \quad (18.2)$$

The set-valued function ALLOW selects a subset of  $N(X^{(t+1)})$  in a manner that depends on the entire search trajectory  $X^{(0)}, \dots, X^{(t+1)}$ .

By analogy with the concept of *abstract data type* in Computer Science [2], and with the related *object-oriented* software engineering framework [49], it is useful to separate the abstract concepts and operations of TS from the detailed implementation, i.e., realization with specific data structures. In other words, *policies* (that determine which trajectory is generated in the search space, what the balance of intensification and diversification is, etc.) should be separated from *mechanisms* that determine *how* a specific policy is realized. A first classification distinguishes between *strict-TS*, which prohibits only the moves leading back to previously visited configurations, and *fixed-TS*, which prohibits only the inverses of moves which have been applied recently in the search, their recency being judged according to a prohibition parameter  $T$ , also called tabu tenure.

Let  $\mu^{-1}$  denote the *inverse* of a move, for example, if  $\mu_i$  is changing the  $i$ -th bit of a binary string from 0 to 1,  $\mu_i^{-1}$  changes the same bit from 1 to 0. A neighbor is allowed if and only if it is obtained from the current point by applying a move such that its inverse has not been used during the last  $T$  iterations. In detail, if LASTUSED( $\mu$ ) is the last usage time of move  $\mu$  ( $\text{LASTUSED}(\mu) = -\infty$  at the beginning):

$$N_A(X^{(t)}) = \{X = \mu \circ X^{(t)} \text{ s. t. } \text{LASTUSED}(\mu^{-1}) < (t - T)\}. \quad (18.3)$$

If  $T$  changes with the iteration counter depending on the search status, and in this case the notation is  $T^{(t)}$ , the general dynamical system that generates the search trajectory comprises an additional evolution equation for  $T^{(t)}$ :

$$T^{(t)} = \text{REACT}(T^{(t-1)}, X^{(0)}, \dots, X^{(t)}), \quad (18.4)$$

$$N_A(X^{(t)}) = \{X = \mu \circ X^{(t)} \text{ s. t. } \text{LASTUSED}(\mu^{-1}) < (t - T^{(t)})\}, \quad (18.5)$$

$$X^{(t+1)} = \text{BEST-NEIGHBOR}(N_A(X^{(t)})). \quad (18.6)$$

Rules to determine the prohibition parameter by reacting to the repetition of previously visited configurations have been proposed in [26] (*reactive-TS*, RTS for short). In addition, there are situations where the single reactive mechanism on  $T$  is not sufficient to avoid long cycles in the search trajectory and therefore a second reaction is needed [26].

The prohibition parameter  $T$  used in Equation (18.3) is related to the amount of *diversification*: the larger  $T$ , the longer the distance that the search trajectory must go before it is allowed to come back to a previously visited point. In particular, the following relationships between prohibition and diversification are demonstrated in [11] for a search space consisting of binary strings with basic moves flipping individual bits:

- The Hamming distance  $H$  between a starting point and successive points along the trajectory is strictly increasing for  $T + 1$  steps:

$$H(X^{(t+\Delta t)}, X^{(t)}) = \Delta t \quad \text{for } \Delta t \leq T + 1.$$

- The minimum repetition interval  $R$  along the trajectory is  $2(T + 1)$ :

$$X^{(t+R)} = X^{(t)} \Rightarrow R \geq 2(T + 1).$$

In general, because a larger prohibition value implies a more limited choice of moves, it makes sense to set  $T$  to the smallest value that guarantees a sufficient degree of diversification.

In *reactive-TS* [26] the prohibition  $T$  is determined through feedback (i.e., *reactive*) mechanisms during the search.  $T$  is equal to one at the beginning (the inverse of a given move is prohibited only at the next step), it increases only when there is *evidence* that diversification is needed, it decreases when this evidence disappears. The evidence that diversification is needed is signaled by the repetition of previously visited configurations. This criterion needs to be generalized when the search space dimension becomes very large, so that the exact repetition of configurations can become very rare even if the trajectory is confined. In this case, one can monitor an appropriate distance measure from a given starting configuration. An insufficient growth of the distance as a function of the number of steps can be taken as evidence of confinement, see, for example, [20].

A more radical *escape* mechanism can be triggered when the basic prohibition mechanism is not sufficient to guarantee diversification. In [26] the escape (a number of random steps) is triggered when too many configurations are repeated too often. Further details about applications, implementation, and data structures can be found in [14].

A reactive determination of the  $T$  value can change the process of escaping from a local minimum in a qualitative manner: one obtains an (optimistic) logarithmic increase in the *strict-TS* algorithm, and an (pessimistic) increase that behaves like the square root of the number of iterations in the reactive case [14].

Robust stochastic algorithms related to the previously described deterministic versions can be obtained in many ways. For example, prohibition rules can be substituted with *probabilistic generation-acceptance rules* with large probability for allowed moves, small for prohibited ones, see, for example, the *probabilistic-TS* [68]. Asymptotic results for TS can be obtained in probabilistic TS [56]. In a different proposal (*robust-TS*) the prohibition parameter is randomly changed between an upper and a lower bound during the search [122].

Finally, other possibilities which are softer than prohibitions exist. For example, the HSAT [67] variation of GSAT introduces a tie-breaking rule into GSAT: if more than one move produces the same (best)  $\Delta f$ , the preferred move is the one that has not been applied for the longest span. HSAT can be seen as a “soft” version of Tabu Search: while TS prohibits recently applied moves, HSAT discourages recent moves if the same  $\Delta f$  can be obtained with moves that have been “inactive” for a longer time.

### 18.2.2 Reacting on the Neighborhood

Local search based on perturbing a candidate solution is a first paradigmatic case where simple online adaptation and learning strategies can be applied. Let  $\mathcal{X}$  be the

search space,  $X^{(t)}$  the current solution at iteration (“time”)  $t$ .  $N(X^{(t)})$  is the neighborhood of point  $X^{(t)}$ , obtained by applying a set of basic moves  $\mu_0, \mu_1, \dots, \mu_M$  to the current configuration:

$$N(X^{(t)}) = \{X \in \mathcal{X} \text{ such that } X = \mu_i(X^{(t)}), i = 0, \dots, M\}.$$

*Local search* starts from an admissible configuration  $X^{(0)}$  and builds a *search trajectory*  $X^{(0)}, \dots, X^{(t+1)}$ . The successor of the current point is a point in the neighborhood with a lower value of the function  $f$  to be minimized. If no neighbor has this property, i.e., if the configuration is a local minimizer, the search stops:

$$Y \leftarrow \text{IMPROVING-NEIGHBOR}(N(X^{(t)})), \quad (18.7)$$

$$X^{(t+1)} = \begin{cases} Y & \text{if } f(Y) < f(X^{(t)}) \\ X^{(t)} & \text{otherwise (search stops).} \end{cases} \quad (18.8)$$

*IMPROVING-NEIGHBOR* returns an improving element in the neighborhood. In a simple case this is the element with the lowest  $f$  value, but other possibilities exist, as we will see in what follows.

Online learning strategies can be applied in two contexts: selection of the neighbor or selection of the neighborhood. While these strategies are part of the standard bag of tools, they in fact can be seen as simple forms of reaction to the recent history of evaluations.

When the neighborhood is fixed, one can modify the unresponsive strategy which considers all neighbors before selecting one of the best moves (*best-improvement local search*) and obtain a very simple reactive strategy like FIRSTMOVE. FIRSTMOVE accepts the first improving neighbor if one is found before examining all candidates. The simple adaptation is clear: the exact number of neighbors evaluated before deciding the next move depends not only on the instance but also on the particular local properties in the configuration space around the current point. On the average, less neighbors will need to be evaluated at the beginning of the search, when finding an improving move is simple, more neighbors when the trajectory goes deeper and deeper into a given local minimum attractor.

When the neighborhood is changed depending on the local configuration one obtains, for example, the Variable Neighborhood Search (VNS) [72]. VNS considers a *set of neighborhoods*, defined a priori at the beginning of the search, and then uses the most appropriate one during the search.

Variable Neighborhood Descent (VND) [74], see Figure 18.1, uses the default neighborhood first and the ones with a higher number only if the default neighborhood fails (i.e., the current point is a local minimum for  $N_1$ ), and only until an improving move is identified, after which it reverts back to  $N_1$ . When VND is coupled with an ordering of the neighborhoods according to the *strength* of the perturbation, one realizes the principle *use the minimum strength perturbation leading to an improved solution*, which is present also in more advanced RSO methods. The consideration of neighborhoods of increasing diameter (distance of its members w.r.t. the starting configuration) can be considered as a form of *diversification*. A strong similarity with the design principle of Reactive Tabu Search is present, see later in

```

1. function VariableNeighborhoodDescent ( $N_1, \dots, N_{k_{max}}$ )
2.   repeat until no improvement or max CPU time elapsed
3.      $k \leftarrow 1$  // index of the default neighborhood
4.     while  $k \leq k_{max}$ :
5.        $X' \leftarrow \text{BestNeighbor } (N_k(X))$  // neighborhood exploration
6.       if  $f(X') < f(X)$ 
7.          $X \leftarrow X'$ ;  $k \leftarrow 1$  // success: back to default neighborhood
8.       else
9.          $k \leftarrow k + 1$  // try with the following neighborhood

```

**Fig. 18.1** The VND routine. Neighborhoods with higher numbers are considered only if the default neighborhood fails and only until an improving move is identified.  $X$  is the current point.

this chapter, where diversification through prohibitions is activated when there is evidence of entrapment in an attraction basin and gradually reduced when there is evidence that a new basin has been discovered.

An explicitly reactive-VNS is considered in [35] for the Vehicle Routing Problem with Time Windows (VRPTW), where a construction heuristic is combined with VND using first-improvement local search. Furthermore, the objective function used by the local search operators is modified to consider the waiting time to escape from a local minimum. A preliminary investigation about a self-adaptive neighborhood ordering for VND is presented in [78]. The different neighborhoods are ranked according to their observed benefits in the past.

Let us also note some similarities between VNS and the adaptation of the search region in stochastic search techniques for continuous optimization, see the discussion later in this chapter. Neighborhood adaptation in the continuous case, see for example, the Affine Shaker algorithm in [25], is mainly considered to speed-up convergence to a local minimizer, not to jump to nearby valleys.

A related possibility to cause a more radical move when simple ones are not sufficient to escape from a local minimum is *iterated local search* (ILS). ILS is based on building a sequence of locally optimal solutions by perturbing the current local minimum and applying local search after starting from the modified solution. The work about large-step Markov chain of [94–96, 126] contains very interesting results coupled with a clear description of the principles.

In VNS minimal perturbations maintain the trajectory in the starting attraction basin, while excessive ones bring the method closer to a random sampling, therefore loosing the boost which can be obtained by the problem structural properties. A possible solution consists of perturbing by a short random walk of a length which is *adapted* by statistically monitoring the progress in the search. Memory and reactive learning can be used in a way similar to [20] to adapt the *strength* of the perturbation to the local characteristics in the neighborhood of the current solution for the considered instance. Creative perturbations can be obtained by temporarily changing the objective function with penalties so that the current local minimum is displaced, like in [31, 45]; see also the later description about reactively changing the objective function or by *fixing* some configuration variables and optimizing sub-parts of the problem [92].

### 18.2.3 Reacting on the Annealing Schedule

A widely popular stochastic local search technique is the Simulated Annealing (SA) method [88] based on the theory of Markov processes. The trajectory is built in a randomized manner: the successor of the current point is chosen stochastically, with a probability that depends only on the difference in  $f$  value w.r.t. the current point and not on the previous history:

$$Y \leftarrow \text{NEIGHBOR}(N(X^{(t)})),$$

$$X^{(t+1)} \leftarrow \begin{cases} Y & \text{if } f(Y) \leq f(X^{(t)}) \\ Y & \text{if } f(Y) > f(X^{(t)}), \text{ with probability } p = e^{-(f(Y)-f(X^{(t)}))/T} \\ X^{(t)} & \text{if } f(Y) > f(X^{(t)}), \text{ with probability } (1-p). \end{cases} \quad (18.9)$$

SA introduces a *temperature* parameter  $T$  which determines the probability that worsening moves are accepted: a larger  $T$  implies that more worsening moves tend to be accepted and therefore a larger diversification occurs. An analogy with energy-minimization principles in physics is present and this explains the “temperature term,” as well as the term “energy” to refer to the function  $f$ .

If the local configuration is close to a local minimizer and the temperature is already very small in comparison to the upward jump which has to be executed to escape from the attractor, the system will *eventually* escape, but an enormous number of iterations can be spent around the attractor. The memoryless property (current move depending only on the current state, not on the previous history) makes SA look like a dumb animal indeed. It is intuitive that a better performance can be obtained by using memory, self-analyzing the evolution of the search, developing simple models, and activating more direct *escape* strategies aiming at a better usage of the computational resources devoted to optimization.

Even if a vanilla version of a cooling schedule for SA is adopted (starting temperature  $T_{\text{start}}$ , geometric cooling schedule  $T_{t+1} = \alpha T_t$ , with  $\alpha < 1$ , final temperature  $T_{\text{end}}$ ), a sensible choice has to be made for the three involved parameters  $T_{\text{start}}$ ,  $\alpha$ , and  $T_{\text{end}}$ . The work [130] suggests to estimate the distribution of  $f$  values. The standard deviation of the energy distribution defines the maximum-temperature scale, while the minimum change in energy defines the minimum-temperature scale. These temperature scales tell us where to begin and end an annealing schedule.

The analogy with physics is further pursued in [89], where concepts related to *phase transitions* and *specific heat* are used. The idea is that a phase transition is related to solving a sub-part of a problem. After a phase transition corresponding to the big reconfiguration occurs, finer details in the solution have to be fixed, and this requires a slower decrease of the temperature.

When the parameters  $T_{\text{start}}$  and  $\alpha$  are fixed a priori, the useful span of CPU time is practically limited. After the initial period the temperature will be so low that the system *freezes* and, with large probability, no tentative moves will be accepted anymore in the remaining CPU time of the run. For a new instance, guessing appropriate

parameter values is difficult. Furthermore, in many cases one would like to use an *anytime algorithm*, so that longer allocated CPU times are related to possibly better and better values until the user decides to stop. *Non-monotonic cooling schedules* are a reactive solution to this difficulty, see [1, 46, 105]. The work [46] suggests to reset the temperature once and for all at a constant temperature not only high enough to escape local minima but also low enough to visit them, for example, at the temperature  $T_{\text{found}}$  when the best heuristic solution is found in a preliminary SA simulation.

A non-monotonic schedule aims at exploiting an attraction basin rapidly by decreasing the temperature so that the system can settle down close to the local minimizer, *increasing the temperature* to diversify the solution and visit other attraction basins, decreasing again after reaching a different basin. The implementation details have to do with determining an *entrapment* situation, for example, from the fact that no tentative move is accepted after a sequence  $t_{\max}$  of tentative changes and determining the detailed temperature decrease–increase evolution as a function of events occurring during the search [1, 105]. Enhanced versions involve a learning process to choose a proper value of the heating factor depending on the system state. Let us note that similar “strategic oscillations” have been proposed in tabu search, in particular in the reactive tabu search of [26], see later in this chapter, and in variable neighborhood search.

Modifications departing from the exponential acceptance rule and other adaptive stochastic local search methods for combinatorial optimization are considered in [99, 100]. The authors appropriately note that the optimal choices of algorithm parameters depend not only on the problem but also on the particular instance and that a proof of convergence to a globally optimum is not a selling point for a specific heuristic: in fact a simple random sampling or even exhaustive enumeration (if the set of configurations is finite) will eventually find the optimal solution, although they are not the best algorithms to suggest. A simple adaptive technique is suggested in [100]: a perturbation leading to a worsening solution is accepted if and only if a fixed number of trials could not find an improving perturbation. The temperature parameter is eliminated. The positive performance of the method in the area of design automation suggests that the success of SA is “due largely to its acceptance of bad perturbations to escape from local minima rather than to some mystical connection between combinatorial problems and the annealing of metals.”

“Cybernetic” optimization is proposed in [60] as a way to use probabilistic information for feedback during a run of SA. The idea is to consider more runs of SA running in parallel and to aim at *intensifying the search* (by lowering the temperature parameter) when there is evidence that the search is converging to the optimum value.

The application of SA to continuous optimization (optimization of functions defined on real variables) is pioneered by [48]. The basic method is to generate a new point with a random step along a direction  $\mathbf{e}_h$ , to evaluate the function and to accept the move with the exponential acceptance rule. One cycles over the different directions  $\mathbf{e}_h$  during successive steps of the algorithm. A first critical choice has to

do with the range of the random step along the chosen direction. A fixed choice obviously may be very inefficient: this opens a first possibility for *learning* from the local  $f$  surface. In particular a new trial point  $\mathbf{x}'$  is obtained from the current point  $\mathbf{x}$  as

$$\mathbf{x}' = \mathbf{x} + \text{RAND}(-1, 1)v_h \mathbf{e}_h,$$

where  $\text{RAND}(-1, 1)$  returns a random number uniformly distributed between  $-1$  and  $1$ ,  $\mathbf{e}_h$  is the unit-length vector along direction  $h$ , and  $v_h$  is the step-range parameter, one for each dimension  $h$ , collected into the vector  $\mathbf{v}$ . The  $v_h$  value is adapted during the search to maintain the number of *accepted* moves at about one-half of the total number of tried moves. Although the implementation is already reactive and based on memory, the authors encourage more work so that a “good monitoring of the minimization process” can deliver precious feedback about some crucial internal parameters of the algorithm.

In Adaptive Simulated Annealing (ASA), also known as very fast simulated annealing [82], the parameters that control the temperature cooling schedule and the random step selection are automatically adjusted according to algorithm progress. If the state is represented as a point in a box and the moves as an oval cloud around it, the temperature and the step size are adjusted so that all of the search space is sampled at a coarse resolution in the early stages, while the state is directed to promising areas in the later stages.

A reactive determination of parameters in an advanced simulated annealing application for protein folding is presented in [75].

### 18.2.4 Reacting on the Objective Function

In the above methods, the objective function  $f$  remains the guiding source of information to select the next move. Reactive diversification to encourage exploration of areas which are distant from a locally optimal configuration has been considered through an adaptive selection of the neighborhood or of the neighbor based on the local situation and on the past history of the search process. A more direct way to force diversification is to directly prohibit configurations or moves to create a pressure to reach adequate distances from a starting point.

This part considers a different way to achieve similar results, by reactively changing the function guiding the local search. For example, the act of visiting a local minimum may cause a local increase of the evaluation function value so that the point becomes less and less appealing, until eventually the trajectory is gently pushed to other areas. Of course, the real objective function values and the corresponding configurations are saved into memory before applying the modification process. The physics analogy is that of pushing a ball out of a valley by progressively raising the bottom of the valley.

A relevant problem for which objective function modifications have been extensively used is maximum satisfiability (MAX-SAT): the input consists of logic variables—with false and true values—and the objective is to satisfy the maximum number of clauses (a clause is the logical OR of literals, a literal is a variable or its

negation). The decision version is called SAT, one searches for a variable assignment, if any exists, which makes a formula true.

The influential algorithm GSAT [117] is based on local search with the standard basic moves flipping the individual variables (from false to true and vice versa). Different noise strategies to escape from locally optimal configurations are added to GSAT in [115, 116]. In particular, the GSAT-with-walk algorithm introduces random walk moves with a certain probability. A prototypical evaluation function modification algorithm is the breakout method proposed in [98] for the related constraint satisfaction problem. The cost is measured as the sum of the weights associated to the violated constraints. Each weight is one at the beginning, at a local minimum the weight of each nogood is increased by one until one escapes from the given local minimum (a breakout occurs). Clause-weighting has been proposed in [114] for GSAT. A positive weight is associated with each clause to determine how often the clause should be counted when determining which variable to flip. The weights are dynamically modified during problem solving and the qualitative effect is that of “filling in” local optima while the search proceeds. Clause-weighting and the breakout technique can be considered as “reactive” techniques where a repulsion from a given local optimum is generated in order to induce an escape from a given attraction basin.

New clause-weighting parameters are introduced and therefore new possibilities for tuning the parameters based on feedback from preliminary search results. The algorithm in [113] suggests to use weights to encourage more priority on satisfying the “most difficult” clauses. One aims at *learning how difficult a clause is to satisfy*. These hard clauses are identified as the ones which remain unsatisfied after a try of local search descent followed by plateau search. Their weight is increased so that future runs will give them more priority when picking a move. More algorithms based on the same weighting principle are proposed in [63, 64], where clause weights are updated after each flip: the reaction from the unsatisfied clauses is now immediate as one does not wait until the end of a try (weighted GSAT or WGSAT). If weights are only increased, after some time their size becomes large and their relative magnitude will reflect the overall statistics of the SAT instance, more than the local characteristics of the portion of the search space where the current configuration lies. To combat this problem, two techniques are proposed in [64], either *reducing* the clause weight when a clause is satisfied or storing the weight increments which took place recently, which is obtained by a weight decay scheme (each weight is reduced by a factor  $\phi$  before updating it). Depending on the size of the increments and decrements, one achieves “continuously weakening incentives not to flip a variable” instead of the strict prohibitions of Tabu Search. The second scheme takes the *recency of moves* into account, the implementation is through a weight decay scheme updating so that each weight is reduced before a possible increment by  $\delta$  if the clause is not satisfied:

$$w_i \leftarrow \phi w_i + \delta,$$

where one introduces a decay rate  $\phi$  and a “learning rate”  $\delta$ . A faster decay (lower  $\phi$  value) will limit the temporal extension of the context and imply a faster forgetting

of old information. A critique of some *warping* effects that a clause-weighting dynamic local search can create on the fitness surface is present in [123]: in particular let us note that the fitness surface is changed in a global way after encountering a local minimum. Points which are very far from the local minimum, but which share some of the unsatisfied clauses, will also see their values changed.

A more recent proposal of a dynamic local search (DLS) for SAT is in [124]. The authors start from the Exponentiated Sub-Gradient (ESG) algorithm [112], which alternates search phases and weight updates and develop a scheme with low time complexity of its search steps: Scaling and Probabilistic Smoothing (SAPS). Weights of satisfied clauses are multiplied by  $\alpha_{\text{sat}}$ , while weights of unsatisfied clauses are multiplied by  $\alpha_{\text{unsat}}$ , then all weights are smoothed toward their mean  $\bar{w}$ :  $w \leftarrow w \rho + (1 - \rho) \bar{w}$ . A *reactive version* of SAPS (RSAPS) is then introduced that adaptively tunes one of the algorithm's important parameters.

A similar approach of dynamically modifying the objective function has been proposed with the term of Guided Local Search (GLS) [127, 128] for other applications. GLS aims at enabling intelligent search schemes that exploit problem- and search-related information to guide a local search algorithm. Penalties depending on solution features are introduced and dynamically manipulated to distribute the search effort over the regions of a search space. A penalty formulation for TSP including memory-based trap-avoidance strategies is proposed in [129]. One of the strategies avoids visiting points that are close to points visited before, a generalization of the previously described STRICT-TS strategy. A recent algorithm with an *adaptive clause weight redistribution* is presented in [83]. It adopts resolution-based preprocessing and reactive adaptation of the total amount of weight to the degree of stagnation of the search.

Let us note that the use of a dynamically modified (learned) evaluation function is related to the machine learning technique of *reinforcement learning* (RL). Early applications of RL in the area of local search is presented in [33, 34]. Some reinforcement learning approaches for optimization are also discussed in [8]. Recent work includes [15], on-the-fly parameter tuning for evolutionary algorithms in [55], and the presentation in [14].

### 18.3 Applications of Reactive Search Optimization

It must be noted that Reactive Search Optimization is not a rigid algorithm but a general framework: specific algorithms have been introduced for unconstrained and constrained tasks, with different stochastic mechanisms and rules for selecting neighbors. As it usually happens in heuristics, the more specific knowledge about a problem is used, the better the results. Nonetheless, it was often the case that simple RSO versions realized with very limited effort could duplicate the performance of more complex schemes because of their simple embedded feedback loop and without intensive parameter and algorithm tuning. A long-term goal of RSO is the progressive shift of the learning capabilities from the algorithm user to the algorithm itself, through machine learning techniques.

The RSO framework and related algorithms and tools have been and are being applied to a growing number of “classical” discrete optimization problems, continuous optimization tasks, and real-world problems arising in widely different contexts. The Web, see for example, Google scholar, lists thousands of citations to the seminal papers, the following list is a selection of some applications we are aware of. We are always happy to hear from users and developers interested in RSO principles and applications.

In the following we summarize some applications in “classical” combinatorial tasks in Section 18.3.1, where by classical we mean abstract definitions of problems which have been extensively studied in the Computer Science and Operations Research community.

Then we present applications in the area of neural networks in Section 18.3.2, where RSO has been used to solve the optimization problems related to machine learning. Let us note that the synergy between optimization and machine learning is explored in the opposite direction in this case, i.e., of using optimization to solve machine learning tasks.

Then we discuss versions of RSO for continuous optimization tasks in Section 18.3.3.

Finally, in Section 18.3.4, we present some applications to problems which are closer to the real application areas. These problems are of course related to their abstract and clean definitions but usually contains more details and require more competence in the specific area to make substantial progress.

### 18.3.1 Classic Combinatorial Tasks

The seminal paper about RSO for Tabu Search (Reactive Tabu Search) presented preliminary experimental results on the 0-1 Knapsack Problem and on the Quadratic Assignment Problem [26]. A comparison with Simulated Annealing on QAP tasks is contained in [27]. An early experimental comparison of Reactive Search Optimization with alternative heuristics (Repeated Local Minima Search, Simulated Annealing, Genetic Algorithms, and Mean Field Neural Networks) is presented in [28]. An application of a self-controlling software approach to Reactive Tabu Search is presented in [57] with results on the QAP problem.

#### 18.3.1.1 Knapsack and Related Problems

A reactive local search-based algorithm (adaptive memory search) for the 0/1-Multidemand Multidimensional Knapsack Problem (0/1-MDMKP) is proposed in [5]. The 0/1-MDMKP represents a large class of important real-life problems, including portfolio selection, capital budgeting, and obnoxious and semi-obnoxious facility location problems. A different application is considered in [76] for the disjunctively constrained knapsack problem (DCKP), a variant of the standard knapsack problem with special disjunctive constraints. A disjunctive constraint is a couple of items for which only one item is packed.

### 18.3.1.2 Problems on Graphs

A reactive tabu search algorithm for Minimum Labeling Spanning Tree is considered in [39, 40], together with other meta-heuristics. The problem is as follows: Given a graph  $G$  with a color (label) assigned to each edge one looks for a spanning tree of  $G$  with the minimum number of different colors. The problem has several applications in telecommunication networks, electric networks, multi-modal transportation networks, among others, where one aims at ensuring connectivity by means of homogeneous connections.

The graph partitioning problem has been a test case for advanced local search heuristics starting at least from the seminal Kernighan and Lin paper [84], which proposes a variable-depth scheme. This is in fact a simple prohibition-based (tabu) scheme where swaps of nodes among the two sets of the partitions are applied and the just swapped nodes are kept fixed during a sequence of tentative moves in search of an improving chain. Greedy, Prohibition-based, and Reactive Search Optimization Heuristics for Graph Partitioning are proposed and compared in [11], Multilevel Reactive Tabu Search techniques, based on producing coarse versions of very large graphs are considered for Graph Partitioning in [10].

An RSO scheme is applied to the Maximum Clique Problem (MCP) in graphs in [19, 23]. A clique is a subset of nodes which are mutually interconnected, the problem is related to identifying densely interconnected communities and, in general, to clustering issues. A relaxed quasi-clique version of the problem where some edges may be missing is addressed in [38].

The work in [132] introduces a new algorithm that combines the stigmergic capabilities of Ant Colony Optimization (ACO) with local search heuristics to solve the maximum and maximum-weighted clique problem. The introduced Reactive Prohibition-based Ant Colony Optimization for MCP (RPACOMCP) complements the intelligent ant colony search with a prohibition-based diversification technique, where the amount of diversification is determined in an automated way through a feedback (history-sensitive) scheme.

In [111] the authors address the problem of computing a graph similarity measure which is generic in the sense that other well-known graph similarity measures can be viewed as special cases of it. They propose and compare two algorithms, an Ant Colony Optimization based algorithm and a Reactive Search Optimization, and show that they obtain complementary results: while the RSO technique achieves good solutions in shorter times, the proposed ACO method eventually attains a better quality.

A classification of methods to manage the prohibition period (Tabu tenure) in the literature is presented in [54] together with a new reactive Tabu tenure adaptation mechanism. The generic method is tested on the  $k$ -coloring problem.

A Reactive Tabu Search algorithm with variable clustering for the unicost Set Covering Problem (SCP) is proposed in [86]. Unicost SCPs arise in graph theory when one must select a minimum covering of edges by nodes or nodes by cliques. In addition, in many practical applications (crew scheduling, political redistricting, conservation biology, etc.) the relative variation in the weights may be small enough to warrant a unicost model.

### 18.3.1.3 Vehicle Routing Problems

A reactive tabu search version for the vehicle routing problem with time windows is designed in [44], while a version of the Vehicle Routing Problem with Backhauls (VRPB) is considered in [50, 104].

A heuristic approach based on a hybrid operation of RTS and Adaptive Memory Programming (AMP) is proposed in [91] to solve the VRPB. One is given a set of customers, some of which are linehauls (delivery points) and some are backhauls (collection points), a set of homogeneous vehicles and a depot. A distinguishing feature of this model is that all backhaul customers must be visited after all linehaul customers are served on each route. RTS is used with an escape mechanism which manipulates different neighborhood schemes in order to continuously balance intensification and diversification during the search process. The adaptive memory strategy takes the search back to the unexplored regions of the search space by maintaining a set of elite solutions and using them strategically with the RTS. The authors in [101] address the pickup and delivery problem with time windows using reactive tabu search.

A reactive VNS technique for VRPTW is also proposed in [35]. Vehicle routing with soft time windows and Erlang travel times is studied in [109].

### 18.3.1.4 Satisfiability and Related Problems

Maximum satisfiability is considered in [20, 21, 97], reactive Scaling and Probabilistic Smoothing (SAPS) in [124], constraint satisfaction in [102]. Reactive local search techniques for the Maximum  $k$ -Conjunctive Constraint Satisfaction Problem (MAX- $k$ -CCSP) in [22]. A worst-case analysis of tabu search as a function of the tabu list length for the MAX-TWO-SAT problem is presented in [97], with applications also to a reactive determination of the prohibition.

## 18.3.2 Neural Networks and VLSI Systems with Learning Capabilities

While derivative-based methods for training from examples have been used with success in many contexts (error back-propagation is an example in the field of neural networks), they are applicable only to differentiable performance functions and are not always appropriate in the presence of local minima. In addition, the calculation of derivatives is expensive and error-prone, especially if special-purpose VLSI hardware is used. A radically different approach is used in [29]: the task is transformed into a combinatorial optimization problem (the points of the search space are binary strings) and solved with the Reactive Search Optimization algorithm. To speed up the neighborhood evaluation phase a stochastic sampling of the neighborhood is adopted and a “smart” iterative scheme is used to compute the changes in the performance function caused by changing a single weight. RSO escapes rapidly from local minima, it is applicable to non-differentiable and even discontinuous functions

and it is very robust with respect to the choice of the initial configuration. In addition, by fine-tuning the number of bits for each parameter one can decrease the size of the search space, increase the expected generalization, and realize cost-effective VLSI.

Reactive Tabu Search in Semi-supervised Classification is proposed in [133]. With a linear kernel their RTS implementation can effectively find optimal global solutions for the primal Mixed Integer Programming Transductive Support Vector Machine (MIP-TSVM) with relatively large problem dimension.

In contrast to the exhaustive design of systems for pattern recognition, control, and vector quantization, an appealing possibility consists of specifying a general architecture, whose parameters are then tuned through Machine Learning (ML). ML becomes a combinatorial task if the parameters assume a discrete set of values: the RTS algorithm permits the training of these systems with low number of bits per weight, low computational accuracy , no local minima “trapping”, and limited sensitivity to the initial conditions [16, 17].

Special-purpose VLSI modules have been developed to be used as components of fully autonomous massively parallel systems for real-time adaptive applications. In contrast to many “emulation” approaches, the developed VLSI completely reflects the combinatorial structure used in the learning algorithms.

Applications considered are in the area of pattern recognition (Optical Character Recognition), event triggering in High Energy Physics [3], control of non-linear systems [28], and compression of EEG signals [24]. The first product was the TOTEM chip [16, 18] and more general special-purpose VLSI realizations are described in [3, 4]. A parallel neurochip for neural networks implementing the Reactive Tabu Search algorithm and application case studies are presented in [52]. A Fast Programmable Gate Array (FPGA) implementation of the TOTEM chip is presented in [6].

### ***18.3.3 Continuous Optimization***

A simple benchmark on a function with many suboptimal local minima is considered in [29], where a straightforward discretization of the domain is used. A novel algorithm for the global optimization of functions (C-RTS) is presented in [30], in which a combinatorial optimization method cooperates with a stochastic local minimizer. The combinatorial optimization component, based on Reactive Search Optimization, locates the most promising boxes, where starting points for the local minimizer are generated. In order to cover a wide spectrum of possible applications with no user intervention, the method is designed with adaptive mechanisms: in addition to the reactive adaptation of the prohibition period, the box size is adapted to the local structure of the function to be optimized (boxes are larger in “flat” regions, smaller in regions with a “rough” structure). An application of intelligent prohibition-based strategies to continuous optimization is presented in [43].

A Reactive Affine Shaker method for continuous optimization is studied in [36, 37]. The work presents an adaptive stochastic search algorithm for the optimization

of functions of continuous variables where the only hypothesis is the pointwise computability of the function. The main design criterion consists of the adaptation of a search region by an affine transformation which takes into account the local knowledge derived from trial points generated with uniform probability. Heuristic adaptation of the step size and direction allows the largest possible movement per function evaluation. The experimental results show that the proposed technique is, in spite of its simplicity, a promising building block to consider for the development of more complex optimization algorithms, particularly in those cases where the objective function evaluation is expensive.

A Gregarious Particle Swarm Optimizer (GPSO) is proposed in [106]. The particles (the different local searchers) adopt a reactive determination of the step size, based on feedback from the last iterations. This is in contrast to the basic particle swarm algorithm, in which no feedback is used for the self-tuning of algorithm parameters. The novel scheme presented, when tested on a benchmark for continuous optimization, besides generally improving the average optimal values found, reduces the computation effort.

### ***18.3.4 Real-World Applications***

Reactive search schemes have been applied to a significant number of “real-world” problems; this term refers to applications where domain-specific knowledge is required. Rather than defining classes with properties common to many problems, these applications aim at the “pointwise” solution of a specific issue with detailed modeling, therefore providing an essential benchmark for optimization techniques.

#### **18.3.4.1 Power Distribution Networks**

A real-world application in the area of electric power distribution, service restoration in distribution systems is studied in [125]. Fast optimal setting for voltage control equipment considering interconnection of distributed generators is proposed in [103], service restoration in distribution systems in [66], and distribution load transfer operation in [65].

#### **18.3.4.2 Industrial Production and Delivery**

A continuation of the previously mentioned work on VRPTW [44] is proposed in [108] to aid in the coordination and synchronization of the production and delivery of multi-product newspapers to bulk delivery locations. The problem is modeled as an open vehicle routing problem with time windows and zoning constraints. The methodology is applied to the newspaper production and distribution problem in a major metropolitan area.

In the field of industrial production planning, [58] studies applications of modern heuristic search methods to pattern sequencing problems. Flexible job-shop scheduling is studied in [41, 42]. The plant location problem is studied in [53]. The work in [59] is dedicated to solving the continuous flow-shop scheduling problem. Adaptive self-tuning neurocontrol is considered in [107]. The objective is to construct an adaptive control scheme for unknown time-dependent nonlinear plants.

#### 18.3.4.3 Telecommunication Networks

Various applications of RSO focussed on problems arising in the design and management of telecommunication networks. RSO for traffic grooming in optical WDM networks is considered in [12]. Optimal conformance test selection is studied in [51]. Conformance testing is used to increase the reliability of telecommunication applications. Locating hidden groups in communication networks is addressed in [93] by using hidden Markov models. A communication network is a collection of social groups that communicate via an underlying communication medium. In such a network, a hidden group may try to camouflage its communications among the typical communications of the network. The task of increasing Internet capacity is considered in [62]. The Multiple-choice Multi-dimensional Knapsack Problem (MMKP) with applications to service level agreements and multimedia distribution is studied in [76, 77], where the dynamic adaptation of the resource allocation model is considered for multi-session multimedia. High-quality solutions, reaching the optimum for several instances, are obtained through a reactive local search scheme. In the area of wireless and cellular communication networks, the work in [13] considers the optimal wireless access point placement for location-dependent services and the work in [61] proposes a tabu search heuristic for the dimensioning of 3G multi-service networks.

#### 18.3.4.4 Vehicle Routing and Dispatching

Real-time dispatch of trams in storage yards is studied in [131].

In the military sector, simple versions of Reactive Tabu Search are considered in [87] in a comparison of techniques dedicated to designing an unmanned aerial vehicle (UAV) routing system. Hierarchical Tabu Programming is used in [7] for finding underwater vehicle trajectories. Aerial reconnaissance simulations is the topic of [110]. The authors in [9] use an adaptive tabu search approach for solving the aerial fleet refueling problem.

#### 18.3.4.5 Industrial and Architectural Design

In the automotive sector, RSO is used in [71] for improving vehicle safety: a mixed reactive tabu search method is used to optimize the design of a vehicle B-pillar subjected to roof crush.

Reactive Tabu Search and sensor selection in active structural acoustic control problems are considered in [85].

The solution of the engineering roof truss design problem is discussed in [70]. An application of reactive tabu search for designing barrelled cylinders and domes of generalized elliptical profile is studied in [32]. The cylinders and domes are optimized for their buckling resistance when loaded by static external pressure by using a structural analysis tool.

#### 18.3.4.6 Biology

A reactive stochastic local search algorithm is used in [90] to solve the Genomic Median Problem (GMP), an optimization problem inspired by a biological issue. It aims at finding the chromosome organization of the common ancestor of multiple living species. It is formulated as the search for a genome that minimizes a rearrangement distance measure among given genomes. Additional applications in bio-informatics include, for example [120], which proposes an adaptive bin framework search method for a beta-sheet protein homopolymer model. A novel approach is studied based on the use of a bin framework for adaptively storing and retrieving promising locally optimal solutions. Each bin holds a set of conformations within a certain energy range and one uses an adaptive strategy for restarting a given search process with a conformation retrieved from these bins when the search stagnates. An adaptive mechanism chooses which conformations should be stored, based on the set of conformations already stored in memory, and biases choices when retrieving conformations from memory in order to overcome search stagnation. The energy and diversity thresholds for each bin are dynamically modified during the search process.

An adaptive meta-search method that alternates between two distinct modes of the search process at different levels is proposed in [118, 119] for protein folding. The high-level process ensures that unexplored promising parts of the search landscape are visited and the low-level search provides the thorough exploration of local neighborhoods. Multiple search processes are used in an intelligent way.

Finally, visual representation of data through clustering is considered in [47].

## References

1. Abramson, D., Dang, H., Krisnamoorthy, M.: Simulated annealing cooling schedules for the school timetabling problem. *Asia-Pac. J. Oper. Res.* **16**, 1–22 (1999). URL [citeseer.ist.psu.edu/article/abramson97simulated.html](http://citeseer.ist.psu.edu/article/abramson97simulated.html)
2. Aho, A.V., Hopcroft, J.E., Ullman, J.D.: Data Structures and Algorithms. Addison-Wesley (1983)
3. Anzellotti, G., Battiti, R., Lazzizzeri, I., Lee, P., Sartori, A., Soncini, G., Tecchioli, G., Zorat, A.: Totem: a highly parallel chip for triggering applications with inductive learning based on the reactive tabu search. In: AIHENP95. Pisa, Italy (1995)

4. Anzellotti, G., Battiti, R., Lazzizzera, I., Soncini, G., Zorat, A., Sartori, A., Tecchiolli, G., Lee, P.: Totem: a highly parallel chip for triggering applications with inductive learning based on the reactive tabu search. *Int. J. Mod. Phys. C* **6**(4), 555–560 (1995)
5. Arntzen, H., Hvattum, L.M., Lokketangen, A.: Adaptive memory search for multidemand multidimensional knapsack problems. *Comput. Oper. Res.* **33**(9), 2508–2525 (2006). DOI <http://dx.doi.org/10.1016/j.cor.2005.07.007>
6. Avogadro, M., Bera, M., Danese, G., Loporati, F., Spelgatti, A.: The Totem neurochip: an FPGA implementation. In: *Signal Processing and Information Technology, 2004. Proceedings of the Fourth IEEE International Symposium on*, pp. 461–464 (2004)
7. Balicki, J.: Hierarchical Tabu Programming for Finding the Underwater Vehicle Trajectory. *IJCSNS* **7**(11), 32 (2007)
8. Baluja, S., Barto, A., Boyan, K.B.J., Buntine, W., Carson, T., Caruana, R., Cook, D., Davies, S., Dean, T., et al.: Statistical Machine Learning for Large-Scale Optimization. *Neural Comput. Surv.* **3**, 1–58 (2000)
9. Barnes, J., Wiley, V., Moore, J., Ryer, D.: Solving the aerial fleet refueling problem using group theoretic tabu search. *Math. Comput. Model.* **39**, 617–640 (2004)
10. Battiti, R., Bertossi, A., Cappelletti, A.: Multilevel Reactive Tabu Search for Graph Partitioning. Preprint UTM **554** (1999)
11. Battiti, R., Bertossi, A.A.: Greedy, prohibition, and reactive heuristics for graph partitioning. *IEEE Trans. Comput.* **48**(4), 361–385 (1999)
12. Battiti, R., Brunato, M.: Reactive search for traffic grooming in WDM networks. In: S. Palazzo (ed.) *Evolutionary Trends of the Internet, IWDC2001, Taormina, Lecture Notes in Computer Science LNCS 2170*, pp. 56–66. Springer, Berlin/Heidelberg, Germany (2001)
13. Battiti, R., Brunato, M., Delai, A.: Optimal wireless access point placement for location-dependent services. Technical Report, University di Trento DIT-03-052 (2003)
14. Battiti, R., Brunato, M., Mascia, F.: *Reactive Search and Intelligent Optimization, Operations Research/Computer Science Interfaces*, vol. 45. Springer, Berlin/Heidelberg, Germany (2008)
15. Battiti, R., Campigotto, P.: Reinforcement learning and reactive search: an adaptive max-sat solver. In: Ghallab, N.F.M., Spyropoulos, C.D., Avouris N. (eds.) *Proceedings ECAI 08: 18th European Conference on Artificial Intelligence, Patras, Greece, 21–25 Jul 2008*. IOS Press, Amsterdam (2008)
16. Battiti, R., Lee, P., Sartori, A., Tecchiolli, G.: Combinatorial optimization for neural nets: Rts algorithm and silicon. Technical Report, Dept. of Mathematics, University of Trento, IT (1994). Preprint UTM 435
17. Battiti, R., Lee, P., Sartori, A., Tecchiolli, G.: Totem: A digital processor for neural networks and reactive tabu search. In: *Fourth International Conference on Microelectronics for Neural Networks and Fuzzy Systems, MICRONEURO 94*, pp. 17–25. IEEE Computer Society Press, Torino, Italy (1994). Preprint UTM 436-June 1994, Università di Trento, IT
18. Battiti, R., Lee, P., Sartori, A., Tecchiolli, G.: Special-purpose parallel architectures for high-performance machine learning. In: *High Performance Computing and Networking*. Milano, Italy (1995). Preprint UTM 445, December 1994, Università di Trento, IT
19. Battiti, R., Protasi, M.: Reactive local search for maximum clique. In: Italiano, G.F., Orlando S. (eds.) *Proceedings of the Workshop on Algorithm Engineering (WAE'97)*, Ca' Dolfin, Venice, Italy, pp. 74–82 (1997)
20. Battiti, R., Protasi, M.: Reactive search, a history-sensitive heuristic for MAX-SAT. *ACM Journal of Experimental Algorithms* **2**(ARTICLE 2) (1997). <http://www.jea.acm.org/>
21. Battiti, R., Protasi, M.: Solving MAX-SAT with non-oblivious functions and history-based heuristics. In: Du, D., Gu, J., Pardalos P.M. (eds.) *Satisfiability Problem: Theory and Applications*, no. 35 in *DIMACS: Series in Discrete Mathematics and Theoretical Computer Science*, pp. 649–667. American Mathematical Society, Association for Computing Machinery (1997)
22. Battiti, R., Protasi, M.: Reactive local search techniques for the maximum k-conjunctive constraint satisfaction problem (MAX-k-CCSP). *Discrete Appl. Math.* **96**, 3–27 (1999)

23. Battiti, R., Protasi, M.: Reactive local search for the maximum clique problem. *Algorithmica* **29**(4), 610–637 (2001)
24. Battiti, R., Sartori, A., Tecchiolli, G., Tonella, Zorat, A.: Neural compression: an integrated approach to eeg signals. In: Alspector, J., Goodman, R., Brown T.X. (eds.) International Workshop on Applications of Neural Networks to Telecommunications (IWANNT\*95), pp. 210–217. Stockholm, Sweden (1995)
25. Battiti, R., Tecchiolli, G.: Learning with first, second, and no derivatives: a case study in high energy physics. *Neurocomputing* **6**, 181–206 (1994)
26. Battiti, R., Tecchiolli, G.: The reactive tabu search. *ORSA J. Comput.* **6**(2), 126–140 (1994)
27. Battiti, R., Tecchiolli, G.: Simulated annealing and tabu search in the long run: a comparison on QAP tasks. *Comput. Math. Appl.* **28**(6), 1–8 (1994)
28. Battiti, R., Tecchiolli, G.: Local search with memory: Benchmarking rts. *Oper. Res. Spektrum* **17**(2/3), 67–86 (1995)
29. Battiti, R., Tecchiolli, G.: Training neural nets with the reactive tabu search. *IEEE Trans. Neural Netw.* **6**(5), 1185–1200 (1995)
30. Battiti, R., Tecchiolli, G.: The continuous reactive tabu search: blending combinatorial optimization and stochastic search for global optimization. *Ann. Oper. Res. – Metaheuristics in Comb. Optimization* **63**, 153–188 (1996)
31. Baxter, J.: Local optima avoidance in depot location. *J. Oper. Res. Soc.* **32**(9), 815–819 (1981)
32. Blachut, J.: Tabu search optimization of externally pressurized barrels and domes. *Eng. Optimization* **39**(8), 899–918 (2007)
33. Boyan, J., Moore, A.: Learning evaluation functions to improve optimization by local search. *J. Mach. Learn. Res.* **1**, 77–112 (2001)
34. Boyan, J.A., Moore, A.W.: Learning evaluation functions for global optimization and boolean satisfiability. In: Press A. (ed.) In: Proceedings of 15th National Conf. on Artificial Intelligence (AAAI), pp. 3–10 (1998)
35. Braysy, O.: A reactive variable neighborhood search for the vehicle-routing problem with time windows. *INFORMS J. COMPUT.* **15**(4), 347–368 (2003)
36. Brunato, M., Battiti, R.: RASH: A self-adaptive random search method. In: Cotta, C., Sevaux, M., Sørensen K. (eds.) *Adaptive and Multilevel Metaheuristics, Studies in Computational Intelligence*, vol. 136. Springer, Berlin/Heidelberg, Germany (2008)
37. Brunato, M., Battiti, R., Pasupuleti, S.: A memory-based rash optimizer. In: Geffner, A.F.R.H.H. (ed.) *Proceedings of AAAI-06 Workshop on Heuristic Search, Memory Based Heuristics and Their Applications*, pp. 45–51. Boston, MA. (2006). ISBN 978-1-57735-290-7
38. Brunato, M., Hoos, H., Battiti, R.: On effectively finding maximal quasi-cliques in graphs. In: Maniezzo, V., Battiti, R., Watson J.P. (eds.) *Proceedings 2nd Learning and Intelligent Optimization Workshop, LION 2, Trento, Italy, December 2007, LNCS*, vol. 5313. Springer, Berlin/Heidelberg, Germany (2008)
39. Cerulli, R., Fink, A., Gentili, M., Voss, S.: Metaheuristics comparison for the minimum labelling spanning tree problem. *The Next Wave on Computing, Optimization, and Decision Technologies*, pp. 93–106. Springer, New York (2005)
40. Cerulli, R., Fink, A., Gentili, M., Voß, S.: Extensions of the minimum labelling spanning tree problem. *J. Telecommun. Inf. Technol.* **4**, 39–45 (2006)
41. Chambers, J., Barnes, J.: New tabu search results for the job shop scheduling problem. The University of Texas, Austin, TX, Technical Report Series ORP96-06, Graduate Program in Operations Research and Industrial Engineering (1996)
42. Chambers, J., Barnes, J.: Reactive search for flexible job shop scheduling. Graduate program in Operations Research and Industrial Engineering, The University of Texas at Austin, Technical Report Series, ORP98-04 (1998)
43. Chelouah, R., Siarry, P.: Tabu search applied to global optimization. *Eur. J. Oper. Res.* **123**, 256–270 (2000)
44. Chiang, W., Russell, R.: A reactive tabu search metaheuristic for the vehicle routing problem with time windows. *INFORMS J. Comput.* **9**, 417–430 (1997)

45. Codenotti, B., Manzini, G., Margara, L., Resta, G.: Perturbation: An efficient technique for the solution of very large instances of the euclidean tsp. *INFORMS J. COMPUT.* **8**(2), 125–133 (1996)
46. Connolly, D.: An improved annealing scheme for the QAP. *Eur. J. Oper. Res.* **46**(1), 93–100 (1990)
47. Consoli, S., Darby-Dowman, K., Geleijnse, G., Korst, J., Pauws, S.: Metaheuristic approaches for the quartet method of hierarchical clustering. Technical Report, Brunel University, West London (2008)
48. Corana, A., Marchesi, M., Martini, C., Ridella, S.: Minimizing multimodal functions of continuous variables with the simulated annealing algorithm. *ACM Trans. Math. Softw.* **13**(3), 262–280 (1987). DOI <http://doi.acm.org/10.1145/29380.29864>
49. Cox, B.J.: Object Oriented Programming, an Evolutionary Approach. Addison-Wesley, Menlo Park, CA (1990)
50. Crispim, J., Brandao, J.: Reactive tabu search and variable neighborhood descent applied to the vehicle routing problem with backhauls. In: Proceedings of the 4th Metaheuristics International Conference, Porto, MIC, pp. 631–636 (2001)
51. Csöndes, T., Kotnyek, B., Zoltán Szabó, J.: Application of heuristic methods for conformance test selection. *Eur. J. Oper. Res.* **142**(1), 203–218 (2002)
52. Danese, G., De Lotto, I., Leporati, F., Quaglini, A., Ramat, S., Tecchiolli, G.: A parallel neurochip for neural networks implementing the reactive tabu search algorithm: application case studies. In: Parallel and Distributed Processing, 2001. Proceedings. Ninth Euromicro Workshop on, pp. 273–280 (2001)
53. Delmaire, H., Diaz, J., Fernandez, E., Ortega, M.: Reactive GRASP and Tabu Search based heuristics for the single source capacitated plant location problem. *INFOR* **37**, 194–225 (1999)
54. Devarenne, I., Mabed, H., Caminada, A.: Adaptive tabu tenure computation in local search. In: Proceedings 8th European Conference on Evolutionary Computation in Combinatorial Optimisation, Napoli, March 2008, Lecture Notes in Computer Science, vol. 4972, p. 1. Springer, Berlin/Heidelberg, Germany (2008)
55. Eiben, A.E., Horvath, M., Kowalczyk, W., Schut, M.C.: Reinforcement learning for online control of evolutionary algorithms. In: Brueckner, S., Hassas, S., Jelasity, M., Yamins, D. (eds.) Engineering Self-Organising Systems Conference - 4th International Workshop, ESOA 2006, Hakodate, Japan, May 9, 2006. LNAI, vol. 4335. Springer, Berlin/Heidelberg (2006)
56. Faigle, U., Kern, W.: Some convergence results for probabilistic tabu search. *ORSA J. Comput.* **4**(1), 32–37 (1992)
57. Fescioglu-Unver, N., Kokar, M.: Application of Self Controlling Software Approach to Reactive Tabu Search. In: Self-Adaptive and Self-Organizing Systems, 2008. SASO'08. Second IEEE International Conference on, pp. 297–305 (2008)
58. Fink, A., Voß, S.: Applications of modern heuristic search methods to pattern sequencing problems. *Comput. Oper. Res.* **26**(1), 17–34 (1999)
59. Fink, A., Voß, S.: Solving the continuous flow-shop scheduling problem by metaheuristics. *Eur. J. Oper. Res.* **151**(2), 400–414 (2003)
60. Fleischer, M.A.: Cybernetic optimization by simulated annealing: Accelerating convergence by parallel processing and probabilistic feedback control. *J. Heuristics* **1**(2), 225–246 (1996)
61. Fortin, A., Hail, N., Jaumard, B.: A tabu search heuristic for the dimensioning of 3G multi-service networks. *Wireless Communications and Networking, 2003. WCNC 2003*, vol. 3, pp.1439–1447. IEE Computer Society, Location - Los Alamitos, CA (2003)
62. Fortz, B., Thorup, M.: Increasing internet capacity using local search. *Comput. Optimization Appl.* **29**(1), 13–48 (2004)
63. Frank, J.: Weighting for godot: Learning heuristics for GSAT. In: Proceedings of the National Conference on Artificial Intelligence, vol. 13, pp. 338–343. Wiley, USA (1996)
64. Frank, J.: Learning short-term weights for GSAT. In: Proceedings International Joint Conference on Artificial Intelligence, vol. 15, pp. 384–391. Lawrence Erlbaum, USA (1997)

65. Fukuyama, Y.: Reactive tabu search for distribution load transfer operation. In: Power Engineering Society Winter Meeting, 2000. vol. 2. IEEE Computer Society, Los Alamitos, CA (2000)
66. Genji, T., Oomori, T., Miyazato, K., Hayashi, N., Fukuyama, Y., Co, K.: Service Restoration in Distribution Systems Aiming Higher Utilization Rate of Feeders. In: Proceedings of the Fifth Metaheuristics International Conference (MIC2003), Kyoto, Japan (2003)
67. Gent, I., Walsh, T.: Towards an understanding of hill-climbing procedures for sat. In: Proceedings of the Eleventh National Conference on Artificial Intelligence, pp. 28–33. AAAI Press/The MIT Press, Cambridge, MA (1993)
68. Glover, F.: Tabu search—part i. *ORSA J. Comput.* **1**(3), 190–260 (1989)
69. Glover, F.: Tabu search—part ii. *ORSA J. Comput.* **2**(1), 4–32 (1990)
70. Hamza, K., Mahmoud, H., Saitou, K.: Design optimization of N-shaped roof trusses using reactive taboo search. *Appl. Soft Comput.* **3**(3), 221–235 (2003)
71. Hamza, K., Saitou, K., Nassem, A.: Design optimization of a vehicle b-pillar subjected to roof crush using mixed reactive taboo search. pp. 1–9. Chicago, Illinois (2003)
72. Hansen, N.M.P.: Variable neighborhood search. *Comput. Oper. Res.* **24**(11), 1097–1100 (1997)
73. Hansen, P., Jaumard, B.: Algorithms for the maximum satisfiability problem. *Comput.* **44**, 279–303 (1990)
74. Hansen, P., Mladenovic, N.: Variable neighborhood search. In: Burke, E., Kendall G. (eds.) *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques*, pp. 211–238. Springer, Berlin/Heidelberg, Germany (2005)
75. Hansmann, U.H.E.: Simulated annealing with tsallis weights a numerical comparison. *Physica A: Stat. Theor. Phys.* **242**(1–2), 250–257 (1997). DOI: 10.1016/S0378-4371(97)00203-3
76. Hifi, M., Michrafy, M.: A reactive local search-based algorithm for the disjunctively constrained knapsack problem. *J. Oper. Res. Soc.* **57**(6), 718–726 (2006)
77. Hifi, M., Michrafy, M., Sbihi, A.: A Reactive Local Search-Based Algorithm for the Multiple-Choice Multi-Dimensional Knapsack Problem. *Comput. Optimization. Appl.* **33**(2), 271–285 (2006)
78. Hu, B., Raidl, G.R.: Variable neighborhood descent with self-adaptive neighborhood-ordering. In: Cotta, C., Fernandez, A.J., Gallardo J.E. (eds.) *Proceedings of the 7th EU/MEEeting on Adaptive, Self-Adaptive, and Multi-Level Metaheuristics*, Malaga, Spain (2006)
79. Hutter, F., Babic, D., Hoos, H.H., Hu, A.J.: Boosting verification by automatic tuning of decision procedures. In: Baumgartner, J., Sheeran, M. (eds.) *Proceedings of Formal Methods in Computer Aided Design (FMCAD'07)*, pp. 27–34. IEEE Computer Society, Los Alamitos, CA (2006)
80. Hutter, F., Hamadi, Y., Hoos, H., Leyton-Brown, K.: Performance prediction and automated tuning of randomized and parametric algorithms. In: *Proceedings of the 12th International Conference on Principles and Practice of Constraint Programming (CP 2006)*. Springer, Berlin/Heidelberg, Germany (2006)
81. Hutter, F., Hoos, H., Stutzle, T.: Automatic algorithm configuration based on local search. In: *Proceedings of the National Conference on Artificial Intelligence*, vol. 22, p. 1152. Menlo Park, CA; Cambridge, MA 1999, AAAI Press MIT Press London (2007)
82. Ingber, L.: Very fast simulated re-annealing. *Math. Comput. Model.* **12**(8), 967–973 (1989)
83. Ishtaiwi, A., Thornton, J.R., A. Anbulagan, S., Pham, D.N.: Adaptive clause weight redistribution. In: *Proceedings of the 12th International Conference on the Principles and Practice of Constraint Programming, CP-2006*, Nantes, France, pp. 229–243 (2006)
84. Kernighan, B., Lin, S.: An efficient heuristic procedure for partitioning graphs. *Bell Syst. Tech. J.* **49**, 291–307 (1970)
85. Kincaid, R., Laba, K.: Reactive Tabu Search and sensor selection in active structural acoustic control problems. *J. Heuristics* **4**(3), 199–220 (1998)
86. Kinney, G., Barnes, J., Colletti, B.: A reactive Tabu Search algorithm with variable clustering for the Unicost Set Covering Problem. *Int. J. Oper. Res.* **2**(2), 156–172 (2007)

87. Kinney Jr, G., Hill, R., Moore, J.: Devising a quick-running heuristic for an unmanned aerial vehicle (UAV) routing system. *J. Oper. Res. Soc.* **56**, 776–786 (2005)
88. Kirkpatrick, S., Jr., C.D.G., Vecchi, M.P.: Optimization by simulated annealing. *Science* **220**, 671–680 (1983)
89. Laarhoven, P.J.M., Aarts, E.H.L. (eds.): *Simulated annealing: theory and applications*. Kluwer, Norwell, MA, USA (1987)
90. Lenne, R., Solnon, C., Stutzle, T., Tannier, E., Birattari, M.: Reactive stochastic local search algorithms for the genomic median problem. *Lecture Notes in Computer Science* **4972**, 266. Springer, Berlin/Heidelberg (2008)
91. Login, A., Areas, S.: Reactive tabu adaptive memory programming search for the vehicle routing problem with backhauls. *J. Oper. Res. Soc.* **58**, 1630–1641 (2007)
92. Lourenco, H.: Job-shop scheduling: computational study of local search and large-step optimization methods. *Euro. J. Oper. Res.* **83**, 347–364 (1995)
93. Magdon-Ismail, M., Goldberg, M., Wallace, W., Siebecker, D.: Locating hidden groups in communication networks using hidden markov models. *Lecture Notes in Computer Science*, vol. 2665, pp. 126–137. Springer, Berlin/Heidelberg (2003)
94. Martin, O., Otto, S.W., Felten, E.W.: Large-step Markov chains for the traveling salesman problem. *Complex Syst.* **5**:3, 299 (1991)
95. Martin, O., Otto, S.W., Felten, E.W.: Large-step Markov chains for the tsp incorporating local search heuristics. *Oper. Res. Lett.* **11**, 219–224 (1992)
96. Martin, O.C., Otto, S.W.: Combining simulated annealing with local search heuristics. *Ann. of Oper. Res.* **63**, 57–76 (1996)
97. Mastrolilli, M., Gambardella, L.: MAX-2-SAT: How good is tabu search in the worst-case? In: *Proceedings of the National Conference on Artificial Intelligence*, pp. 173–178. Menlo Park, CA; Cambridge, MA 1999. AAAI Press MIT Press, London (2004)
98. Morris, P.: The breakout method for escaping from local minima. In: *Proceedings of the National Conference on Artificial Intelligence*, vol. 11, p. 40. Wiley, USA (1993)
99. Nahar, S., Sahni, S., Shragowitz, E.: Experiments with simulated annealing. In: *DAC '85: Proceedings of the 22nd ACM/IEEE conference on Design automation*, pp. 748–752. ACM Press, New York, NY, USA (1985). DOI <http://doi.acm.org/10.1145/317825.317977>
100. Nahar, S., Sahni, S., Shragowitz, E.: Simulated annealing and combinatorial optimization. In: *DAC '86: Proceedings of the 23rd ACM/IEEE conference on Design automation*, pp. 293–299. IEEE Press, Piscataway, NJ, USA (1986)
101. Nanry, W., Wesley Barnes, J.: Solving the pickup and delivery problem with time windows using reactive tabu search. *Transportation Res. Part B* **34**(2), 107–121 (2000)
102. Nonobe, K., Ibaraki, T.: A tabu search approach for the constraint satisfaction problem as a general problem solver. *Euro. J. Oper. Res.* **106**, 599–623 (1998)
103. Oomori, T., Genji, T., Yura, T., Takayama, S., Watanabe, T., Fukuyama, Y., Center, T., Inc, K., Hyogo, J.: Fast optimal setting for voltage control equipment considering interconnection of distributed generators. In: *Transmission and Distribution Conference and Exhibition 2002: Asia Pacific*. IEEE/PES, vol. 2 (2002)
104. Osman, I., Wassan, N.: A reactive tabu search meta-heuristic for the vehicle routing problem with back-hauls. *J. Scheduling* **5**(4), 263–285 (2002)
105. Osman, I.H.: Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Ann. Oper. Res.* **41**(1–4), 421–451 (1993)
106. Pasupuleti, S., Battiti, R.: The gregarious particle swarm optimizer (G-PSO). In: *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, pp. 67–74. ACM New York, NY, USA (2006)
107. Potocnik, P., Grabec, I.: Adaptive self-tuning neurocontrol. *Math. Comput. Simulation* **51** (3–4), 201–207 (2000)
108. Russell, R., Chiang, W., Zepeda, D.: Integrating multi-product production and distribution in newspaper logistics. *Comput. Oper. Res.* **35**(5), 1576–1588 (2008)
109. Russell, R., Urban, T.: Vehicle routing with soft time windows and Erlang travel times. *J. Oper. Res. Soc.* (2007)

110. Ryan, J., Bailey, T., Moore, J., Carlton, W.: Reactive tabu search in unmanned aerial reconnaissance simulations. Proceedings of the 30th conference on Winter simulation, pp. 873–880 (1998)
111. Sammoud, O., Sorlin, S., Solnon, C., Ghédira, K.: A comparative study of ant colony optimization and reactive search for graph matching problems. In: Gottlieb, J., Raidl G.R. (eds.) Evolutionary Computation in Combinatorial Optimization – EvoCOP 2006, LNCS, vol. 3906, pp. 230–242. Springer, Budapest (2006)
112. Schuurmans, D., Southery, F., Holte, R.: The exponentiated subgradient algorithm for heuristic boolean programming. In: Proceedings of the International Joint Conference on Artificial Intelligence, vol. 17, pp. 334–341. Lawrence Erlbaum, USA (2001)
113. Selman, B., Kautz, H.: Domain-independent extensions to GSAT: solving large structured satisfiability problems. In: Proceedings of IJCAI-93, pp. 290–295 (1993)
114. Selman, B., Kautz, H.: An empirical study of greedy local search for satisfiability testing. In: Proceedings of the Eleventh National Conference on Artificial Intelligence (AAAI-93). Washington, D.C. (1993)
115. Selman, B., Kautz, H., Cohen, B.: Noise strategies for improving local search. In: Proceedings of the National Conference on Artificial Intelligence, vol. 12. Wiley, USA (1994)
116. Selman, B., Kautz, H., Cohen, B.: Local search strategies for satisfiability testing. In: Trick, M., Johnson D.S. (eds.) Proceedings of the Second DIMACS Algorithm Implementation Challenge on Cliques, Coloring and Satisfiability, no. 26 in DIMACS Series on Discrete Mathematics and Theoretical Computer Science, pp. 521–531 (1996)
117. Selman, B., Levesque, H., Mitchell, D.: A new method for solving hard satisfiability problems. In: Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92), pp. 440–446. San Jose, CA (1992)
118. Shmygelska, A.: Novel Heuristic Search Methods for Protein Folding and Identification of Folding Pathways. Ph.D. thesis, The University of British Columbia (2006)
119. Shmygelska, A.: An extremal optimization search method for the protein folding problem: the go-model example. In: Proceedings of the 2007 GECCO Conference Companion on Genetic and Evolutionary Computation, pp. 2572–2579. ACM Press, New York, NY, USA (2007)
120. Shmygelska, A., Hoos, H.: An adaptive bin framework search method for a beta-sheet protein homopolymer model. BMC Bioinform. **8**(1), 136 (2007)
121. Steiglitz, K., Weiner, P.: Algorithms for computer solution of the traveling salesman problem. In: Proceedings of the Sixth Allerton Conference on Circuit and System Theory, Urbana, Illinois, pp. 814–821. IEEE, New York (1968)
122. Taillard, E.: Robust taboo search for the quadratic assignment problem. Parallel Comput. **17**, 443–455 (1991)
123. Tompkins, D., Hoos, H.: Warped landscapes and random acts of SAT solving. Proceedings of the Eighth International Symposium on Artificial Intelligence and Mathematics (ISAIM-04) (2004)
124. Tompkins, F.H.D., Hoos, H.: Scaling and probabilistic smoothing: efficient dynamic local search for sat. In: Proceedings Principles and Practice of Constraint Programming—CP 2002 : 8th International Conference, CP 2002, Ithaca, NY, USA, September 9–13, LNCS, vol. 2470, pp. 233–248. Springer, Berlin/Heidelberg, Germany (2002)
125. Toune, S., Fudo, H., Genji, T., Fukuyama, Y., Nakanishi, Y.: Comparative study of modern heuristic algorithms to service restoration in distribution systems. IEEE Trans. Power Deliv. **17**(1), 173–181 (2002)
126. Vossen, T., Verhoeven, M., ten Eikelder, H., Aarts, E.: A quantitative analysis of iterated local search. Computing Science Reports 95/06, Department of Computing Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands (1995)
127. Voudouris, C., Tsang, E.: Partial constraint satisfaction problems and guided local search. In: Proceedings of 2nd International Conference on Practical Application of Constraint Technology (PACT 96), London, pp. 337–356 (1996)

128. Voudouris, C., Tsang, E.: Guided local search and its application to the traveling salesman problem. *Eur. J. Oper. Res.* **113**, 469–499 (1999)
129. Wah, B., Wu, Z.: Penalty formulations and trap-avoidance strategies for solving hard satisfiability problems. *J. Comput. Sci. Tech.* **20**(1), 3–17 (2005)
130. White, S.: Concepts of scale in simulated annealing. In: AIP Conference Proceedings, vol. 122, pp. 261–270 (1984)
131. Winter, T., Zimmermann, U.: Real-time dispatch of trams in storage yards. *Ann. Oper. Res.* **96**, 287–315 (2000). URL <http://citeseer.ist.psu.edu/winter00realtime.html>
132. Youssef, S., Elliman, D.: Reactive prohibition-based ant colony optimization (rpaco): a new parallel architecture for constrained clique sub-graphs. In: Proceedings of the 16th IEEE International Conference on Tools with Artificial Intelligence, pp. 63–71. IEEE Computer Society, Washington, DC, USA (2004)
133. Zennaki, M., Ech-cherif, A., Lamirel, J.: Using reactive tabu search in semi-supervised classification. In: Tools with Artificial Intelligence, 2007. ICTAI 2007. 19th IEEE International Conference, Patras, Greece, vol. 2. IEEE Computer Society, Los Alamitos, CA (2007)



# Chapter 19

## Stochastic Search in Metaheuristics

Walter J. Gutjahr

**Abstract** Stochastic search is a key mechanism underlying many metaheuristics. The chapter starts with the presentation of a general framework algorithm in the form of a stochastic search process that contains a large variety of familiar metaheuristic techniques as special cases. Based on this unified view, questions concerning convergence and runtime are discussed on the level of a theoretical analysis. Concrete examples from diverse metaheuristic fields are given. In connection with runtime results, important topics as instance difficulty, phase transitions, parameter choice, No-Free-Lunch theorems, or fitness landscape analysis are addressed. Furthermore, a short sketch of the theory of black-box optimization is given, and generalizations of results to stochastic search under noise are outlined.

### 19.1 Introduction

The aim of this chapter is to present a unified view of *stochastic search* which is used as a core mechanism in many metaheuristics. Not every metaheuristic applies a probabilistic mechanism to organize the exploration of the search space; there are, e.g., deterministic versions of Tabu Search. Interestingly enough, however, the incorporation of “random” (or more precisely, pseudo-random) steps into the algorithmic design is rather the usual than the exceptional case in the field of metaheuristics. Thus, it makes sense to have a closer look at this feature.

One would expect that all metaheuristics that perform stochastic search have some properties in common. Admittedly, at the moment, we are still far away from a general theory containing every stochastic metaheuristic as a special case. Nevertheless, some observations are available that are not restricted to a particular

---

Walter J. Gutjahr  
University of Vienna, Universitaetsstrasse 5/9, A-1010, Vienna, Austria  
e-mail: walter.gutjahr@univie.ac.at

metaheuristic algorithm, but have been made, possibly in different appearance, for several seemingly unrelated algorithms.

The emphasis of this chapter is on results that lead to a deeper understanding of principles and properties common to more than one stochastic metaheuristic. Because of this goal, we concentrate on *theoretical* results, which can be rigorous or (at least) precise, where “rigorous” is understood in a mathematical sense, and “precise” means that some form of analytic derivation (although not necessarily a rigorous one) is used for predicting numerical experimental outcomes. It is clear that experimental results are at least as important—presumably even more important. However, they usually contribute to a smaller degree to a unifying understanding, so we shall not focus on them here.

The chapter is organized as follows: In Section 19.2, we develop a common formal framework capturing the essential features of most stochastic metaheuristics, and we shortly address the motivation for applying *stochastic* search in metaheuristic algorithms. Sections 19.3 and 19.4 are devoted to convergence results and to results dealing with required optimization time, respectively. The practically important issue of parameter choice in metaheuristics is briefly outlined in Section 19.5. Section 19.6 discusses “No-Free-Lunch” theorems and their implications for stochastic search, in particular the desirability of a problem-specific fitness landscape analysis. Some techniques for the last are outlined in Section 19.7. The purest form of stochastic search algorithms are (stochastic) black-box optimizers, which are discussed in Section 19.8. Section 19.9 outlines an important special application area of metaheuristics, namely optimization under uncertainty or noise. Section 19.10, finally, concludes the chapter.

## 19.2 General Framework

The aim of the stochastic search algorithms investigated in this chapter is the exact or approximative solution of combinatorial optimization (CO) problems of the form

$$f(x) \rightarrow \min \text{ such that } x \in S, \quad (19.1)$$

where  $S$  is a finite search space,  $f$  is a real-valued function called *objective function*, and “min” can be replaced by “max.” The function  $f$  is also called *cost function* (if to be minimized) or *fitness function* (if to be maximized). We consider iterative algorithms  $A$  of the following general type: In iteration  $t$ , algorithm  $A$  uses a memory  $M_t$  and a list  $L_t$  of solutions  $x_i \in S$ . The list  $L_t$  contains new “trial points” for the optimization. The algorithm proceeds as follows:

1. Initialize  $M_1$  according to some rule.
2. In iteration  $t = 1, 2, \dots$ , until some stopping criterion is satisfied,
  - a. determine the list  $L_t$  as a function  $g(M_t, z_t)$  of  $M_t$  and of a random influence  $z_t$ ;
  - b. determine the objective function values  $f(x_i)$  of all  $x_i \in L_t$  and form a list  $L_t^+$  containing the pairs  $(x_i, f(x_i))$ ;

- c. determine the new memory content  $M_{t+1}$  as a function  $h(M_t, L_t^+, z'_t)$  of the current  $M_t$ , of the list of solution-value pairs  $L_t^+$ , and of a random influence  $z'_t$ .

The currently proposed (approximate) solution  $x_t^{\text{curr}}$  in iteration  $t$  results as some function of  $(M_t, L_t^+)$  specified by  $A$ . Also the stopping criterion defined by  $A$  depends on  $(M_t, L_t^+)$ .

In this formalism, one can imagine  $z_t$  and  $z'_t$  as vectors of (pseudo-)random numbers that are used by the stochastic algorithm. The function  $g(M_t, z_t)$  specifies, for a given memory  $M_t$ , a probability distribution for the list of new search points; the function  $h(M_t, L_t^+, z'_t)$  specifies, to given memory  $M_t$  and current list  $L_t^+$  of solution-value pairs, a probability distribution for the new content of the memory. If the functions  $g$  and  $h$  are independent of  $z_t$  resp.  $z'_t$ , we obtain the special case of a *deterministic* search algorithm.

The generic algorithm above, which is an extension of the generic black-box optimizer presented in [12] (discussed in Section 19.8), covers most—if not all—stochastic metaheuristics. We shall outline this by giving two special examples:

- *Simulated Annealing (SA)*: A neighborhood structure on  $S$  is used.  $M_t$  consists of a single element, the current search point  $x$ . Also  $L_t$  consists of a single element, the currently investigated neighbor solution  $y$  to  $x$ . To determine  $L_t$  from  $M_t$ , choose a random neighbor  $y$  to the element  $x$  in  $M_t$ . To update  $M_t$  to  $M_{t+1}$ , decide by the stochastic acceptance rule used in SA whether  $y$  is accepted or not. If yes,  $M_{t+1}$  contains  $y$ , otherwise it contains  $x$ .
- *Canonical Genetic Algorithm (GA)*:  $M_t$  consists of  $k$  solutions, and  $L_t$  also consists of  $k$  solutions. To determine  $L_t$  from  $M_t$ , apply the operators *mutation* and *crossover* to the solutions in  $M_t$ . This yields  $L_t$ . To update  $M_t$  to  $M_{t+1}$ , apply fitness-proportional *selection* to the population contained in  $L_t$ , using the corresponding objective function values. The result gives  $M_{t+1}$ .

In principle, the functions  $g$  and  $h$  may use any information on the problem instance. The important special case where  $g$  and  $h$  are only allowed to use the knowledge of the search space  $S$  and of the problem type, but not of the concrete problem instance, is denoted as *black-box optimization* and will be dealt with in Section 19.8.

An important observation is that by construction, the “states”  $(M_t, L_t^+)$  visited during the execution of the algorithm form a *Markov process* in discrete time<sup>1</sup>: The distribution of the next state  $(M_{t+1}, L_{t+1}^+)$  only depends on the current state  $(M_t, L_t^+)$ . Considering the objective function  $f$  as given, already  $(M_t)$  ( $t = 1, 2, \dots$ ) can be seen as a Markov process, since (via  $L_t^+$ , which results from  $M_t$ ) the distribution of  $M_{t+1}$  only depends on  $M_t$ . This allows the application of Markov process theory to the analysis of stochastic search algorithms.

We may use the described algorithmic framework for giving a rough classification of several stochastic metaheuristics:

---

<sup>1</sup> Since  $g$  and  $h$  do not depend on the iteration counter  $t$ , the Markov process is homogeneous. Dependence on  $t$  can easily be modeled by adding  $t$  as a component to the memory  $M_t$ .

1. *Stochastic Local Search Algorithms*: Examples are Iterated Local Search (ILS), Simulated Annealing (SA), Generalized Hillclimbers (GHCs), or Variable Neighborhood Search (VNS).  $M_t$  contains a small, fixed number of solutions (e.g., incumbent solution, current search point, and current neighbor) derived by using a neighborhood structure on  $S$ .
2. *Population-Based Stochastic Search Algorithms*: Examples are GAs and basic forms of Estimation-of-Distribution Algorithms (EDAs).  $M_t$  contains a “population” of solutions. The size of this population is a parameter of the algorithm.
3. *Model-Based Stochastic Search Algorithms*: The concept of model-based search has been introduced by Zlochin et al. [89]. This group of metaheuristics contains Ant Colony Optimization (ACO), some more elaborated forms of EDAs, or Cross-Entropy Optimization. Here,  $M_t$  consists of a vector of real-valued parameters, e.g., a pheromone vector in ACO, sometimes also of additional information.

Although metaheuristics as *Particle Swarm Optimization* (PSO) or some variants of *Evolution Strategies* (ES) do not deal with CO problems, but rather with *continuous* search spaces  $S$  instead, these metaheuristics can be used for CO problems as well by means of suitable problem encodings. For example, the Binary PSO algorithm proposed by Kennedy and Eberhart [51] maintains vectors interpreted as positions, best positions, and velocities of “particles,” from which discrete solutions can be derived by a probabilistic mechanism. In the classification above, this leads us to the model-based class with  $M_t$  containing a list of vectors of real numbers.

We close this section with the question of the general motivation for introducing *stochastic* elements (the random variables  $z_t$  and  $z'_t$  above) into a metaheuristic. Perhaps the simplest reason is that care must be taken to prevent a search algorithm from cycling through a small portion of the search space. Let us look at a simple example. Assume that we perform search in the set  $S = \{0, 1\}^n$  of binary strings of length  $n$ , with some cost function  $f$  on  $S$ . For simplicity, let us suppose that all occurring cost values are different from each other. Our algorithm always stores the current solution  $x$  as well as the solution  $w$  visited before  $x$  has been visited, and it iteratively moves from the current  $x$  to the lowest cost neighbor solution  $y$  of  $x$  different from  $w$ , where  $x$  and  $y$  are called neighbors if they differ exactly in one bit position. This deterministic search algorithm is able to quickly find a locally optimal solution (i.e., a solution that does not have a neighbor with lower cost), but neither is it able to stop at a local optimum  $x_{\text{loc}}$  nor does it typically leave the neighborhood of some suboptimal  $x_{\text{loc}}$  in order to continue the search for the global optimum. Of course, this undesirable behavior can be avoided by increasing our “tabu list” (consisting only of  $w$  in the naive algorithm above), but this comes at the price of increased computational cost. An alternative way to give the search process the freedom to leave local optima is to allow *random* moves from a solution point to a neighbor. Whenever we choose this alternative, it is much easier to ensure that no point in the solution space is excluded from the search in advance.

### 19.3 Convergence Results

The search process  $(M_t, L_t^+)$  is only helpful if it leads us to an optimal solution of Equation (19.1) or, at least, to a good approximation to an optimal solution. Ideally, the current solution  $x_t^{\text{curr}}$  derived from the state at time  $t$  becomes an element of the set  $S^*$  of optimal solutions in some iteration  $t_1$  and remains unchanged in subsequent iterations. This behavior is denoted as convergence to the optimum. Since we consider *stochastic* search algorithms, the definition of convergence has to be modified. In probability theory, there are several different notions for the convergence of a stochastic process. One of the most natural in our context is *convergence in probability*: A stochastic search algorithm  $A$  converges to the optimum in probability, if the probability of the event  $x_t^{\text{curr}} \in S^*$  converges (in the mathematical sense of the word) to unity as  $t \rightarrow \infty$ .

Convergence to the optimum in probability can be achieved easily even by simple stochastic search algorithms: Consider the (usually very inefficient) *random search* algorithm, where  $L_t$  consists in each iteration of a single solution  $x_t$  that is chosen at random from  $S$  according to some fixed distribution *independently* of  $M_t$  (and hence of the previous iterations). Let  $M_t$  contain the *best-so-far* solution  $x_t^{\text{bsf}}$  encountered up to iteration  $t - 1$ : The variable  $x_t^{\text{bsf}}$  is initialized arbitrarily for  $t = 1$ , and in each iteration where  $f(x_t)$  turns out as better than  $f(x_t^{\text{bsf}})$ ,  $x_t^{\text{bsf}}$  is set to  $x_t$ . If for each iteration  $t$ , we choose the currently proposed solution  $x_t^{\text{curr}}$  as the best-so-far solution  $x_t^{\text{bsf}}$ , random search converges to the optimum in probability. However, the runtime until hitting an optimal solution may be huge.

Interestingly, some more efficient algorithms (from a practical point of view) do *not* share the mentioned convergence property. For example, Rudolph [68] showed that the canonical GA, as described in the previous section, with  $x_t^{\text{curr}}$  defined as the best element of the current generation, *never* converges to the optimum in probability; this is simply due to the fact that by possible mutations, the probability of the event that the current population does not contain an element from  $S^*$  has always a strictly positive lower bound. By adding the “elite” solution  $x_t^{\text{bsf}}$  as an additional component to the memory  $M_t$ , the algorithm can be made convergent to the optimum in probability.<sup>2</sup>

For a stochastic search algorithm  $A$ , it would be desirable that not only the probability of  $x_t^{\text{curr}} \in S^*$  converges to one but exploitation of the search experience increases the average fitness of the sample points, i.e., of the elements of  $L_t$ , which is not the case for random search. If for an algorithm  $A$ , that part of the memory  $M_t$  that is responsible for the generation of the list  $L_t$  of sample points converges to some state supporting only optimal (or at least good) solutions, one can expect that the quality of the sample points improves during the process, such that the search algorithm will arrive at the optimum faster than random search.

---

<sup>2</sup> Elitism as a mechanism ensuring convergence of a GA has already been analyzed in [39], which appears to be the first paper on GA convergence.

Convergence results of the last kind are harder to show (and require stricter conditions on algorithms and parameter choices), but there exist such results in the literature for several metaheuristics. The first of them were derived for SA. For SA with a logarithmic cooling scheme, Hajek [38] gave necessary and sufficient conditions for the current search point  $x_t$  (the solution contained in  $M_t$ ) to converge in probability to  $S^*$ . Contrary to the best-so-far solution  $x_t^{\text{bsf}}$  which does not influence the process itself, the current search point  $x_t$  defines the candidates for the next sample point and thus determines the distribution of  $L_t$ . If  $x_t$  gradually focuses more and more on promising regions of the search space instead of doing “blind” random search (as in the first, high-temperature phase of SA), the chance of detecting the global optimum is increased compared to the random search algorithm. Therefore, convergence of  $x_t$  is more meaningful than convergence of  $x_t^{\text{bsf}}$  only.

In the ACO case, the “sample-generating” part of the memory  $M_t$  consists of the vector  $\tau_t$  of pheromone values determining the distribution of the solutions to be sampled in the current iteration. For algorithms of the MAX-MIN-Ant-System type developed by Stützle and Hoos [74] using “elitism” (i.e., incorporating also  $x_t^{\text{bsf}}$  into  $M_t$ ), conditions have been given in [29] and in [72] ensuring that not only  $x_t^{\text{bsf}}$  converges to the optimum but also  $\tau_t$  converges to a limiting vector that only allows the generation of an optimal solution. A related result for Cross-Entropy Optimization was shown by Margolin [54].

What have these results for different metaheuristics in common? Typically, when proving a “strong” form of convergence for a stochastic search algorithm in the just-mentioned sense, the parametrization of the algorithm has to be chosen in such a way that a proper balance between *exploration* and *exploitation* is preserved: When the emphasis is too much on the exploration pole, random search-type behavior results, and the sample-generating part of the memory  $M_t$  does not converge at all. On the other hand, when exploitation is emphasized too much, one does obtain convergence, but it is usually “premature” convergence to a suboptimal solution. By keeping the balance, convergence is still ensured, but slowed down to allow the detection of a global optimum. The concrete form of the exploration-exploitation trade-off depends on the algorithm under consideration. For example, for SA, high values of the temperature parameter favor exploration, low values favor exploitation.

In the context of some stochastic metaheuristics, it has turned out as convenient to approach the question of convergence via a *system dynamics* approach. For example, for PSO, Trelea [78] identifies *attractors*, i.e., stable fixed points of a dynamic process concretizing our generic  $(M_t, L_t^+)$  dynamics to the PSO case. In the case of convergence, only attractors can be limiting points. In [78], also the exploration–exploitation trade-off and its connection to parameter choice are explicitly addressed.

A very small selection of convergence results for stochastic metaheuristics have been mentioned in this section. For some other results, see, e.g., [48, 79] (GHCs), [26, 27] (EDAs), [28, 75] (ACO), or [35] (VNS).

## 19.4 Runtime Results

From the viewpoint of applications, the question whether and in which sense a stochastic search algorithm  $A$  converges is less relevant than the question what amount of computation times  $A$  requires for finding an optimal or a sufficiently good solution. Theoretical investigations must start with the convergence issue nevertheless, since important performance measures are undefined or infinite if  $A$  has a nonzero probability of *never* arriving at an optimum, as, e.g., in the case of premature convergence.<sup>3</sup>

Typical performance measures in the runtime analysis of stochastic search algorithms are (among others) as follows:

- The probability  $\mu_t = \Pr\{x_t^{\text{curr}} \in S^*\}$  that the current solution in iteration  $t$  is optimal. He and Yu [44] (cf. also [87]) call  $1 - \mu_t$  the *convergence rate*.
- The expected value or the distribution of the *first hitting time* (FHT)  $T_1$ , defined by  $T_1 = \min\{t \geq 1 : x_t^{\text{curr}} \in S^*\}$ .
- The expected value or the distribution of the time until a solution with a relative cost deviation from the optimum of less than some  $\varepsilon$  has been found.

The measures above relate to a single given problem instance, say, a fixed distance matrix in a case of a TSP. In order to obtain more general information, one is usually rather interested in the behavior of  $A$  for a *class* of problem instances. In *complexity analysis*, all instances of a given problem of a certain *size*  $n$  are considered (say, all  $[n \times n]$  distance matrices in the case of a TSP), where a suitable measure for instance size is applied. Then, the dependence of a fixed performance measure on  $n$  is studied. Since algorithm  $A$  has a different expected first hitting time for each instance of size  $n$ , some sort of aggregation is necessary. The two most important options for aggregation are to consider either the *worst case* performance over all instances of size  $n$  or the *average case* performance, given some probability distribution on the set of instances of size  $n$ .

### 19.4.1 Some Methods for Runtime Analysis

The mathematical analysis of the runtime of stochastic search algorithms is still in its infancies. Each metaheuristic field has developed some specific techniques for analyzing computation times on selected optimization problems. However, a few general methods that turned out as successful for more than one metaheuristic algorithm can be identified. Below, we shortly outline four of these methods. The reader is also referred to [33, 64] for more details.

---

<sup>3</sup> To ask, say, for the expected time until first hitting an optimal solution without being sure that the optimum will be reached, is as meaningless as to ask “How much training time would it take in the average for a randomly selected person to win an olympic gold medal?” Also by being content with an approximate solution of a certain minimum quality (call it the “silver medal”) instead of the optimal solution, one does not escape this difficulty.

(1) *Markov Chain Theory.* As noted in Section 19.2, the process  $(M_t)$  is a Markov process. In cases where the memory content  $M_t$  can only take finitely many values, the state space for this process is finite, i.e.,  $(M_t)$  is a (homogeneous) *Markov chain*. An example is GAs, where  $M_t$  contains a population of solutions  $x \in S$ . In the probabilistic literature, much is known about Markov chains, and some results can be exploited for the analysis of corresponding stochastic search algorithms. Following He and Yao [42], let us suppose, e.g., that by construction of  $A$ , states of  $M_t$  containing an optimal solution are never left again during the process (they are “absorbing states”), whereas other states have a probability larger than zero of being left in the next iteration (they are “transient states”). Let  $\mathbf{A}$  and  $\mathbf{T}$  denote the set of absorbing states and of transient states, respectively, and let  $j = |\mathbf{A}|$  and  $k = |\mathbf{T}|$ . Giving the  $j$  states in  $\mathbf{A}$  the lowest and the  $k$  states in  $\mathbf{T}$  the highest indices, the probability transition matrix  $P$  of the Markov chain  $(M_t)$  can be decomposed in the form

$$P = \begin{bmatrix} I_j & 0 \\ R & T \end{bmatrix},$$

where  $I_j$  is the  $[j \times j]$  identity matrix,  $0$  is the  $[j \times k]$  matrix with all elements equal to zero, and  $R$  and  $T$  are  $[k \times j]$  and  $[k \times k]$  matrices, respectively. He and Yao [42] show by direct application of a classical Markov chain result that the vector  $\mathbf{m}$  containing as the  $i$ th component the expected first hitting time  $m_i$  of the set of absorbing (i.e., optimal) states when starting from transient state  $i$  is given by

$$\mathbf{m} = (I_k - T)^{-1}(1, \dots, 1)^t,$$

where  $I_k$  is the  $[k \times k]$  identity matrix. In principle, this would allow the computation of expected first hitting times, but the matrix  $I_k - T$  is usually difficult to invert. Thus, the result can only be applied in cases where  $P$  has some special form (see, e.g., [64]).

(2) *Level Sets.* This method evolved in papers on the analysis of evolutionary algorithms (EAs) such as [8] or [11]. It tries to circumvent the state-space explosion for growing  $n$ , unavoidable in the direct application of the Markov chain approach, by grouping solutions into classes, where the fitness values are used as a natural criterion for defining the classes. Certain ranges of the fitness function (“levels”) correspond to certain subsets (“level sets”) of the search space  $S$ . The level sets have to be ordered in such a way that if  $x \in A_j$  and  $y \in A_k$  for two level sets  $A_j$  and  $A_k$  with  $j < k$ , it must always hold that  $f(x) < f(y)$ . In the cases easiest to analyze, the stochastic search algorithm  $A$  never returns to a level set corresponding to a lower fitness value after it has already visited a level set corresponding to a higher fitness value. For example, this monotonicity property is satisfied if  $A$  relies on the best-so-far solution  $x_t^{\text{bsf}}$  for the update from  $(M_t, L_t^+)$  to  $(M_{t+1}, L_{t+1}^+)$ , since  $x^{\text{bsf}}$  can never decrease for increasing  $t$ . Now, if it is possible to determine a lower bound for the probability that the process jumps from some level  $j$  to a higher level  $k > j$ , an upper bound for the expected staying time in level  $j$  can be derived, and from those bounds, one can obtain an upper bound for the time until the highest (i.e., optimal) level is reached.

This idea has turned out as fruitful for runtime analysis purposes not only in the field of EAs but also in the ACO field (see, e.g., [33, 34, 37]). In the PSO field, the level-set method has recently been applied by Sudholt and Witt [76]. For an extension of the method using the concept of *potential functions*, see [84].

(3) *Drift Analysis.* Drift analysis derives from martingale theory and has been applied for the analysis of SA (see [69]) and later for that of EAs (see, e.g., [41, 43]). Consider again the Markov process  $(M_t)$  and suppose that  $x_t^{\text{curr}}$  can be derived directly from  $M_t$ , i.e.,  $x_t^{\text{curr}} = x^{\text{curr}}(M_t)$ . (If also the information in  $L_t^+$  is required for getting  $x_t^{\text{curr}}$ , the process  $(M_t, L_t^+)$  must be considered instead of  $(M_t)$ .) Based on  $x_t^{\text{curr}}$ , a *distance*  $V(M)$  between state  $M$  and the set of states supporting optimal solutions may be defined. For example, one may set  $V(M) = |f(x^{\text{curr}}(M)) - f^*|$ , where  $f^*$  is the objective function value of the optimal solution. The one-step *mean drift* in state  $M$  is defined as the conditional expectation

$$E(V(M_t) - V(M_{t+1}) \mid M_t = M) = V(M) - \sum_{M'} P(M, M')V(M'),$$

where  $P(M, M')$  is the transition probability from state  $M$  to state  $M'$ . If the mean drift is always zero, the process  $V(M_t)$  is a martingale, which means that an optimal solution can only be found by chance. Hopefully, however, the drift generated by a stochastic search algorithm is positive, such that there is a tendency of the process to approach the set of optimal solutions.

He and Yao [41] show that from a lower bound on the mean drift, an upper bound on the expected first hitting time can be derived: If the mean drift in state  $M$  is larger or equal to some constant  $c_{\text{low}} > 0$  for any  $M$  with  $V(M) > 0$ , then the expected first hitting time after start in state  $M_1$  satisfies  $E(T_1 \mid M_1) \leq V(M_1)/c_{\text{low}}$ . With the help of this and similar lemmas, the behavior of some EAs on simple test functions has been successfully analyzed. The generality of the formalism shows that drift analysis should be applicable in principle to every type of stochastic search algorithms.

(4) *Stochastic Approximation.* In some cases, where the process  $(M_t)$  itself appears too difficult for a mathematical analysis, one may try to obtain asymptotic approximations to this process for limiting cases concerning special parameter values. An example is given in [32, 34], where the behavior of the *Ant System* variant of ACO, developed by Dorigo et al. [13], is analyzed on simple test problems for the case of small learning rate  $\rho$  for the pheromone update ( $\rho$  is usually called “evaporation rate” in the ACO literature). In Ant System, the solution quality achieved in iteration  $t$  can also decrease compared to iteration  $t - 1$ . Therefore, the level-set method is not applicable to this algorithm, contrary to some variants of MAX-MIN-Ant-System. However, letting  $\rho$  become small allows the application of the theory of slow learning that has been developed early in the learning literature (see [63]). As stated in Section 19.2, in ACO, the memory  $M_t$  contains a vector of pheromone values. In the limiting case  $\rho \rightarrow 0$ , the dynamics of this vector becomes deterministic and can be described by a system of differential equations. A similar approach has been pursued by Purkayastha and Baras [66].

Stochastic approximation techniques of this type may also be helpful for the analysis of other stochastic search algorithms where the memory content  $M_t$  lies in a continuous state space, e.g., EDAs or PSO. Indeed, in one of the first articles using an approach of this type, Gonzales et al. [26] refer to the analysis of PBIL, which is a special EDA.

### 19.4.2 Instance Difficulty and Phase Transitions

The methods presented in Section 19.4.1 aim at the analysis of a stochastic metaheuristic  $A$  for a special problem instance  $(S, f)$ . As noted at the beginning of Section 19.4, the topic of interest is typically not the behavior of  $A$  for a single instance, but for a class of instances, say the instances of size  $n$  of a given CO problem. The class may contain instances with completely different properties, such that the concepts of worst-case and of average-case analysis come into play.

In the case of some simple problems such as Generalized OneMax, which will be described in Section 19.4.3, the degree of difficulty is the same for all instances of size  $n$ . This is not true anymore for most CO problems relevant in applications. However, it seems that the degree of difficulty is usually not completely “scattered” among the instances, but often depends on some characteristic parameters of instances which are called *control parameters* or *order parameters*. In the seminal paper by Cheesman et al. [9], it has been shown experimentally that for some fundamental NP-hard combinatorial *decision* problems, such as  $k$ -SAT, Hamilton Circuits or Graph Coloring, different regions of the set of instances, such as “underconstrained” or “overconstrained” regions, have to be distinguished; their boundary is defined by a critical value  $\alpha_c$  of a control parameter, and the probability of the existence of a solution with the required properties changes abruptly from near zero to near one when passing the boundary. The larger the instance size  $n$ , the sharper is the transition. In analogy to phenomena in physics as the melting of ice, this behavior is called *phase transition*. The computation time required for solving the problem is typically high near the phase transition and low for control parameter values far from  $\alpha_c$  (“easy–hard–easy pattern”); sometimes, also “easy–hard” or “hard–easy” patterns are found.

Similar phase transition phenomena have also been observed in NP-hard combinatorial *optimization* problem, e.g., Number Partitioning [24], resource-constrained project scheduling problems [45], TSPs [88], independent-set problems [3], Max  $k$ -SAT [1] and vertex-cover problems [40]—problems that form the natural range of application for stochastic search algorithms.

Whereas Cheesman et al. [9] use computational experiments for investigating the phase transition phenomenon in CO, essential progress in a theoretical understanding of this phenomenon has been achieved during the last decade by the physics literature, especially by the approach of applying concepts from *statistical mechanics* to CO. Martin et al. [55] and Monasson [59] give introductions to this field. Statistical mechanics investigations of CO problems usually start with

the *Boltzmann distribution*<sup>4</sup> on the search space  $S$ , which is given by  $p(x) = (1/Z_T) \exp(-f(x)/T)$ , where  $x \in S$  is a solution,  $p(x)$  is the probability of  $x$ ,  $f$  is the cost function (called *energy* in the physics literature), the parameter  $T \geq 0$  is called *temperature*, and the normalization factor  $Z_T = \sum_{y \in S} \exp(-f(y)/T)$  is called the *partition function*. The two boundary cases  $T = \infty$  and  $T = 0$  produce a uniform distribution on  $S$ , resp. a distribution that is concentrated on the set of global optima, the so-called *ground states*. The crucial idea is that by letting  $T$  tend to zero and investigating the partition function  $Z_T$  in this asymptotic limit, information on global optima is obtained, in particular information on the optimal cost function value  $f^*$  (“ground-state energy”) or on the number of global optima.

In order to get from single problem instances to instance classes, the quantities derived from the partition function are averaged over the distribution of instances within the class of interest. As an example, consider the number partitioning problem (NPP) with  $n$  items, the weights of which are represented by  $b$ -bit integers. A solution  $x$  consists in a partition of the set of given items into two subsets, and the cost function is the absolute difference between the total weights of the two subsets. Here, the ratio  $b/n$  turns out as the relevant control parameter. The statistical mechanics approach predicts a phase transition around  $b/n \sim 1$ , with an exponentially growing search cost for  $b$  large compared to  $n$ , and polynomially growing cost for  $b$  small compared to  $n$  (see, e.g., [57]). This is in good agreement with experimental results.

From a practical point of view, the results on phase transitions indicate that for testing or tuning a metaheuristic algorithm, a suitable choice of the instance distribution is very important. In particular, it does not make too much sense to mix instances from the “easy” and from the “hard” region, since the last will dominate the average behavior, which may mask the information that can be obtained for the easier instances.

### 19.4.3 Some Notes on Special Runtime Results

For reasons that will be discussed in Section 19.6, it is rather unlikely that for a stochastic search algorithm, universal positive runtime results (i.e., results valid for all CO problems predicting computation times of practical interest) can be obtained. Therefore, the promising way is to investigate different problems separately from each other, starting with very simple ones in order to develop useful analytical techniques, and successively progressing to the hard CO problems found in applications.

As a consequence of the necessity to study runtime issues separately for the single problems, the literature on analytical runtime results for stochastic search heuristics is rather dispersed. An overview would be beyond the scope of this chapter. We focus therefore on some few key issues. Readers interested in more details may

---

<sup>4</sup> The relevance of this distribution in the field of stochastic search is also underlined by the fact that one of the oldest general-purpose stochastic search techniques, namely SA, approximates at each fixed temperature level  $T$  the corresponding Boltzmann distribution.

consult, e.g., the survey [64], which addresses the evolutionary algorithms field excluding the swarm-intelligence metaheuristics ACO and PSO, and the survey concerning ACO provided in [33].

Typical simple problems investigated in the literature consist of artificially constructed test functions, usually pseudo-boolean functions, i.e., functions mapping the set  $S = \{0, 1\}^n$  of binary strings  $x = (x_1, \dots, x_n)$  of length  $n$  into the reals. Examples are the OneMax fitness function  $f(x) = \sum_{i=1}^n x_i$ , the LeadingOnes fitness function  $f(x) = \sum_{i=1}^n \prod_{j=1}^i x_j$ , or the Needle-in-a-Haystack (NIAH) fitness function  $f(x) = \prod_{j=1}^n x_j$ . These three functions (instances) can be generalized to problems (classes of instances). For example, the Generalized OneMax problem contains the fitness functions  $n - d_H(x, x^*)$ , with  $d_H$  denoting the Hamming distance and  $x^* \in S$  being an arbitrary fixed solution. (The OneMax function is the special case where  $x^* = (1, \dots, 1)$ .) Also more general classes have been successfully analyzed in the literature. For example, Droste et al. [11] have shown that for all *linear* pseudo-boolean functions, the expected first hitting time  $E(T_1)$  of a simple EA, the (1+1) EA, grows as  $\theta(n \log n)$  in the instance size  $n$ . In [81], some results on *quadratic* pseudo-boolean functions have been derived; this class already contains NP-hard optimization problems. Also for more complex EAs and for other stochastic metaheuristics, results concerning pseudo-boolean functions have been proved in the meantime (see the cited surveys).

Part of the literature analyzes the behavior of stochastic search algorithms on practically relevant problems from the complexity class  $P$ , i.e., problems for which polynomial-time solution algorithms exist. Such problems are good benchmarks for testing a metaheuristic algorithm which should be able to solve them by requiring only a low computational overhead compared to a problem-specific algorithm. In particular, sorting problems (e.g., [70]), maximum matching problems (e.g., [25]), and minimum spanning tree problems (e.g., [60, 62]) have been analyzed in an EA or ACO context. As expected, the investigated metaheuristics perform worse than the respective “tailored” algorithms, but they usually remain efficient in the sense that only polynomially growing runtime is required.

Very few works exist that analyze stochastic metaheuristics on NP-hard problems. Witt [83] investigates the behavior of the (1+1) EA on a variant of the NPP (see Section 19.4.2) for which a fully polynomial approximation scheme exists. Within  $O(n^2)$  steps, the (1+1) EA finds a solution that is at least  $(4/3)$ -approximate. For the maximum clique problem on random planar graphs, Storch [73] proves that SA with constant temperature finds an optimal solution in linear time with overwhelming probability, while the (1+1) EA needs  $\theta(n^6)$  iterations. For the vertex cover problem, Friedrich et al. [19] show that the (1+1) EA can produce arbitrarily poor solutions, whereas the evolutionary multi-objective optimizer SEMO performs sufficiently well. The poor performance of the (1+1) EA can also be remedied by applying multistarts, as Oliveto et al. [65] demonstrate.

The issue of the possible advantage of random multistart also raises another interesting question. Whether or not random multistart is beneficial depends on the *distribution* of the first hitting time  $T_1$ . Therefore, results that do determine not only the expected value  $E(T_1)$  but the entire distribution of the random variable  $T_1$  would

be very useful. Only few results of this type seem to exist. We give two examples. Garnier et al. [20] show that for the first hitting time  $T_1(n)$  of (1+1) EA on a OneMax instance of size  $n$ , the re-normalized value  $(T_1(n) - en \log n)/n$  converges in distribution to  $-e \log Z + C$  as  $n \rightarrow \infty$ , where  $Z$  is exponentially distributed with parameter  $\lambda = 1$  and  $C$  is a constant. Ladret [53] proves that for the first hitting time  $T_1(n)$  of (1+1) EA on a LeadingOnes instance, the re-normalized value  $(T_1(n) - mn^2)/n^{3/2}$  with  $m = (e - 1)/2$  converges in distribution to a normal distribution with mean 0 and variance  $3(e^2 - 1)/8$ .

## 19.5 Parameter Choice

One of the most important questions for the application of a metaheuristic algorithm is how its parameters should be chosen in order to obtain a good algorithmic performance for the application case at hand. Considering the functions  $g$  and  $h$  of the generic algorithm of Section 19.2, we may distinguish between *sampling parameters* contained in  $g$  (they govern the distribution of the sample points in  $L_t$ ) and *learning parameters* contained in  $h$  (they determine the type and amount of influence of the fitness values observed in the sampled trial points on the new memory content  $M_{t+1}$ ). Examples of sampling parameters are mutation rate and crossover rate in GAs. Examples of learning parameters are temperature in SA and the learning rates used in ACO and in some EDAs, respectively.

A first question in this context is whether it is better to keep parameters constant during the optimization run or whether they should better be changed dynamically. There are good empirical and theoretical arguments for the second alternative. Convergence results for more than one stochastic search algorithm are based on dynamic parameter schemes. For example, the classical convergence results [38] for SA require that the temperature parameter  $T$  is gradually decreased. Similarly, in [29], one of two indicated options for obtaining convergence of an ACO algorithm consists in gradually reducing the learning rate  $\rho$ . It seems that such a dynamic management of a central parameter of a stochastic search algorithm is a key instrument for achieving an exploration–exploitation balance.

Despite this intuitive consideration, it is surprisingly hard to verify the benefits of a dynamic parameter scheme by demonstrating rigorously that performance measures as the expected first hitting time can be improved if the parameter values are *not* kept constant. For example, in the SA literature there has been a long discussion about the question “to cool or not to cool”: Is it really advantageous to decrease  $T$  during the optimization process, as experimental customs and theoretical convergence results suggest, or can the same performance be achieved by applying the so-called Metropolis algorithm that works at fixed, constant temperature  $T$ ? Artificial examples of test functions for which the regular SA with a decreasing temperature scheme is more efficient have been known since the beginning of the 1990s, but it was not before 2005 that Wegener [80] presented a *natural* optimization problem (the minimum spanning tree problem) for which SA can be shown to outperform Metropolis.

In cases where there is no reason suggesting that a gradual reduction of the basic parameter of a stochastic search algorithm might be beneficial, we may still be interested in knowing whether it is better to keep the parameter at a fixed value or to let it oscillate in some way in order to give the process a higher degree of variability. Jansen and Wegener [49] investigate this question analytically for the (1+1) EA, applied to simple test functions such as OneMax and LeadingOnes (see Section 19.4.3). It turns out that the static variant where the parameter under consideration, the mutation probability  $p$  of the (1+1) EA, is fixed to a constant is better for some test functions than the dynamic variant where  $p$  is cyclically changed and is worse for some other test functions. The choice between the static and the dynamic scheme for a specific test function can make the difference between polynomial and exponential runtime.

Apart from the question whether or not parameters should be changed dynamically *during* the process, implementers of metaheuristics are always confronted with the question of how the parameter values should be adapted to properties of concrete problem instances, in particular to the size  $n$  of the instance.

Let us give an example from the ACO domain showing how analytical results can help to get insight into this issue (for details, cf. [33]). First investigations of the runtime of certain ACO variants [34, 61] seemed to indicate that for the Generalized OneMax problem, one has to apply comparably high values of the learning rate  $\rho$  to obtain the favorable expected first hitting time of order  $\theta(n \log n)$  that has been known for the (1+1) EA. In [37], it is demonstrated that a natural ACO algorithm of MAX-MIN-Ant-System type can solve Generalized OneMax within expected time of order  $\theta(n \log n)$  also for a *small* value of  $\rho$  independent of the problem size  $n$ . Similar results are obtained for the LeadingOnes problem. More than that: As soon as one passes from a fitness function giving much “guidance” to the search process, as it is provided by OneMax or LeadingOnes, to a fitness function where parts of the optimal solution have to be identified rather by trial-and-error than by the guidance delivered via the neighborhood structure, it becomes *essential* for the efficiency of the search process that  $\rho$  is chosen small enough. Consider, e.g., a combination of the functions OneMax and NIAH explained in Section 19.4.3, defined by  $f(x) = (\prod_{i=1}^k x_i) \cdot (\sum_{i=k+1}^n x_i + 1)$ . For the maximization of this function, it is necessary that the correct bits on the first part of length  $k$  of the string are found by trial-and-error (this is the NIAH part), and the remaining  $n - k$  bits are optimized as for a OneMax problem. It is shown in [37] that this problem can only be solved efficiently if the learning rate  $\rho$  is decreased with increasing problem size  $n$ , but not too fast: For  $k = \log_2 n$ , a scheme of order  $\theta(n^{-3})$  is suitable to obtain polynomial expected first hitting time. On the other hand, both (1+1) EA and ACO with constant  $\rho$  require exponential expected time.

Another example is the following. In [76], Sudholt and Witt show that for the Binary PSO algorithm, keeping the (usually applied) bound  $v_{\max}$  on the velocity of the particles fixed when increasing the instance size  $n$  leads to an extreme decline of the performance. Scaling  $v_{\max}$  to  $n$  by the function  $v_{\max} = \ln(n - 1)$  considerably improves the runtime behavior on the considered test functions.

## 19.6 No-Free-Lunch Theorems

Looking at the co-existence of a considerable number of stochastic metaheuristics, a natural question would be to ask which is the “best” of them. In more specific terms: Can we derive a *universal* result stating that for all CO problems, some stochastic search algorithm  $A_1$  always performs better than some other stochastic search algorithm  $A_2$ ? To have results of this type would be extremely valuable by simplifying the complex landscape of metaheuristics, but the hope to obtain them broke down when Wolpert ad Macready [85] published their famous *No-Free-Lunch* (NFL) theorems for optimization. Basically, they state that in the average over all possible fitness functions, no black-box search algorithm (may it be deterministic or stochastic) can be better than straightforward random search.

Before discussing this surprising result in more detail, we have to formulate the setting for which it holds in precise terms. It is not an essential restriction to assume that in our formulation (19.1) of a CO problem, the range of the function  $f$  is some *finite* subset  $Y$  of the set of reals: we might simply restrict the range to the image of the finite set  $S$  under  $f$ . Evidently, for fixed  $S$  and  $Y$ , there exist  $|Y|^{|X|}$  different mappings (fitness functions)  $f : S \rightarrow Y$ . Assume that each of them has the same probability. Furthermore, let us restrict ourselves to search algorithms  $A$  (they can be deterministic or stochastic, i.e., the functions  $g$  and  $h$  introduced in Section 19.2 can depend on the random influences  $z$  and  $z'$  or not) with the property that the list  $L_t$  always contains only sample points  $x \in S$  that have *not yet been visited* in previous iterations—in other words, with the property that the sets  $L_t$  are disjoint for  $t = 1, 2, \dots, t_{\max}$ , where  $t_{\max}$  is the iteration in which algorithm  $A$  terminates. (Of course, this is a strong assumption, since it presupposes that  $A$  stores information on already visited sample points in the memory  $M_t$ ; its consequences, especially for *stochastic* search, where memory is saved to some extent by re-sampling, have not been fully investigated up to now.)

It is rather clear that under these circumstances, the fitness values observed in the sample points from  $S$  that have already been visited before some iteration  $t$  do not give any information on the fitness values in the sample points that have not yet been visited. Let us denote the set  $\bigcup\{L_u | u < t\}$  of already visited points (solutions) by  $S_v(t)$ , such that  $S \setminus S_v(t)$  is the set of yet unvisited points. By the assumption of a uniform distribution on the set of all possible fitness functions  $f$ , the function values  $f(x)$  for  $x \in S \setminus S_v(t)$  are *independent* from the (observed) function values in points from  $S_v(t)$ . Therefore, no matter which rule algorithm  $A$  applies to determine  $L_t$ , the information on the fitness values on  $S_v(t)$  gathered in iterations  $u = 1, \dots, t - 1$  is of no use for getting any hint how to explore the yet completely unknown domain  $S \setminus S_v(t)$ . As a consequence, every rule is equally efficient in the average; in particular, it is neither more efficient nor less efficient than random search. To each fitness function  $f$  for which  $A$  performs better than random search, there is another fitness function for which it performs worse.<sup>5</sup>

---

<sup>5</sup> There seem to be close relations between NFL theorems and the well-known philosophical *induction problem* that plays also a role in AI approaches to inductive reasoning. Suppose that the

Of course, this result does not imply that on a special given *problem*, every stochastic search algorithm has the same performance. However, it seems that the NFL theorems force us to investigate search algorithms separately for each problem, because if  $A_1$  dominates  $A_2$  on some problem  $P_1$ , there must be another problem  $P_2$  where  $A_2$  dominates  $A_1$ .

Some researchers have drawn rather radical implications from the NFL theorems, questioning the field of metaheuristics as a whole. For example, Whitley and Watson [82] report that one extreme reaction consists in the conclusion that there are no effective general-purpose search methods at all. Already early, there has been a resistance against such over-interpretations, and authors have begun to investigate the limitations of the NFL theorems. Their arguments proceed mainly along two lines:

1. *Complexity Issues.* Whereas the NFL theorems hold for the set of *all possible* fitness functions, it is usually not this set we encounter in practice when solving optimization problems. Rather than that, the objective functions in classical CO problems have comparably low Kolmogorov complexity (KC). Droste et al. [10] show by a concrete example that NFL theorems need not to hold in classes of functions with restricted complexity, and that “intelligent” search algorithms are able in such a context to outperform random search. English [14, 15] demonstrates that for search spaces  $S$  of medium to large size, almost all functions  $f : S \rightarrow Y$  are “random” (in the sense of having a high KC), and he argues that random functions do not pose *practical* problems for heuristic optimization, because for them already simple optimizers discover good solutions quickly. On the other hand, “hard” problems are rare and therefore not represented adequately by the average-case consideration of the NFL theorems. Further results on the relation between KC and NFL theorems are presented, e.g., in [6, 7].
2. *Influence of Fitness Landscape Properties.* Igel and Toussaint [46, 47] prove a sharpened NFL theorem giving a sufficient *and* necessary condition (closedness of the set of fitness functions under permutation) for the NFL theorem to hold. Using this condition, they show that if a non-trivial neighborhood structure on  $S$  has an influence on the fitness, the NFL theorem does *not* hold. In particular, a set of fitness functions satisfying certain steepness constraints obviate the NFL theorem. (Suitable steepness constraints may, e.g., exclude a case where by a move from a solution to an immediate neighbor solution, the cost function jumps from a global maximum to a global minimum.) Therefore, most practical applications do not fall under the NFL verdict. A similar conclusion is drawn in [52].

---

function evaluation of  $f(x)$  in the solution  $x \in S$  is not done by an algorithmic computation, but rather by the observation of some real-world system (say,  $x$  is a control vector for a chemical plant, and  $f(x)$  is the observed value of an outcome variable). Then the “NFL insight” that there is no logical argument why the observation of  $f(x_1), \dots, f(x_{t-1})$  for some sample solutions  $x_1, \dots, x_{t-1}$  should provide any information on  $f(x_t)$  for the sample solution  $x_t \notin \{x_1, \dots, x_{t-1}\}$ , basically amounts to the intriguing claim by David Hume that we do not have any logical justification for the step called “inductive conclusion,” although this step is indispensable in science as in everyday life.

One may be relieved about the fact that NFL results do not have the consequence that developing efficient metaheuristics is a futile goal, but one should not miss the message: Since complexity properties are hard to deal with in applications (KC is not computable!), it is only the structure of the fitness landscape that may “give us a free lunch” when applying general-purpose search algorithms. This leads us to the topic of fitness landscape analysis.

## 19.7 Fitness Landscape Analysis

A fitness landscape is formally defined as a triple  $(S, d, f)$ , where  $S$  and  $f$  are search space and fitness function, respectively, and  $d$  is a distance function on  $S$  assigning to each pair  $(x, x')$  of solutions  $x, x' \in S$  a nonnegative integer distance [58, 71]. (If a neighborhood structure on  $S$ —as used by ILS, SA, or VNS—is given,  $d$  is derived in a natural way as the shortest distance in the neighborhood graph. Conversely, given  $d$ , solutions  $x, x'$  with  $d(x, x') = 1$  are considered as neighbors.) In the literature, several quantities characterizing properties of the fitness landscape that are relevant for search algorithms have been defined (see, e.g., [58, 67]). For the sake of shortness, let us restrict ourselves to two examples:

- The *fitness distance correlation* (FDC) is defined as

$$\rho(f, d_{\text{opt}}) = \text{cov}(f, d_{\text{opt}}) / [\sigma(f) \sigma(d_{\text{opt}})],$$

where  $d_{\text{opt}}$  is the distance of a solution to the nearest optimal solution, and cov and  $\sigma$  denote covariance and standard deviation, respectively. For a maximization problem, a value of  $\rho(f, d_{\text{opt}})$  near the minimum possible value of  $-1$  indicates that the fitness is ideally correlated with the distance to the optimum solution; such landscapes are easy for stochastic local search algorithms or GAs. On the other hand, a value of  $\rho(f, d_{\text{opt}})$  around  $0$  makes the problem harder, and a value near  $1$  indicates that the problem is “deceptive.”

- The *random walk correlation function* is defined as the average of

$$r(s) = \frac{1}{\sigma^2(f)(m-s)} \sum_{t=1}^{m-s} (f(x_t) - \bar{f})(f(x_{t+s}) - \bar{f}),$$

where  $x_1, \dots, x_m$  is a sampled random walk on the neighborhood graph of  $S$ ,  $\sigma^2(f)$  denotes the variance of the fitness, and  $\bar{f}$  is the mean fitness.

Experimentally, a considerable effect of fitness landscape measures as those above on the efficiency of stochastic search algorithms has been observed.

One of the most interesting parameters in fitness landscape analysis and simultaneously a crucial parameter for the performance of stochastic local search algorithms is the number  $N_{\text{loc}}$  of local optima. Fitness landscapes with a large number of local optima are “rugged” and hard for optimization. Reidys and Stadler [67] give a “correlation length conjecture” for the estimation of  $N_{\text{loc}}$  from the

so-called correlation length  $\ell = -[\ln(|r(1)|)]^{-1}$ , where  $r$  is the random walk correlation function defined above. Empirical evidence supports this conjecture. Garnier and Kallel [21] describe a general technique for estimating  $N_{\text{loc}}$  by performing repeated local search with  $M$  random start solutions and observing the multiplicities by which the found local optima are covered. Moreover, the methodology provides bounds on the search complexity for detecting all local optima. Eremeev and Reeves [16] are even able to determine confidence intervals for  $N_{\text{loc}}$ .

For some problems, the number of local optima can also be estimated analytically. For example, by means of the statistical-mechanics approach outlined in Section 19.4.2, Ferreira and Fontani [17] derive the expression  $N_{\text{loc}} \sim 2.764 \cdot 2^n n^{-3/2}$  for the average number of local optima of the NPP problem (see Section 19.4.2) under a uniform distribution model, which is in good agreement with simulation results. Considering the fact that an NPP instance of size  $n$  has  $2^n$  feasible solutions, we see that a simple ILS implementation will presumably not be very efficient in this case, compared to complete enumeration, unless if one single local search run should take distinctly less than  $O(n^{3/2})$  time.

## 19.8 Black-Box Optimization

The basic forms of most stochastic metaheuristics are characterized by the property that the algorithms do not exploit any information about the specific problem instance (say, the distance matrix in a TSP), but only use information on the search space, including aspects as neighborhood structure, and on the concrete problem type under consideration. In our generic framework, this scenario is represented by the condition that the functions  $g$  and  $h$  are not allowed to depend on the problem instance. One may imagine then that the algorithm  $A$  repeatedly calls a “black-box” procedure returning fitness values to given solutions  $x$ , but  $A$  does not “know” how these fitness values are determined. In this case,  $A$  is called a *black-box optimizer*.

From the viewpoint of a unified theory of stochastic search, it is interesting to investigate the potential of black-box optimizers independently from their concrete algorithmic mechanisms. Recently, some articles have started to study this issue. In [12], Droste et al. investigate upper and lower bounds for the expected first hitting times of stochastic black-box optimizers. The authors introduce a generic stochastic search algorithm “Black-Box Algorithm 1” which is essentially the generic algorithm of Section 19.2 with  $L_t$  restricted to a single element and  $M_t$  consisting of the entire search history, i.e., the sequence  $(x_1, f(x_1), \dots, x_{t-1}, f(x_{t-1}))$  of solutions visited before iteration  $t$ , together with their fitness values. In an algorithm “Black-Box Algorithm 2,” the size of the memory  $M_t$  is restricted by a size bound  $s(n)$  depending on the instance size  $n$  (which can be seen as a property of  $S$  and does therefore not violate the black-box restriction). As a measure of performance, the expected optimization time in the worst case over all instances of size  $n$  is used.

For obtaining lower bounds on  $E(T_1)$ , the authors apply Yao’s [86] minimax principle, which can be stated as follows: The expected optimization time of a stochastic

search algorithm  $A$  in the worst case over all instances is lower bounded by the expected value of the optimization time of an optimal deterministic search algorithm, where the last expected value is taken with respect to an arbitrary instance distribution.

Droste et al. investigate sorting problems, for which they obtain linear lower bounds (and, depending on the measure for “sortedness” which is used as the fitness function, linear or slightly super-linear upper bounds), classes of simple functions such as the linear pseudo-boolean functions, and some more complex pseudo-boolean functions as well as special classes of unimodal functions. To give a flavor of the type of results, let us consider the Generalized OneMax example, for which in [12], the lower bound  $n/\log(2n+1) - 1$  is derived.<sup>6</sup> It is intuitively clear why we obtain a lower bound of order  $O(n/\log n)$  here: Since there are  $n+1$  different fitness function values, each call of the black-box procedure determining the fitness of a solution  $x$  gives us  $\log_2(n+1)$  bits of information. On the other hand, since there are  $2^n$  possibilities for the optimal solution  $x^*$ , we need a total information of  $n$  bits to identify  $x^*$ . Thus, an optimally designed search procedure will require about  $n/\log_2(n)$  black-box calls.

Teytaud and Gelly [77] present lower bounds for black-box optimizers on problems with *continuous* search space and consider the scenario where only pairwise comparisons between fitness values are allowed to govern the search process instead of the overall information contained in the fitness value.

Another interesting perspective is presented by Borenstein and Poli [6]; it relates NFL theorems, fitness landscape analysis, and black-box optimization to each other. The authors argue that it is not sufficient to analyze the fitness landscape for itself; only by connecting it to the operators used during the search, the information contained in it becomes relevant. A “proper” black-box optimizer should not have any *a priori* preference for any regions of the search space, but rather select new sample points (in our notation: the elements of  $L_t$ ) on the basis of their distance from already visited points. This requires that the applied search operators are in some sense consistent with the metric structure on  $S$ , which is described in [6] in algebraic terms.

Finally, let us mention that several practically applied variants of stochastic metaheuristics are not black-box optimizers, but use in addition to a black-box-type core mechanism also information on the problem instance. An example is the use of problem-specific heuristic values in ACO. We might call such algorithms *gray-box optimizers*, distinguishing them also from the “white-box” optimization techniques of mathematical programming (MP). Some metaheuristics, such as GRASP or Extreme Optimization, are inherently “gray-box.” Perhaps the most radical form of gray-box optimizers are hybrids between metaheuristics and MP approaches such as Local Branching [18], which have been termed *matheuristic* algorithms.

---

<sup>6</sup> Note that both EA and ACO algorithms typically solve this problem in  $O(n \log n)$  time [11, 37], which differs from the lower bound by a factor of order  $O((\log n)^2)$ . This overhead may partly be explained by the effort for re-sampling already visited solutions.

## 19.9 Stochastic Search Under Noise

In applications, it often happens that decisions under uncertainty have to be made, where certain parameters of an optimization problem are not deterministically known in advance. Often, it is possible to represent these parameters, by using an appropriate stochastic model, as random variables. This leads to *stochastic optimization problems* where the objective function and sometimes also the constraints are disturbed by “noise.” In this chapter, we restrict ourselves to stochastic combinatorial optimization (SCO) problems of the following frequently occurring form, which is a natural extension of the deterministic CO problem (19.1):

$$E(f(x, \omega)) \rightarrow \min \text{ such that } x \in S. \quad (19.2)$$

Therein,  $E$  denotes the expectation operator, and  $\omega$  is a random influence with a distribution given by the stochastic model of the problem ( $\omega$  is not to be confounded with the random variables  $z$  and  $z'$  used by the stochastic search algorithm, see Section 19.2). As in the deterministic case, “min” can be replaced by “max.” As an example, consider the *stochastic total tardiness problem*, where a set of jobs  $1, \dots, n$  together with their due dates  $d_1, \dots, d_n$  are given. Each job has a processing time  $Y_i$  which is a random variable with known distribution. The objective is to find a sequential arrangement  $x$  of the  $n$  jobs such that the expected value of the sum of their tardiness values is minimized, where the tardiness of job  $i$  is given as  $(C_i - d_i)^+$ , with  $C_i$  denoting the completion time of job  $i$ . Note that  $C_i = C_i(x, Y)$  depends on both the solution  $x$  and the vector  $Y$  of random processing times. Here,  $\omega$  can be considered as identical to  $Y$ , and  $f(x, \omega) = \sum_{i=1}^n (C_i(x, \omega) - d_i)^+$ .

In the literature on stochastic optimization, several methods have been developed to solve problems of the form (19.2). In particular, *metaheuristic* algorithms have also been applied in this field; a recent survey is given in [4]. A survey focusing on EAs can be found in [50].

Basically, three different approaches to solve problems of the form above by a metaheuristic search algorithm following a black-box optimization paradigm<sup>7</sup> can be distinguished: (i) If possible, a procedure for the numerical computation of the expectation in Equation (19.2) is implemented, and the problem is solved in the same manner as a deterministic CO problem, performing black-box calls of the numerical procedure to obtain fitness evaluations. Often, this requires a large amount of computation time or is even infeasible. (ii) A sample of random instances for the uncertain parameters, distributed according to the given stochastic model, is generated as an approximation to the exact distribution, and after that, large-scale optimization averaging over this sample is done. This is called a *fixed-sample* approach. (iii) In a *variable-sample* approach, sampling and optimization are not two successive phases, but rather alternate during the iterations of the search algorithm. This allows the use of smaller sample sizes.

---

<sup>7</sup> For approaches using “white-box” mathematical programming techniques such as the Integer L-Shaped Method, see, e.g., [23].

General-purpose variable sample SCO algorithms have been derived from certain metaheuristics as SA [2, 22, 36], ACO [5, 30, 31], or VNS [35]. The structure of these algorithms is an extension of our generic scheme of Section 19.2. All we have to do is to replace in step 2b of the generic algorithm the evaluation of the objective function values  $f(x_i)$  by the determination of *sample average estimates*  $\tilde{F}(x_i) = \sum_{v=1}^N f(x_i, \omega_v)$  approximating  $F(x_i) = E(f(x_i, \omega))$ , where the  $\omega_v$  form a random sample for the uncertain parameter  $\omega$  according to the given distribution. The sample size  $N$  needs not to be fixed over the iterations, but can be chosen as a function of  $M_t$ . Typically,  $N$  is gradually increased to improve the accuracy of the estimate.

Using this approach and applying suitable schemes for  $N = N(M_t)$ , convergence results for the mentioned modifications of SA, ACO, or VNS have been derived in [30, 35, 36] by generalizations of the known convergence results for the corresponding basic metaheuristics.

## 19.10 Conclusions

In the past, metaheuristics have evolved in different scientific sub-communities, to a certain extent separately from each other. Although a strong tendency to a crosslinking between these sub-communities can be observed in the last years (see [56]), which already resulted in considerable synergy effects as well as in the establishment of a joint experimental methodology, a common theoretical framework enabling an immediate exploitation of progress in one of the metaheuristic subfields by other subfields seems still to be lacking. Much work will have to be done yet for achieving a unified understanding of metaheuristic algorithms.

One of the key elements around which a holistic view of the different metaheuristic techniques might be organized is the role that *stochastic search* takes in most of them. The results cited in this chapter may indicate possible starting points for a process leading to a general theory of stochastic search, but this process will still to have take place. Anyway, the fascination of many successes in solving real-world problems by particular metaheuristics should not lead the community astray from the attempt to envisage also the “big picture” by looking at what the single metaheuristic paradigms have in common.

As discussed in Section 19.6, it is not probable that one single metaheuristic will turn out as “superior” to the others and expel them from the application fields. Rather than that, it may be anticipated that the current situation of a co-existence of different metaheuristics will prevail. This makes it especially desirable to increase our understanding of the specific benefits of each of them in a framework within which they can be compared.

Many important topics have been excluded in this chapter, such as stochastic search in (non-linear) continuous, in multi-objective, or in dynamic optimization. Issues as, e.g., runtime analysis, are even less developed in these areas than in single-objective static CO, and many related open problems represent a challenge for future research.

## References

1. Achlioptas, D., Naor, A., Peres, Y.: Rigorous location of phase transitions in hard optimization problems. *Nature* **435**, 759–764 (2005)
2. A simulated annealing algorithm with constant temperature for discrete stochastic optimization. *Manage. Sci.* **45**, 748–764 (1999)
3. Barbosa, V.C., Ferreira, R.G.: On the phase transitions of graph coloring and independent sets. *Physika A* **343**, 401–423 (2004)
4. Bianchi, L., Dorigo, M., Gambardella, L.M., Gutjahr, W.J.: A survey on metaheuristics for stochastic combinatorial optimization. To appear in: *Natural Computing* **8**, 239–287 (2009)
5. Birattari, M., Balaprakash, P., Dorigo, M.: The ACO/FRACE algorithm for combinatorial optimization under uncertainty. In: Doerner K. et al. (ed.) *Metaheuristics – Progress in Complex Systems Optimization*, Springer: Berlin, Germany (2006)
6. Borenstein, Y., Poli, R.: Information Perspective of Optimization. Proceedings of 9th Conference on Parallel Problem Solving from Nature, Springer LNCS, vol. **4193**, pp. 102–111. Berlin Heidelberg (2006)
7. Borenstein, Y., Poli, R.: Structure and Metaheuristics. Proceedings of Genetic and Evolutionary Computation Conference '06, pp. 1087–1093 (2006)
8. Borisovsky, P.A., Eremeev, A.V.: A study on performance of the (1+1)-evolutionary algorithm. In: *Proceedings of Foundations of Genetic Algorithms*, vol. **7**, pp. 271–287. Morgan Kaufmann, San Francisco (2003)
9. Cheesman, P., Kenafsky, B., Taylor, W.M.: Where the really hard problems are. In: Morgan Kaufmann (ed.) *Proceedings of IJCAI '91*, pp. 331–337 (1991)
10. Droste, S., Jansen, T., Wegener, I.: Perhaps not a free lunch but at least a free appetizer. *Proceedings of Genetic and Evolutionary Computation Conference '99*, Orlando, FL, USA pp. 833–839 (1999)
11. Droste, S., Jansen, T., Wegener, I.: On the analysis of the (1+1) evolutionary algorithm. *Theor. Comput. Sci.* **276**, 51–81 (2002)
12. Droste, S., Jansen, T., Wegener, I.: Upper and lower bounds for randomized search heuristics in black-box optimization. *Theory Comput. Syst.* **39** 525–544 (2006)
13. Dorigo, M., Maniezzo, V., Colormi, A.: Ant System: optimization by a colony of cooperating agents. *IEEE Trans. Syst., Man Cybern* **26**, 1–13 (1996)
14. English, T.: Optimization is easy and learning is hard in the typical function. *Proceedings of Congress in Evolutionary Computation '00*, La Jolla, USA pp. 924–931 (2000)
15. English, T.: On the structure of sequential search: beyond “no free lunch”. *Proc. EvoCOP '04*, Springer LNCS, Coimbra, Portugal **3004**, 95–103 (2004)
16. Eremeev, A.V., Reeves, C.R.: On confidence intervals for the number of local optima. *Applications of Evolutionary Computing*, Springer LNCS, Berlin Heidelberg vol. **2611**, pp. 224–235 (2003)
17. Ferreira, F.F., Fontanari, J.F.: Probabilistic analysis of the number partitioning problem. *J. Phys. A: Math. Gen.* **31**, 3417–3428 (1998)
18. Fischetti, M., Lodi, A.: Local branching. *Mathematical Programming Ser. B* **98**, 23–47 (2003)
19. Friedrich, T., He, J., Hebbinghaus, N., Neumann, F., Witt, C.: Approximating covering problems by randomized search heuristics using multi-objective models. *Proceedings of 9th Annual Conference on Genetic and Evolutionary Computation*, pp. 797–804 (2007)
20. Garnier, J., Kalel, L., Schoenauer, M.: Rigorous hitting times for binary mutations. *Evol. Comput.* **7**, 45–68 (1999)
21. Garnier, J., Kallel, L.: Efficiency of local search with multiple local optima. *SIAM J. Discrete Math.* **15**, 122–141 (2002)
22. Gelfand, S.B., Mitter, S.K.: Simulated annealing with noisy or imprecise measurements. *J. Optim. Theor. Appl.* **69**, 49–62 (1989)
23. Gendreau, M., Laporte, G., Seguin, R.: An exact algorithm for the vehicle routing problem with stochastic demands and customers. *Transport. Sci.* **29**, 143–155 (1995)

24. Gent, I.P., Walsh, T.: Analysis of heuristics for number partitioning. *Comput. Intell.* **14**, 430–450 (1998)
25. Giel, O., Wegener, I.: Evolutionary algorithms and the maximum matching problem. Proceedings of 20th Annual Symposium on Theoretical Aspects of Computer Science, Seattle, Washington, USA pp. 415–426 (2003)
26. González, C., Lozano, J.A., Larrañaga, P.: Analyzing the PBIL algorithm by means of discrete dynamical systems. *Complex Syst.* **11**, 1–15 (1997)
27. González, C., Lozano, J.A., Larrañaga, P.: Mathematical modelling of discrete estimation of distribution algorithms. In: Larrañaga et al. (eds.) *Estimation of Distribution Algorithms, A New Tool for Evolutionary Computation*, Kluwer, Dordrecht 147–163 (2002)
28. Gutjahr, W.J.: A graph-based ant system and its convergence. *Future Gen. Comput. Syst.* **16**, 873–888 (2000)
29. Gutjahr, W.J.: ACO algorithms with guaranteed convergence to the optimal solution. In: *Process. Lett.* **82**, 145–153 (2002)
30. Gutjahr, W.J.: A converging ACO algorithm for stochastic combinatorial optimization. Proceedings of 2nd Symposium on Stochastic Algorithms, Foundations and Applications, Springer LNCS, Berlin Heidelberg vol. **2827**, pp. 10–25 (2003)
31. Gutjahr, W.J.: S-ACO: An ant-based approach to combinatorial optimization under uncertainty. Proceedings of 4nd Int. Workshop on Ant Colony Optimization and Swarm Intelligence, Springer LNCS, Berlin Heidelberg New York vol. **3172**, pp. 238–249 (2004)
32. Gutjahr, W.J.: On the finite-time dynamics of ant colony optimization. *Methodol. Comput. Appl. Probability* **8**, 105–133 (2006)
33. Gutjahr, W.J.: Mathematical runtime analysis of ACO algorithms: survey on an emerging issue. *Swarm Intell.* **1**, 59–79 (2007)
34. Gutjahr, W.J.: First steps to the runtime complexity analysis of ant colony optimization. *Comput. Oper. Res.* **35**, 2711–2727 (2008)
35. Gutjahr, W.J., Katzensteiner, S., Reiter, P.: A VNS algorithm for noisy problems and its application to project portfolio analysis. Proceedings of SAGA 2007 (Stochastic Algorithms: Foundations and Applications), Springer LNCS, Berlin Heidelberg vol. **4665**, pp. 93–104 (2007)
36. Gutjahr, W.J., Pfleg, G.: Simulated annealing for noisy cost functions. *J. Global Optim.* **8**, 1–13 (1996)
37. Gutjahr, W.J., Sebastiani, G.: Runtime analysis of ant colony optimization with best-so-far reinforcement. *Methodology and Computing in Applied Probability* **10**, 409–433 (2008)
38. Hajek, B.: Cooling schedules for optimal annealing. *Math. of Operat. Res.* **13**, 311–329 (1988)
39. Hartl, R.F.: A global convergence proof for a class of genetic algorithms. Technical Report, University of Vienna (1990)
40. Hartmann, A.K., Barthel, W., Weigt, M.: Phase transition and finite-size scaling in the vertex-cover problem. *Comput. Phys. Commun.* **169**, 234–237 (2005)
41. He, J., Yao, X.: Drift analysis and average time complexity of evolutionary algorithms. *Artif. Intell.* **127**, 57–85 (2003)
42. He, J., Yao, X.: Towards an analytic framework for analysing the computation time of evolutionary algorithms. *Artif. Intell.* **145**, 59–97 (2003)
43. He, J., Yao, X.: A study of drift analysis for estimating computation time of evolutionary algorithms. *Nat. Comput.* **3**, 21–35 (2004)
44. He, J., Yu, X.: Conditions for the convergence of evolutionary algorithms. *J. Syst. Arch.* **47**, 601–612 (2001)
45. Herroelen, W., De Reyck, B.: Phase transitions in project scheduling. *J. Oper. Res. Soc.* **50**, 148–156 (1999)
46. Igel, C., Toussaint, M.: On classes of functions for which no free lunch results hold. *Inf. Process. Lett.* **86**, 317–321 (2003)
47. Igel, C., Toussaint, M.: A no-free-lunch theorem for non-uniform distributions of target functions. *J. Math. Model. Algorithms* **3**, 313–322 (2004)

48. Jacobson, S.H., Yücesan, E.: Analyzing the performance of generalized hill climbing algorithms. *J. Heuristics* **10**, 387–405 (2004)
49. Jansen, T., Wegener, I.: On the analysis of a dynamic evolutionary algorithm. *J. Discrete Algorithms* **4**, 181–199 (2006)
50. Jin, Y., Branke, J.: Evolutionary optimization in uncertain environments – a survey. *IEEE Trans. Evol. Comput.* **9**, 303–317 (2005).
51. Kennedy, J., Eberhart, R.C.: A discrete binary version of the particle swarm algorithm. *Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics*, Orlando, FL, USA pp. 4104–4109 (1997)
52. Koehler, G.J.: Conditions that obviate the no-free-lunch theorems for optimization. *Informs J. Comput.* **19**, 273–279 (2007)
53. Ladret, V.: Asymptotic hitting time for a simple evolutionary model of protein folding. *J. Appl. Probability* **42**, 39–51 (2005)
54. Margolin, L.: On the convergence of the cross-entropy method. *Ann. Oper. Res.* **134**, 201–214 (2005)
55. Martin, O.C., Monasson, R., Zecchina, R.: Statistical mechanics methods and phase transitions in optimization problems. *Theor. Compu. Sci.* **265**, 3–67 (2001)
56. Merelo, J.-J., Cotta, C.: Building bridges: the role of subfields in metaheuristics. *SIGEVOlution* **1**(4), 9–15 (2006)
57. Mertens, S.: A physicist’s approach to number partitioning. *Theor. Comput. Sci.* **265**, 79–108 (2001)
58. Merz, P., Freisleben, B.: Fitness landscape analysis and memetic algorithms for the quadratic assignment problem. *IEEE Trans. Evol. Comput.* **4**, 337–352 (2000)
59. Monasson, R.: Introduction to phase transitions in random optimization problems. Technical Report, Laboratoire de Physique Théorique de l’ENS, Paris (2007)
60. Neumann, F., Wegener, I.: Randomized local search, evolutionary algorithms, and the minimum spanning tree problem. *Theor. Comp. Sci.* **378**, 32–40 (2007).
61. Neumann, F., Witt, C.: Runtime analysis of a simple ant colony optimization algorithm. *Proceedings of ISAAC ’06*, Springer LNCS, Berlin Heidelberg vol. **4288**, pp. 618–627 (2006)
62. Neumann, F., Witt, C.: Ant colony optimization and the minimum spanning tree problem. *Proceedings of LION ’08, Learning and Intelligent Optimization, Theoretical Computer Science* **411**, 2406–2413 (2010)
63. Norman, F.: *Markov Processes and Learning Models*. Academic Press, New York (1972)
64. Oliveto, P.S., He, J., Yao, X.: Time complexity of evolutionary algorithms for combinatorial optimization: a decade of results. *Int. J. Automat. and Comput.* **4**, 281–293 (2007)
65. Oliveto, P.S., He, J., Yao, X.: Evolutionary algorithms and the vertex cover problem. *Proceedings of the Congress on Evolutionary Computation CEC ’07*, Singapore pp. 1870–1877 (2007)
66. Purkayastha, P., Baras, J.S.: Convergence results for ant routing algorithms via stochastic approximation and optimization. *Proceedings of 46th IEEE Conference on Decision and Control*, pp. 340–345 (2007)
67. Reidys C.M., Stadler, P.F.: Combinatorial landscapes. *SIAM Rev.* **44**, 3–54 (2002)
68. Rudolph, G.: Convergence Analysis of canonical genetic algorithms. *IEEE Trans. Neural. Netw.* **5**, 96–101 (1994)
69. Sasaki, G.H., Hajek, B.: The time complexity of maximum matching by simulated annealing. *J. ACM* **35**, 67–89 (1988)
70. Scharnow, J., Tinnefeld, K., Wegener, I.: Fitness landscapes based on sorting and shortest path problems. *Proceedings of 7th Conference on Parallel Problem Solving from Nature*, 54–63 (2002)
71. Schiavinotto, T., Stützle, T.: A review of metrics on permutations for search landscape analysis. *Comput. Oper. Res.* **34**, 3143–3153 (2007)
72. Sebastiani, G., Torrisi, G.L.: An extended ant colony algorithm and its convergence analysis. *Methodol. Comput. Appl. Probability* **7**, 249–263 (2005)

73. Storch, T.: How randomized search heuristics find maximum cliques in planar graphs. Proceedings of 8th Annual Conference on Genetic and Evolutionary Computation, Seattle, Washington, USA pp. 567–574 (2006)
74. Stützle, T., Hoos, H.H.: MAX-MIN Ant System. Future Gen. Comput. Sys. **16**, 889–914 (2000)
75. Stützle, T., Dorigo, M.: A short convergence proof for a class of ACO algorithms. IEEE Trans. Evol. Comput. **6**, 358–365 (2002)
76. Sudholt, D., Witt, C.: Runtime analysis of binary PSO. Proceedings of 10th Annual Conf. on Genetic and Evolutionary Computation, New York, USA pp. 135–142 (2008)
77. Teytaud, O., Gelly, S.: General lower bounds for evolutionary algorithms. Proceedings of 9th Conference on Parallel Problem Solving from Nature, pp. 21–31 (2006)
78. Trelea, I.C.: The particle swarm optimization algorithm: convergence analysis and parameter selection. Inf. Process. Lett. **85**, 317–325 (2003)
79. Vaughan, D.E., Jacobson, S.H., Kaul, H.: Analyzing the performance of simultaneous generalized hill climbing algorithms. Comput. Optim. Appl. **37**, 103–119 (2007)
80. Wegener, I.: Simulated annealing beats metropolis in combinatorial optimization. Proceedings of ICALP '05, Springer LNCS, Berlin Heidelberg vol. **3580**, pp. 589–601 (2005)
81. Wegener, I., Witt, C.: On the analysis of a simple evolutionary algorithm on quadratic pseudo-boolean functions. J. Discrete Algorithms **3**, 61–78 (2005)
82. Whitley, D., Watson, J.P.: Complexity theory and the no free lunch theorem. In: Burke, E.K., Kendall, G. (eds.) Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques, Kluwer, Boston, 317–399 (2005)
83. Witt, C.: Worst-case and average-case approximations by simple randomized search heuristics. Proceedings of 22nd Annual Symposium on Theoretical Aspects of Computer Science, Springer LNCS, Berlin Heidelberg vol. **3404**, pp. 44–56 (2005)
84. Witt, C.: Runtime analysis of the  $(\mu + 1)$  EA on simple pseudo-bolan functions. Proceedings of 8th Annual Conference on Genetic and Evolutionary Computation, Seattle, Washington, pp. 651–658 (2006)
85. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Trans. Evol. Comput. **1**, 67–82 (1997).
86. Yao, A.C.: Probabilistic computations: towards a unified measure of complexity. Proceedings of 17th IEEE Symposium on the Foundations of Computer Science, 222–227 (1977)
87. Yu, Y., Zhou, Z.-H.: A new approach to estimating the expected first hitting time of evolutionary algorithms. Proceedings of 21th National Conference on Artificial Intelligence, Boston, MA, 555–560 (2006)
88. Zhang, W.: Phase transitions and backbones of the asymmetric travelling salesman problem. J. Artif. Intell. Res. **21**, 471–497 (2004)
89. Zlochin, M., Birattari, M., Meuleau, N., Dorigo, M.: Model-based search for combinatorial optimization: a critical survey. Ann. Oper. Res. **131**, 373–379 (2004)



# Chapter 20

## An Introduction to Fitness Landscape Analysis and Cost Models for Local Search

Jean-Paul Watson

**Abstract** Despite their empirical effectiveness, our theoretical understanding of metaheuristic algorithms based on local search (and all other paradigms) is very limited, leading to significant problems for both researchers and practitioners. Specifically, the lack of a theory of local search impedes the development of more effective metaheuristic algorithms, prevents practitioners from identifying the metaheuristic most appropriate for a given problem, and permits widespread conjecture and misinformation regarding the benefits and/or drawbacks of particular metaheuristics. Local search metaheuristic performance is closely linked to the structure of the fitness landscape, i.e., the nature of the underlying search space. Consequently, understanding such structure is a first step toward understanding local search behavior, which can ultimately lead to a more general theory of local search. In this chapter, we introduce and survey the literature on fitness landscape analysis for local search, placing the research in the context of a broader, critical classification scheme delineating methodologies by their potential to account for local search metaheuristic performance.

### 20.1 Introduction

Despite widespread success, very little is known about *why* local search metaheuristics work so well and under what conditions. This situation is largely due to the fact that researchers typically focus on demonstrating, and not analyzing, algorithm performance. Most local search metaheuristics are developed in an ad hoc manner. A researcher devises a new search strategy or a modification to an existing strategy, typically arrived at via intuition. The algorithm is implemented, and the resulting performance is compared with that of existing algorithms on sets of widely available

---

Jean-Paul Watson

Sandia National Laboratories, Discrete Math and Complex Systems Department, P.O. Box 5800  
MS 1318, Albuquerque, New Mexico, USA 87185-1318  
e-mail: jwatson@sandia.gov

benchmark problems. If the new algorithm outperforms existing algorithms, the results are published, advancing the state of the art. Unfortunately, most researchers (including, often, the author) fail to actually prove that the proposed enhancement(s) actually led to the observed performance increase (as typically, multiple new features are introduced simultaneously) or whether the increase was due to fine tuning of the algorithm or associated parameters, implementation tricks, flaws in the comparative methodology, or some other factor(s). Hooker [10] refers to this approach to algorithm development as the *competitive testing* paradigm, arguing that “this *modus operandi* spawns a host of evils that have become depressingly familiar to the algorithmic research community” and that the “emphasis on competition is fundamentally anti-intellectual” [10], (p. 33). However, although few would argue with Hooker’s criticisms, the competitive testing paradigm remains the dominant paradigm for developing new algorithms—*independent of whether they are exact or approximate, or based on constructive or local search.*

Due largely to the widespread practice of competitive testing, theoretical results concerning the operation of local search metaheuristics are very limited. In particular, we currently lack fundamental models of local search metaheuristic behavior. The importance of behavioral models cannot be understated. Ideally, behavioral models enable practitioners to identify those problems for which a particular local search metaheuristic is likely to be effective and those problem instances that are likely to be more difficult than others. The broad availability of behavioral models would enable researchers to identify fundamental similarities and differences between different local search metaheuristics and identify new research directions in order to improve the performance of existing local search algorithms. In contrast, the current lack of behavioral models has led to several undesirable side-effects, including widespread conjecture and mythology regarding the benefits and/or operation of particular local search metaheuristics.

A foundational concept in the development of behavioral models of local search metaheuristics is the notion of a *fitness landscape*, i.e., the topological structure over which search is being executed. Given a specific landscape structure—defined by a search space, objective function, and neighborhood operator, a local search metaheuristic can be viewed as a strategy for navigating this structure in the search for optimal or near-optimal solutions. Given this context, the effectiveness of a particular metaheuristic is a function of how “well tuned” the navigation strategy is to the given landscape. Consequently, knowledge of the fitness landscape structure is key to developing effective metaheuristics and consequently has been a primary focus in the theoretical analysis of local search metaheuristics. The objective of this chapter is to survey the prior research on fitness landscape analysis for local search metaheuristics and assess the potential of these efforts to explain one or more aspects of metaheuristic behavior/performance.

To facilitate focus and brevity, our emphasis is on local search metaheuristics, e.g., tabu search, simulated annealing, and variable neighborhood search. These algorithms, which iteratively modify a single solution via repeated perturbations, contrast to both constructive (e.g., GRASP and ant colony optimization) and evolutionary (e.g., genetic algorithms and genetic programming) metaheuristics. However, fitness landscape analysis does play a role in these paradigms as well. For example,

local search is widely used in both GRASP and ant colony optimization to post-process solutions generated by the core constructive procedures. Similarly, fitness landscape analysis has long been central in genetic algorithm theory, although the analysis is significantly complicated by the presence of a population of solutions and multi-parent recombination. Where appropriate, we cite relevant literature in both of these sub-fields. Subsequently, where there is no risk of confusion, we refer to local search metaheuristics simply as metaheuristics.

We begin in Section 20.2 by formally introducing the concept of a fitness landscape and a local search metaheuristic. Next, we introduce in Section 20.3 a classification scheme for models of local search metaheuristic behavior, specifically focusing on the objectives of any particular analysis; such objectives are often implicit or unspecified in the literature, and making the objectives explicit allows for an assessment of the ultimate utility of a given methodology. We survey the broad range of landscape structures identified by various researchers in Section 20.4, discussing their role in, and ability to account for, metaheuristic performance. Most models of local search metaheuristic performance are implicit, in that they do not propose specific models of metaheuristic run-time dynamics. In Section 20.5, we survey the limited exceptions—which are exemplars of the types of models that ultimately will inform a rigorous theory of local search. Finally, we conclude in Section 20.6.

## 20.2 Combinatorial Optimization, Local Search, and the Fitness Landscape

We begin by formally defining a combinatorial optimization problem (COP), before introducing the concepts of a fitness landscape and a local search metaheuristic. First, we explicitly differentiate a *problem* (e.g., Boolean satisfiability) from a *problem instance* (e.g., a 100-variable, 300-clause instance of random k-SAT with  $k = 3$ ). We denote a combinatorial optimization problem by  $\Pi$  and an instance of  $\Pi$  by  $\Omega$ .  $\Omega$  is drawn from some (possibly infinite) universe of possible problem instances, which we denote  $U_\Pi$ .

An instance  $\Omega$  of a combinatorial optimization problem  $\Pi$  is fully specified by two components: the state space and the objective function. The state space  $S$  is a finite, although typically astronomically large, set of possible solutions to  $\Omega$ . The objective function  $F$  assigns a numeric “worth” to each state  $s \in S$ . The only formal restriction on  $F$  is that there must exist a total order of the co-domain, such that a maximal or minimal value is well defined. Typically,  $F : S \rightarrow R^+$  or  $F : S \rightarrow Z^+$ . The objective function is commonly referred to as a *fitness function*.

Given a COP instance  $\Omega$ , the ultimate objective is to locate a solution  $s \in S$  such that  $F(s)$  is optimal, i.e., minimal or maximal. Without loss of generality, we assume the objective is minimization unless otherwise noted.

Local search proceeds via iterative modifications to complete solutions  $s \in S$ , in contrast to constructive and population-based metaheuristics. We further restrict our attention to the subset of local search metaheuristics that operate via iterative modifications to a *single* complete solution  $s$ , i.e., we ignore more complicated local

search metaheuristics that employ elite pools and strategies such as optima linking [30] and path relinking [8]. In doing so, our goal is pragmatic: to discuss the state of local search theory with respect to the simplest (albeit still effective) class of local search metaheuristic before tackling more complex powerful derivatives.

All single-solution local search metaheuristics consist of the following four core components: the state space, the objective function, the move operator, and the navigation strategy. Search begins from an initial solution  $s = s_{init}$  that is generated either at random or via some heuristic procedure and proceeds via a sequence of iterations. The *move operator* specifies the set of allowable modifications (i.e., the neighborhood) to the current solution  $s$  at any given iteration, one of which is selected by the *navigation strategy* to serve as the basis for the next iteration. The best solution encountered in any iteration is stored and returned when the algorithm terminates, typically after a time limit is exceeded or maximal number of iterations is completed. We now explore each of these four components in more detail, providing simple examples of each as applied to both the well-known Traveling Salesman Problem (TSP) and Maximum Satisfiability Problem (MAX-SAT).

### **20.2.1 The State Space and the Objective Function**

Both the state space  $S$  and the objective function  $F$  are taken directly from  $\Omega$ , the problem instance under consideration. For example, in an  $n$ -city TSP instance, the state space consists of the set of  $n!$  permutations, each representing a possible tour; the objective function simply returns the total length of the input tour. In an  $n$ -variable,  $m$ -clause MAX-SAT instance, the state space consists of the set of  $2^n$  Boolean vectors of length  $n$ ; the objective function returns the number of the  $m$  clauses that are satisfied in a solution  $s \in S$ . In general, although we do not consider the issue here, the details of how solutions are represented can impact the definition of both the move operator and the navigation strategy. However, this is typically not the case for many well-known combinatorial optimization problems—including the TSP and MAX-SAT.

### **20.2.2 The Move Operator**

Given a state space  $S$ , the notion of locality in a local search algorithm is provided by the move or neighborhood operator  $N$ .  $N$  defines the set of allowable modifications to the current solution  $s$  in any given iteration. In single-solution local search algorithms,  $N : S \rightarrow P(S)$ , where  $P(S)$  denotes the power set of  $S$ . More complicated move operators, e.g., those whose domain and/or codomain are cross products of  $S$  and  $P(S)$ , respectively, are found primarily in evolutionary algorithms and other related approaches such as optima linking, path relinking, and scatter search; see [14] for a discussion of these and related issues. Given a solution  $s$ , the set  $N(s)$  is known as the *neighborhood* of  $s$ . Similarly, if  $s' \in N(s)$ , then  $s'$  is said to be a *neighbor* of  $s$ .

Local search metaheuristics often employ rather simple move operators. For example, the most widely used move operator for MAX-SAT maps each solution  $s \in S$

to the subset of  $n$  solutions (where  $n$  is the total number of Boolean variables) in  $S$  that differ from  $s$  in the value of a single variable assignment; this is known as the “1-flip” neighborhood. Similarly, most local search metaheuristics for the TSP are based in part on the 2-opt move operator [19], where the neighbors of a solution  $s \in S$  are defined as the subset of  $\binom{n}{2}$  solutions in  $S$  obtained by inverting the subtour between any pair of distinct cities on the tour specified by  $s$ . More complex move operators can be obtained via straightforward generalization of these basic operators, e.g.,  $k$ -flip move operators in MAX-SAT and  $k$ -opt move operators in the TSP.

Move operators can vary significantly in their attempts to maintain “logical” locality. Both the 1-flip and 2-opt move operators induce minimal disruptions to the current solution  $s$ : 1-flip inverts the value of a single Boolean variable, while 2-opt changes exactly 2 edges. However, in local search algorithms such as iterated local search [20], the differences between neighboring solutions can be much more substantial, e.g., under the generalized  $k$ -opt move operator for the TSP [13]. In both cases, however, the perturbation is local in the sense that a neighboring solution is obtained via a *single* application of a move operator. We raise this issue to note that a “local” search metaheuristic may in fact proceed by making drastic modifications to individual solutions.

Mathematically, the move operator  $N$  induces a relation on the space  $S \times S$ , and the properties of this relation can influence the performance of local search. For simplicity, we refer to the relation induced by  $N$  simply as  $N$ . Although both the 1-flip and 2-opt move operators are symmetric, in that  $s' \in N(s) \Rightarrow s \in N(s')$ , this is generally not required. Further,  $N$  is necessarily transitive and anti-reflexive. Beyond defining the immediate neighborhood, a move operator also defines the connectivity of the search space, i.e., what solutions can be reached via a finite sequence of moves from an initial solution. A move operator  $N$  is said to induce a *connected* search space if from an arbitrary solution there always exists a sequence of moves to an optimal solution.  $N$  is said to induce a *fully connected* search space if there exists a sequence of moves between any two arbitrary solutions. Both the 1-flip and 2-opt move operators induce fully connected, and consequently connected, search spaces. However, many powerful, problem-specific move operators induce disconnected search spaces, e.g., see [25].

### 20.2.3 The Navigation Strategy

The mechanism for selecting some neighbor  $s' \in N(s)$  at each iteration of local search is embodied in the navigation strategy, which we denote by  $\Delta$ . One of the simplest navigation strategies follows the basic principle of gravity: select a neighbor  $s' \in N(s)$  with  $F(s') < F(s)$ . Two well-known variants of this greedy strategy form the core of nearly all navigation strategies. In *next-descent* search, the neighbors  $N(s)$  are randomly ordered, and the first neighbor  $s' \in N(s)$  such that  $F(s') < F(s)$  is selected. In *steepest-descent* search, the neighbor that provides the maximal decrease in the objective function value ( $\operatorname{argmin}_{s' \in N(s)} F(s')$ ) is selected, with ties broken randomly.

By iterating greedy descent, local search will eventually arrive at a solution  $s \in S$  from which no immediate improvement in the value of the objective function is possible;  $s$  is known as a local optimum, such that  $\forall s' \in N(s), F(s') \geq F(s)$ . Unless  $s$  is also a globally optimal solution, the navigation strategy must then guide search to unexplored regions of the search space. When there exists a neighbor  $s' \in N(s)$  such that  $F(s) = F(s')$ ,  $s$  is actually contained in a plateau, which may or may not be locally optimal; this issue is discussed further in Section 20.2.4.

Within the local search community, strategies for escaping or avoiding local optima are commonly referred to as *metaheuristics*. Formally, a metaheuristic is a heuristic that dynamically alters the behavior of a core local search heuristic, typically in response to the properties of recently visited solutions. In most local search metaheuristics, the core heuristic is greedy descent; a metaheuristic is activated when the descent strategy becomes trapped in a local optimum, and deactivated once search is successfully directed toward new regions of the search space. Conceptually, metaheuristics and greedy descent are distinct forms of navigation strategies, each operating at a different level of abstraction. However, in practice, the boundary between the two is often fuzzy, for example, in simulated annealing. Consequently, metaheuristics are often viewed as atomic entities, such that the distinction between the core heuristic and the metaheuristic is ignored. We present them as such, while at the same time acknowledging any intended distinction between the core and metaheuristic.

Perhaps the most obvious way to escape a local optimum is to generate a new starting solution  $s_{init}$  and then re-initiate greedy descent. This process can be iterated until a global optimum is located. The resulting metaheuristic is commonly referred to as *iterated descent*, which is distinct from the next-descent and steepest-descent procedures. In practice, iterated descent is a simple way to improve the performance of a core greedy descent strategy. Further, iterated descent can locate very high-quality solutions for some combinatorial optimization problems, e.g., see Beveridge et al. [2].

Clearly, the probability of iterated descent locating a global optimum approaches 1 as the number of greedy descents approaches  $\infty$ . However, from a practical standpoint, iterated descent is only effective if the fitness distribution of the local optima assumes a certain form, i.e., one in which the left tail of the distribution is non-negligible. For many well-known combinatorial optimization problems, the fitness distribution of local optima in small problem instances satisfies this requirement. At the same time, it has been empirically demonstrated that such tails typically vanish at larger problem sizes (for example in the TSP), causing iterated descent to perform poorly due to what has come to be known as a “central limit catastrophe” [13].

#### 20.2.4 The Fitness Landscape

Given a local search metaheuristic  $A$  and a combinatorial optimization problem  $\Pi$ , we are interested in determining what makes a particular instance  $\Omega \in U_\Pi$  easy or difficult for  $A$ . Problem difficulty, or equivalently search cost, is dictated by the

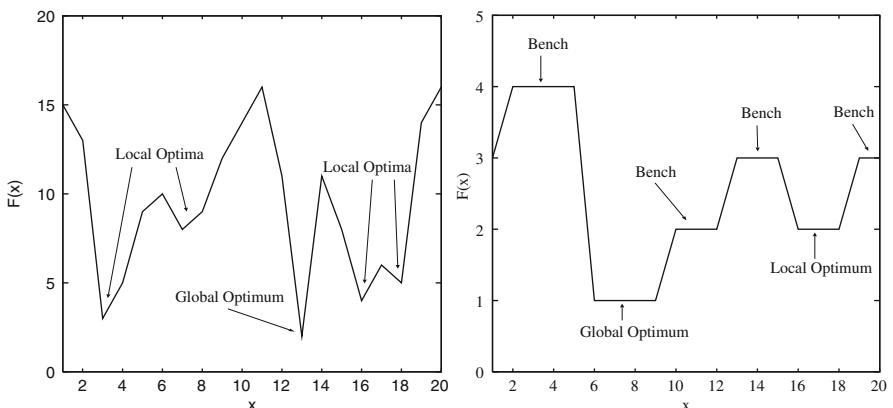
interaction of  $A$  with the underlying search space. For example, suppose all globally optimal solutions to  $\Omega$  reside in a small region of the search space containing otherwise poor local optima. If  $A$  consistently biases search toward regions of the search space containing generally high-quality local optima, then the cost (on average) of locating optimal solutions to  $\Omega$  using  $A$  is likely to be large. In contrast, if  $A$  intensifies search in regions of the search space with poor local optima, then  $A$  is more likely to locate optimal solutions to  $\Omega$  in shorter run-times.

Due to the central role of the search space in determining problem difficulty, much of the research on models of problem difficulty for local search has concentrated identifying structural features of the search space that are likely to influence the cost of local search. Given a local search metaheuristic  $A$ , the search space is defined by the combination of (1) the state space  $S$ , (2) the move operator  $N$ , and (3) the objective function  $F$ . Formally, we define the search space  $L = (S, N, F)$  as a vertex-weighted directed graph  $G = (V, E)$  in which

1.  $V = S$
2.  $\forall v \in V$ , the weight  $w_v$  of  $v$  is equal to  $F(v)$
3.  $E = \{(i, j) | i \neq j \wedge \exists i, j : i \in N(j)\}$ .

Within the local search community, the graph  $G$  is known as a *fitness landscape*, a concept first introduced by the theoretical biologist Sewall Wright in 1932 [44].

We provide two examples of very simple fitness landscapes in Figure 20.1; in general, landscapes are high dimensional and extremely difficult to visualize. In both examples,  $S = \{1, 2, \dots, 20\}$  and  $N(x) = \{x - 1, x + 1\}$ , subject to the boundary conditions  $N(1) = \{20, 2\}$  and  $N(20) = \{19, 1\}$ . *Type I* fitness landscapes are characterized by deep, punctuated valleys with abrupt changes in the fitness of neighboring solutions. In contrast, *Type II* fitness landscapes are dominated by plateaus of equally fit neighboring solutions, with discrete jumps in fitness between the plateaus. We differentiate between the two types of fitness landscapes for three reasons. First, different terminology is associated with the two landscape types.



**Fig. 20.1** Examples of Type I (left figure) and Type II (right figure) fitness landscapes.

Second, Type I and Type II landscapes have different implications for the design of metaheuristic navigation strategies. Third, these two types are representative of the fitness landscapes found in most *NP*-hard optimization problems. For example, the TSP and MAX-SAT, respectively, possess Type I and Type II fitness landscapes.

In a Type I fitness landscape, the two key features of interest are local optima and global optima. A *local optimum* is a point  $x \in S$  such that  $\forall y \in N(x), F(x) \leq F(y)$ . In our example Type I landscape, the following vertices are local optima: 3, 7, 13, 16, and 18. A *global optimum* is a point  $x \in S$  that is both locally optimal and  $\forall y \in S, F(x) \leq F(y)$ . In our example Type I landscape, vertex 13 is the sole global optimum. The *attractor basin* of a local optimum  $s$  consists of all  $s' \in S$  such that  $s$  results with non-zero probability when a descent-based procedure is applied to  $s'$ ; as first noted by Reeves [31], attractor basin membership may be stochastic due to the different forms of randomization commonly associated with descent procedures.

Plateaus are the dominant feature of Type II fitness landscapes. Informally, a plateau is simply an interconnected region of the fitness landscape where all points have equal fitness. Formally, a plateau is defined as a set  $P \subseteq S$  such that

1.  $\forall x \in P, F(x) = C$  for some constant  $C$
2. For any two points  $x, y \in P$  there is a sequence of solutions  $\{x, a_1, \dots, a_n, y\}$  such that  $\forall i, a_i \in S$  and  $F(x) = F(a_1) = \dots = F(a_n) = F(y) = C$
3. (a)  $a_1 \in N(x)$ , (b)  $\forall i \neq n - 1, a_{i+1} \in N(a_i)$ , and (c)  $y \in N(a_n)$

If for some  $x \in P$  there exists a  $y \in N(x)$  such that  $F(y) < C$ , the plateau is called a *bench*, and all such solutions  $y$  are called *exits*. If there are no exits from a plateau, then the plateau is locally optimal. If the plateau is locally optimal and  $\forall x \in S, C \leq F(x)$ , then the plateau is also globally optimal. All benches, local optima, and global optima are labeled in our example Type II fitness landscape. There are many additional nuances regarding the terminology of features found in Type II fitness landscapes; an overview is provided by Frank et al. [7].

The qualitative differences between Type I and Type II fitness landscapes have an important impact on the design of navigation strategies and metaheuristics for local search. For example, both next-descent and steepest-descent typically terminate once a solution  $s$  is located with no lower fitness neighbors. The implicit, built-in assumption is that the local optimum  $s$  is not a member of a plateau, or that if  $s$  is a member of a plateau, then the plateau itself is locally optimal. In general, these assumptions do not hold when dealing with Type II fitness landscapes; if greedy descent terminates at a local optimum, it is possible that the optimum resides on a bench, from which an exit may exist. Additionally, the attractor basins in Type II fitness landscapes are often very shallow. For example, Frank et al. [7] have shown that in MAX-SAT, it is often possible to escape a local optimum by accepting a single non-improving move. Consequently, the emphasis on navigation strategies in Type II fitness landscapes is on moving quickly from one plateau to another, either by finding an exit from a bench or by temporarily accepting non-improving moves. In contrast, in Type I fitness landscapes the emphasis is on escaping local optima with potentially large and deep attractor basins.

## 20.3 Landscape Analysis and Cost Models: Goals and Classification

As discussed previously, most research on local search focuses on developing newer, better performing algorithms. The goal in such research is clearly to *demonstrate* algorithm performance. Paul Cohen notes in his book *Empirical Methods for Artificial Intelligence* ([5], p. 249) that “It is good to demonstrate performance, but it is even better to *explain* [emphasis added] performance.” The hard sciences advance via the development of accurate models of the object or objects of interest, models that are both consistent with existing observations and suggest new behavioral hypotheses. Currently, models of local search metaheuristics for *any* combinatorial optimization problem are rare to non-existent.

In developing a model of a given object, we generally concentrate on capturing specific behaviors or small sets of behaviors. In the context of local search metaheuristics, the behavior of interest is generally the cost required to locate an optimal solution (or, more generally, a solution with a given quality threshold) to a problem instance. Due to the stochastic nature of local search (with the sole noted exception of some variants of tabu search), search cost is a random variable with a particular distribution. *Cost models* of local search metaheuristics are behavioral models that capture various aspects of the cost distribution. Most often, we focus on the *average* or typical search cost, as defined by either the distribution mean or median. It is well known that given a fixed problem size (e.g., 100-city TSPs), the average search cost across instances can vary by many orders of magnitude. One objective in developing cost models is to account for a significant proportion, and ideally all, of this variability. A more aggressive, penultimate objective is to develop cost models that account for the entire distribution of search cost.

In this section, we discuss a general classification scheme for cost models of local search metaheuristics. We consider three different types of cost models, differing in both the type of information upon which they are based and the extent to which they attempt to explicitly capture metaheuristic run-time dynamics. *Static* cost models (Section 20.3.1) are functions of one or more features of the fitness landscape and only implicitly consider metaheuristic dynamics. In contrast, *quasi-dynamic* and *dynamic* cost models (Sections 20.3.2 and 20.3.3, respectively) are based on analyses of metaheuristic run-time behavior. Quasi-dynamic cost models are functions of simple summary statistics of metaheuristic behavior. In contrast, dynamic cost models explicitly model low-level metaheuristic behavior using Markov chains.

### 20.3.1 Static Cost Models

*Static* cost models are strictly based on fitness landscape features; metaheuristic dynamics are completely and explicitly ignored. In a static cost model, the independent variables are fitness landscapes features, or combinations thereof, and the dependent variable is the mean or median search cost. To facilitate model evaluation, static cost models are expressed as linear or multiple regression models. Under this

formulation, the accuracy of a static cost model can be naturally quantified as the  $r^2$  value of the corresponding regression model, i.e., the proportion of the total variability accounted for by the model. Most static cost model considered to date are based on a single feature of the fitness landscape. For purposes of brevity, we often denote a static cost model based on the feature  $X$  as the  $X$  static cost model or simply the  $X$  model. Similarly, given the close relationship between static cost models and regression models, we frequently use the two terms interchangeably. Finally, regression methods make certain assumptions (e.g., model errors are homogeneous across the range of the independent variable) in order to generate valid statistical inferences concerning model parameters. These assumptions are generally not satisfied in metaheuristic research. The motivation in using regression models is to (1) quantify overall model accuracy using the associated  $r^2$  value and (2) analyze worst-case deviations from a predicted/expected value. Failure to satisfy regression assumptions does not impact our ability to achieve either of these objectives.

The quality of a static cost model is tied to the model  $r^2$ : models with larger  $r^2$  values are more accurate. However, there are limits on the absolute level of accuracy that we can reasonably expect to achieve. As discussed in Section 20.4, the most accurate static cost models of local search only yield  $r^2 \approx 0.25$  in the worst case, which is typically observed for the most difficult sets of problem instances. Although failure to develop more accurate static cost models, despite intense research effort, is not evidence for their impossibility, there does appear to be a practical limit on what can be achieved. Because static cost models ignore metaheuristic dynamics, the existence of models with even  $r^2 \approx 0.5$  is in some sense surprising. In expressing fitness landscape features as atomic numeric quantities, there is also the obvious potential for loss of information. Further, there are practical (although not theoretical) limits on the accuracy with which we can measure various quantities, including search cost.

### 20.3.2 Quasi-dynamic Cost Models

A first-order approach to improving static cost models is to incorporate coarse-grained information concerning metaheuristic run-time behavior. For example, we might track simple summary statistics that capture defining characteristics of the set of solutions generated by a metaheuristic. Given such summary statistics, we can then construct regression models relating these summary statistics to search cost, and quantify model accuracy as the resulting  $r^2$ . We refer to such cost models as *quasi-dynamic* cost models. The “quasi-dynamic” modifier derives from the fact that the model is based on aggregate statistics relating to run-time behavior, as opposed to an explicit model of metaheuristic run-time dynamics. The sole difference between static and quasi-dynamic cost models is in the nature of the information captured in the independent variable(s).

Most of the issues relating to possible limitations on the accuracy of static cost models equally apply to quasi-dynamic cost models. However, because they account for some aspects of run-time behavior, we would expect in some sense the

accuracy of quasi-dynamic cost models to be higher than that of static cost models, although less than the fully dynamic cost models considered below. Empirical evidence supports this observation: some of the most accurate cost models of local search metaheuristics developed to date are quasi-dynamic [35] and achieve a worst case accuracy of  $r^2 \approx 0.65$ .

### 20.3.3 Dynamic Cost Models

Because they are respectively based on fitness landscape features and summary statistics of run-time behavior, static and quasi-dynamic cost models yield no *direct* insight into the dynamical behavior of local search. To gain insight as to why particular landscape or run-time statistics are highly correlated with search cost, we turn to *dynamic* cost models. Dynamic cost models are high-resolution models (e.g., Markov models) of the run-time behavior of local search metaheuristics. Research on dynamic cost models can be traced to Hoos [11], who used Markov models to posit an explanation for certain run-time behaviors observed for Walk-SAT and other local search algorithms in the Random 3-SAT phase transition region. However, the ability of these models to account for variability in problem difficulty was not considered.

To date, dynamic cost models are represented as Markov chains which coarse-grain the search space in some way. In one common approach, each state of the Markov chain captures the distance  $i$  to the nearest *target* (e.g., optimal) solution, in addition to other algorithm-specific attributes. Transitions in the Markov chain correspond to iterations of the local search metaheuristic. A dynamic cost model is constructed by specifying a set of states and then estimating the various transition probabilities between the states. The details of the estimation process are algorithm dependent. The search cost predicted by a dynamic cost model is defined as the mean number of iterations until an absorbing state (i.e., a state with  $i = 0$ ) is encountered. For some Markov chains, analytic formulas for the mean time to absorption are easily derived. When analytic formulas are not immediately available, it is pragmatic to resort to simulation of the cost model to estimate mean search cost.

To quantify the accuracy of a dynamic cost model, straightforward linear regression models can be used, in which the predicted and actual search costs serve as the independent and dependent variables, respectively; model accuracy is then quantified by the  $r^2$  value of the linear model. Dynamic cost models differ from their static counterparts in that they explicitly consider the metaheuristic and move beyond simple numeric characterizations of either fitness landscape features or run-time behavior. Consequently, we a priori anticipate higher levels of accuracy than are possible for static and quasi-dynamic cost models. This conjecture is supported in practice;  $r^2$  values in excess of 0.90 are reported in the literature. However, the near-perfect accuracy does not come without costs: dynamic cost models are generally more expensive to construct than static or quasi-dynamic cost models and are generally far less intuitive.

### ***20.3.4 Descriptive Versus Predictive Cost Models***

For all practical purposes, the cost models we discuss are purely descriptive, in that they provide *a posteriori* explanations for why one problem instance is more difficult than another for a given local search metaheuristic. In principle, cost models could be used to compute a relatively tight confidence interval, via standard regression techniques, for the expected cost required to locate an optimal solution to a new problem instance. However, because the most accurate cost models to date (as discussed below) are functions of the set of *all* optimal solutions to a problem instance, the effort required to generate the prediction actually exceeds that of simply locating an optimal solution. Given an accurate cost model, the problem of runtime prediction is essentially equivalent to the problem of estimating the value of the model parameters. The nature of the cost-accuracy trade-off in model parameter estimation is currently an open research question.

This does *not* imply that cost models are a scientific curiosity, useless in practice. Cost models have been used to make specific predictions regarding the behavior of local search metaheuristics (e.g., see [42]). Further, and perhaps most importantly, cost models can explicitly identify those features of the fitness landscape that are overwhelmingly responsible for problem difficulty in local search. By identifying such features, we are enabling algorithm designers to focus on the areas most likely to yield performance improvements and to move beyond the ad hoc, benchmark-driven design methodology that is current employed [10].

## **20.4 Fitness Landscape Features and Static Cost Models**

The performance of any local search metaheuristic is dictated by the interaction of the metaheuristic with the underlying fitness landscape. Toward understanding this interaction, researchers have initiated numerous investigations of the structural characteristics of the fitness landscapes of various combinatorial optimization problems. As a result, several fitness landscape features have been identified that have been shown, via abstract argument or concrete inference, to influence problem difficulty for local search. Examples of such features include<sup>1</sup> the following:

- The number and/or distribution of local optima
- The strength and size of local optima attractor basins
- The size and extension of the search space

Although the importance of these features is widely acknowledged, little or no empirical evidence exists to substantiate the *extent* to which any of these features, or combination thereof, is actually correlated with local search cost. Because the strength of the relationships has not, in general, been quantified, it is possible or

---

<sup>1</sup> Kauffman (p. 44, [16]) provides a more comprehensive list developed for adaptive local search algorithms.

even likely that the prime factor(s) dictating problem difficulty for local search have either yet to be identified or remain largely unexplored.

Structural features of the fitness landscape also have, or at least *should* have, a major influence on the design of metaheuristics. Local search metaheuristics differ largely in their approach to escaping the attractor basins of local optima, and the complexity of the proposed escape mechanisms—in terms of algorithmic details—is highly variable. Ideally, designers tailor a metaheuristic to the class of fitness landscapes that the algorithm is likely to encounter. Yet, very few concrete details are known about attractor basin strength, i.e., the expected computational effort required to escape local optima. This is true for nearly all combinatorial optimization problems. Consequently, it is unclear whether further attention on novel escape mechanisms is warranted, or if researchers should shift their focus to designing more effective high-level search strategies, such as those associated with advanced implementations of tabu search.

While important, the study of factors such as local optima attractor basin strength is not necessarily a driver in overall problem difficulty. In particular, we observe that once the problem of escaping local optima is solved, the broader issue of how to perform effective global search remains open. We now survey those global structural features of the fitness landscape that have been proposed to account for the variability in problem difficulty for local search. We present the motivation behind each feature, summarize prior research, and identify limitations. It should be noted that not all of these features have been investigated *explicitly* in the context of a cost model of local search. However, the objective of correlating the presence of particular features with search difficulty is a common, necessary theme. For illustrative purposes, we additionally provide in some cases graphics illustrating cost model accuracy drawn from the author’s own research on job-shop scheduling (JSP). Finally, we note that an alternative, complementary perspective on fitness landscape analysis is provided in [12].

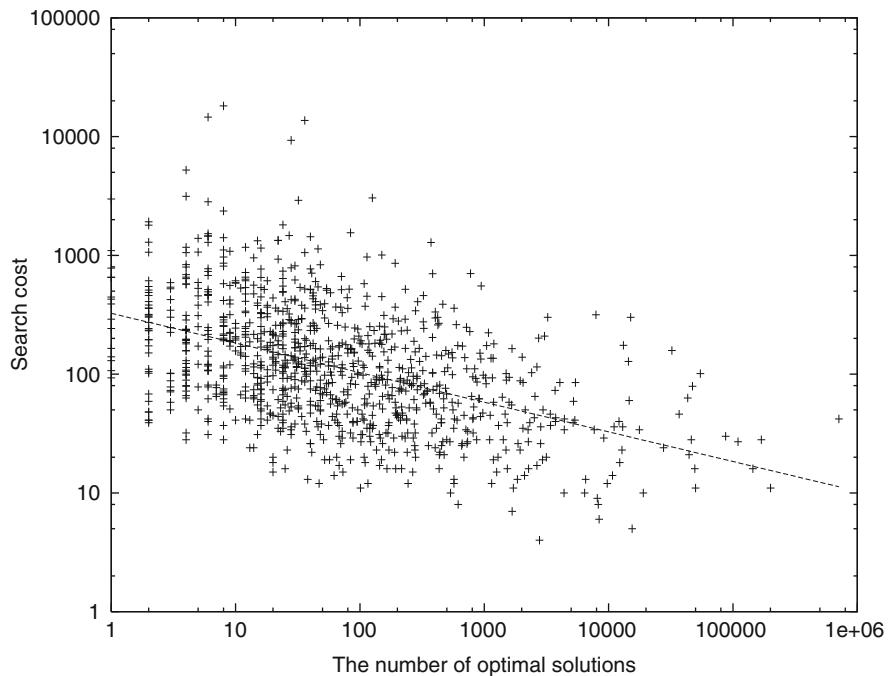
### 20.4.1 The Number of Optimal Solutions

One of the most intuitive measures of problem difficulty is the number of globally optimal solutions in a fitness landscape. It should be difficult to locate a global optimum if they are relatively rare. Conversely, if global optima are numerous, then it should be relatively easy for local search to find one.

The relationship between the number of globally optimal solutions and problem difficulty for local search was originally analyzed in the context of MAX-SAT and the more general MAX-CSP [4]. The motivation behind this research was to develop an explanation for the easy-hard-easy pattern in problem difficulty observed in the phase transition regions of these problems [17, 28]. It was initially conjectured that the peak in search cost was due to changes in the number of optimal solutions. Yokoo [45] proved that this was not the case, by showing that the mean number of optimal solutions varies in no special way near the phase transition region. In a more refined analysis, Clark et al. demonstrated a relatively strong

negative  $\log_{10}$ - $\log_{10}$  correlation between the number of globally optimal solutions and search cost for three MAX-SAT metaheuristics, with  $r$ -values ranging anywhere from  $-0.77$  to  $-0.91$ . However, the cost model failed to account for the large cost variance observed for problems with small numbers of optimal solutions, where model residuals varied over three or more orders of magnitude. A similar situation is exhibited in the JSP for tabu search.

In Figure 20.2, we show a scatter plot of the mean search cost required by tabu search (see [41] for details) to locate an optimal solution to 6 job, 6 machine random JSPs, as a function of the number of optimal solutions to a problem instance. The model is not overly accurate, with  $r^2 = 0.2223$ , and the graphic clearly demonstrates the very large residuals common to cost models based on the number of optimal solutions.



**Fig. 20.2** Scatter plot of the number of globally optimal solutions versus search cost for 6 job, 6 machine random job-shop problems; the least-squares fit line is super-imposed.

The distribution of the number of optimal solutions depends in large part on the nature of the objective function, specifically whether all or a fraction of solution attributes dictate solution fitness. For example, the number of optimal solutions to instances of the 2D integer Euclidean Traveling Salesman Problem is generally very small and is frequently equal to 1 [36]. The reason is straightforward: tour length is a function of *all* the cities in the instance, and the likelihood of two tours having identical lengths is relatively small given randomly sampled inter-city distances. The

likelihood of a single optimal solution is even higher if real-valued city coordinates are allowed. A similar situation is observed in the Permutation Flow-Shop Problem [39]. In contrast, the fitness of solutions in the JSP is dictated by a subset of job orderings, i.e., those on the critical path. Consequently, large plateaus of solutions are common in the JSP [40].

While intuitive, the accuracy of static cost models based on the number of optimal solutions is clearly limited. Further, there is anecdotal evidence that accuracy decreases as larger problem instances are considered.

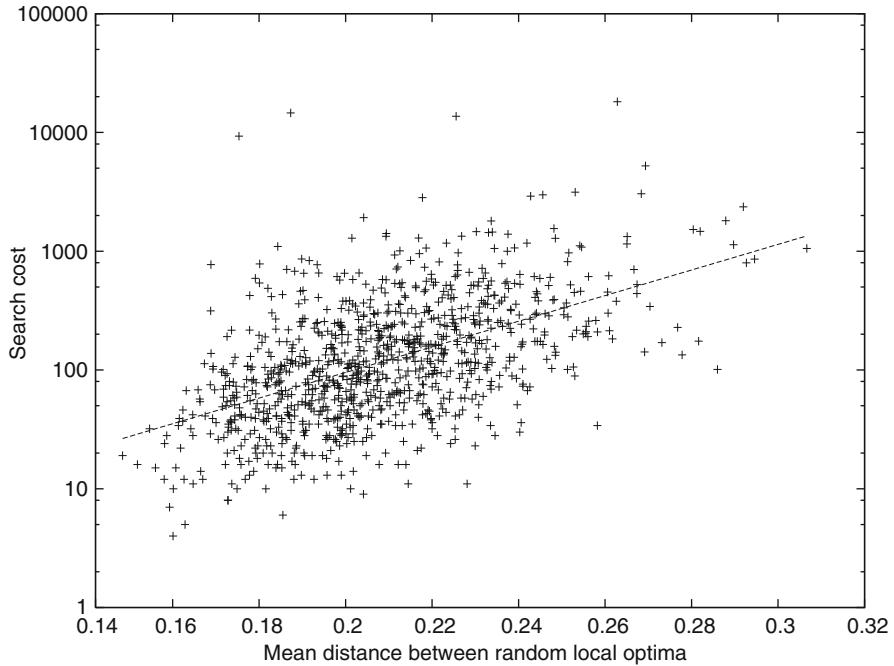
### **20.4.2 The Distance Between Local Optima**

The cost of local search is also influenced by the size of the search space. Search in most metaheuristics is heavily biased toward local optima, suggesting that the size of the sub-space of local optima may be strongly correlated with problem difficulty. A straightforward approach to quantifying the size of the local optima sub-space is to simply measure the mean distance between a sample of random local optima; large distances should be indicative of large sub-spaces. This notion of quantifying search space size was first introduced by Mattfeld et al. [21] in the context of the JSP. However, Mattfeld et al. did not investigate the ability of the metric to account for the variability in problem difficulty across a fixed set of instances; rather, they used the metric to account for differences between distinct types of JSP instances.

In Figure 20.3, we show a scatter plot of the mean search cost required by tabu search (again see [41] for details) to locate an optimal solution to 6 job, 6 machine random JSPs, as a function of the mean distance between random local optima. Accuracy is similar to the static cost model based on the number of optimal solutions, with  $r^2 = 0.2744$ . Similarly, the accuracy of such models tends to decrease with increases in problem size and the graphic exhibits very large residuals, varying over several orders of magnitude.

### **20.4.3 The Distance Between Local and Global Optima**

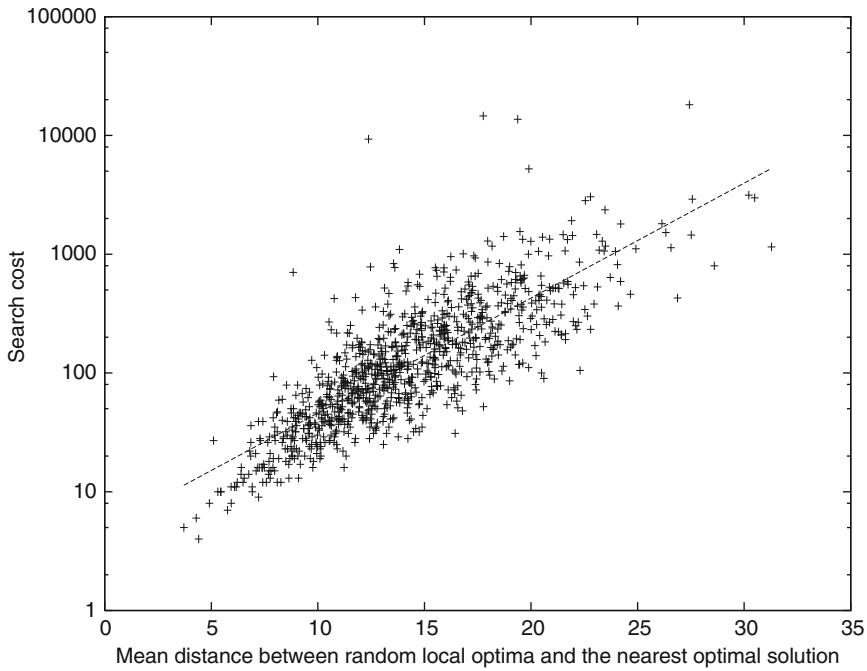
The number of globally optimal solutions and the size of the search space  $S$  are conceptually independent; it is possible to embed as many as  $|S|$  optimal solutions within a search space  $S$ . Undoubtedly, both factors influence problem difficulty for local search. If we fix  $|S|$  and assume that attractor basin strength and size remain relatively constant, we expect problems to become easier as the number of optimal solutions grows. Analogously, it should be more difficult to locate an optimal solution as the number of optimal solutions shrinks. It follows that both the number of optimal solutions and the distance between local optima are, in isolation, unlikely to account for a significant proportion of the variability in problem difficulty for local search.



**Fig. 20.3** Scatter plot of the mean distance between random local optima versus search cost for 6 job, 6 machine random job-shop problems; the least-squares fit line is super-imposed.

To correct for these flaws, we now discuss a measure that simultaneously accounts for the impact of both features on problem difficulty: the mean distance between random local optima and the *nearest* optimal solution. The intuition is that problem difficulty for local search is proportional to the total distance that must be traversed between an initial solution (e.g., a random local optima) and a target solution (e.g., an optimal solution). This measure was first introduced in the context of MAX-SAT by Singer et al. [35]. Well-known local search algorithms for MAX-SAT rapidly descend from poor-quality initial solutions to near-optimal “quasi-solutions,” and subsequent search is restricted to the space of such quasi-solutions. Singer et al. hypothesized that the search cost was proportional to the size of the quasi-solution sub-space, which in turn could be estimated by the mean distance between the first quasi-solution encountered and the nearest optimal solution. Their experimental results demonstrated a very strong ( $r \approx 0.95$ ) correlation between this metric and search cost for easy MAX-SAT instances; for more difficult instances, the accuracy degraded only slightly to  $r \approx 0.75$ .

In Figure 20.4, we show a scatter plot of the mean search cost required by tabu search (again see [41] for details) to locate an optimal solution to 6 job, 6 machine random JSPs, as a function of the mean distance between random local optima and the nearest optimal solution. The  $r^2$  value for the corresponding static cost model is equal to 0.6424, similar to the accuracy obtained by Singer et al. With few excep-



**Fig. 20.4** Scatter plot of the mean distance between random local optima and the nearest optimal solution versus search cost for 6 job, 6 machine random job-shop problems; the least-squares fit line is super-imposed.

tions, residuals vary over roughly 1 to 1.5 orders of magnitude; the improvement is significant relative to static cost models based strictly on the number of optimal solutions or the mean distance between local optima. Clearly, the static cost model based on the measure proposed by Singer et al. is a landmark achievement, representing the first reasonably accurate cost model of any local search metaheuristic, for any combinatorial optimization problem. Previously proposed models achieved accuracy of at most  $r^2 \approx 0.3$  in the worst case, in contrast to the  $r^2 \approx 0.6$ , achieved by Singer et al.

#### 20.4.4 Fitness-Distance Correlation

Another factor hypothesized to influence problem difficulty for *adaptive* local search algorithms is the correlation between solution fitness and the distance to an optimal solution, often simply denoted as FDC (fitness-distance correlation) [18, 22, 24, 37]. In a problem instance with high FDC, good solutions tend to be tightly clustered or, equivalently, share many solution attributes in common. Consequently, an adaptive search algorithm should be able to exploit these similarities during search. For example, Schneider et al. [33] introduce an adaptive local search algorithm for the Traveling Salesman Problem that, after identifying the edges

common to a set of high-quality local optima, restricts subsequent search to the sub-space of solutions with *only* those edges. Similarly, Sourlas [37] introduced an adaptive simulated annealing algorithm for the TSP that determines those edges appearing infrequently in high-quality solutions and prevents subsequent search from generating tours containing those edges. FDC has also been used to account for differences in the relative difficulty of problem instances, e.g., see [15]. As with correlation length, we do not further consider FDC in the context of cost models for local search due to most (basic forms of) local search metaheuristics being non-adaptive in the evolutionary algorithm sense. Further, there is little evidence that FDC can account for a significant proportion of variability in search cost observed for a set of fixed-sized problem instances.

#### **20.4.5 Solution Backbones**

Recently, a number of researchers (e.g., [1] and [36]) have hypothesized that the *backbone* of a problem instance may be correlated with problem difficulty. Informally, the backbone of an instance is the set of solution attributes or variables that possess identical values in *all* optimal solutions; as a consequence, the definition of a backbone depends on the representation scheme used to encode solutions. The intuition behind the backbone measure is that the majority of effort in local search may be spent assigning correct values to backbone variables. Non-backbone variables appear to be significantly less constrained, enabling search to quickly locate an optimal solution once the backbone is located.

The recent interest in backbones is due in large part to the observation that large-backboned problem instances begin to appear in large quantities near the critical region of the Random 3-SAT phase transition [34] [27] [23] [35]; the coincidence of the two observations immediately leads to the hypothesis that backbone size is correlated with problem difficulty. More recently, Achlioptas et al. [1] argue that the shift from small-to large-backboned instances in the phase transition region suggests that the most difficult instances may in fact have a backbone size of 0.5, although this hypothesis has not been verified. Slaney and Walsh [36] analyze the correlation between problem difficulty and backbone size for constructive search algorithms for a number of *NP*-hard optimization problems. For the Traveling Salesman and Number Partitioning Problems, they report a weak-to-moderate correlation (e.g.,  $r$  between 0.138 and  $r = 0.388$ ) between backbone size and the cost of locating an optimal solution.

#### **20.4.6 Landscape Correlation Length**

A number of researchers have hypothesized that the “ruggedness” of a fitness landscape is likely to be highly correlated with problem difficulty for adaptive search algorithms such as genetic and other evolutionary algorithms [16, 38, 43]. A fitness landscape is said to be rugged if there is a rapid change in the fitness between

nearby solutions in the landscape. If the fitness of nearby solutions is uncorrelated, we cannot expect adaptive search to outperform a random walk, i.e., there is no structure to exploit. Ruggedness is frequently quantified as the landscape correlation length, which captures the maximal distance between two arbitrary solutions for which there still exists significant correlation between their fitness values [43]. We do not consider correlation length in the context of static cost models of local search for two reasons. First, most local search metaheuristics are not adaptive, such that correlation length is unlikely to have a major impact on problem difficulty. Second, and more importantly, the extensive research on landscape correlation lengths indicates that for a wide range of well-known *NP*-hard optimization problems, the correlation length is *strictly* a function of problem size [32]. For example, the correlation length in an  $n$ -city TSP is given by  $n/2$ , while in an  $n$ -vertex Graph Bi-Partitioning Problem, it is given by  $(n - 3)/8$  [38]. Perhaps most dramatically, Rana [29] showed that the landscape correlation length is effectively constant over the easy-hard-easy pattern in problem difficulty observed for Random 3-SAT. Consequently, correlation length fails to account for *any* of the variability in problem difficulty observed in sets of fixed-sized problem instances.

#### 20.4.7 Phase Transitions

Much of research on problem difficulty within the artificial intelligence and computer science communities has focused on the identification of so-called *phase transitions* in problem difficulty [9]. A phase transition in a combinatorial optimization problem identifies an order parameter that partitions the universe of problem instances into subsets with differing degrees of expected difficulty. For example, the clause-to-variable ratio  $m/n$  in Random 3-SAT induces a clear pattern: as  $m/n$  ranges from 0 to  $\infty$ , the degree of problem difficulty exhibits a well-known easy-hard-easy pattern [3]. While successful in identifying inter-partition differences in problem difficulty, phase transitions fail to account for the often considerable variability *within* a partition; the latter can vary over many (e.g., 6 or more) orders of magnitude, even for small problem instances. The failure to explain intra-partition variance in problem difficulty should not, however, be viewed as a deficiency of phase transition models; phase transition research was initially motivated by the desire to generate difficult test problems, and this goal has been achieved.

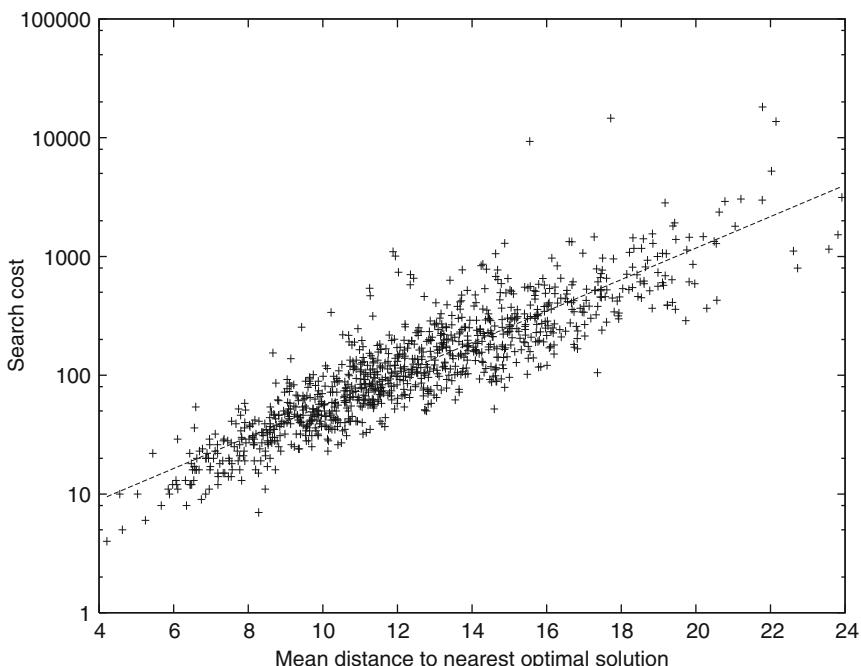
### 20.5 Fitness Landscapes and Run-Time Dynamics

Fitness landscape analysis and the associated static cost models are only a first step toward a more general theory of local search metaheuristics. Rather, the ultimate objective is to develop models linking the fitness landscape structure with models of metaheuristic search dynamics. In contrast to work on fitness landscape analysis, research addressing metaheuristic dynamics is very limited. This is due in part to the

difficulty of the modeling and the relatively recent emphasis on dynamics modeling. In this section, we survey this research, highlighting the accuracy of the models, and the insights they facilitate.

As discussed previously, static cost models only correlate fitness landscape structures with search cost; metaheuristic dynamics are completely ignored. An intermediate between static and dynamic cost models are quasi-dynamic cost models, which are based on summary information concerning metaheuristic dynamics. One example is introduced by Watson et al. [42] in the context of tabu search and the JSP. In Section 20.4.3, we introduced a related static cost model based on the mean distance between random local optima and the nearest optimal solution. However, Watson et al. observe that random local optima are *not* representative of the solutions visited by tabu search during execution. Instead, they proposed a quasi-dynamic cost model based on the mean distance between solutions actually visited by tabu search and the nearest optimal solution.

In Figure 20.5, we show a scatter plot of the mean cost required by tabu search to locate an optimal solution to 6 job, 6 machine random JSPs, as function of the mean distance between solutions visited by tabu search and the nearest optimal solution. The  $r^2$  value for the corresponding quasi-dynamic cost model is 0.7808, representing a 21% increase over the corresponding static cost model; greater improvements



**Fig. 20.5** Scatter plot of the mean distance between solutions visited by tabu search and the nearest optimal solution versus search cost for 6 job, 6 machine random job-shop problems; the least-squares fit line is super-imposed.

in accuracy were obtained for larger problem instances. With few exceptions, the predicted costs are within a factor of five of the observed costs, representing a marked improvement in accuracy relative to the best available static cost models presented previously.

While quasi-dynamic cost models clearly illustrate the improved accuracy that is facilitated by linking fitness landscape structure and metaheuristics dynamics, they provide only indirect insight into metaheuristic dynamics, which is—we argue—the primary objective in developing any theory of local search metaheuristics. Most metaheuristics are randomized, if implicitly (e.g., through specification of an initial starting solution), such that Markov chains can be used to explicitly model search dynamics. There are two primary challenges in developing such Markov models: (1) aggregating elements of the search space, in order to avoid exponential numbers of states and (2) estimating the state transition probabilities.

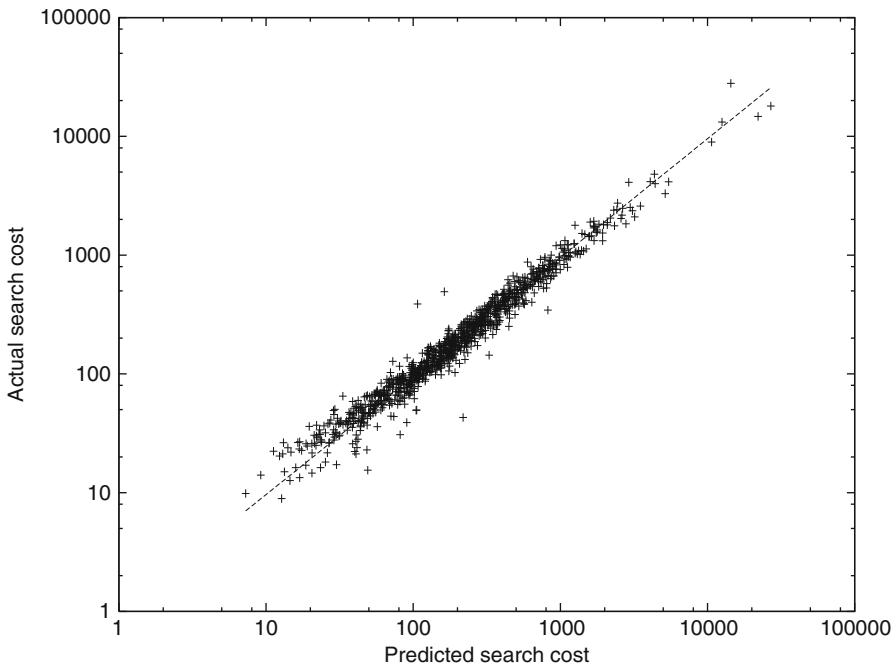
To date, the majority of dynamic models of metaheuristics (which, as discussed below, are very limited) follow Pappadimitriou [26] in aggregating states based on their distance to the nearest optimal solution. Depending on the metaheuristic, it may further be necessary to replicate states in order to account for memory mechanisms, e.g., of the form found employed in tabu search [42].

Hoos [11] proposed a Markov model based on distance aggregation in the context of MAX-SAT. The objective of Hoos' analysis was to provide an explanation for specific stagnation behavior in (at the time) state-of-the-art local search metaheuristics for MAX-SAT, in which the run-time cost required to locate optimal solutions for certain instances could not be explained by existing models. Hoos hypothesized that the observed stagnation behavior was caused by the presence of sub-optimal “traps,” which caused search to be drawn away from regions of the state space containing optimal solutions and introduced a branched Markov model to represent the search dynamic. Using posited transition probabilities, Hoos demonstrated that the model accurately captured the search dynamics observed by MAX-SAT local search metaheuristics on this class of problem instance.

Watson et al. [42] introduced Markov models for a tabu search algorithm for the JSP. The contents of short-term memory were abstractly represented as the current “gradient” or change in the distance to the nearest optimal solution observed between the current and previous iteration of the tabu search metaheuristic and embedded in the Markov state capturing distance to the nearest optimal solution. Transition probabilities were estimated by periodically observing the tabu search algorithm, computing the current search gradient and the distance to the nearest optimal solution; transition probabilities were then estimated using the aggregate sample, for each problem instance.

The resulting Markov models were then simulated to compute the distribution of the number of tabu search iterations required to locate an optimal solution. Comparison of the simulated and empirical results indicated that the proposed Markov model accurately predicts the observed search costs; predicted mean search cost was within a factor of five of the observed value, and the full search cost distribution was reasonably approximated by the Markov model. Although beyond the present scope, Watson et al. also discuss novel observations regarding the linkages between

static, quasi-dynamic, and dynamic cost models in the context of tabu search. Further, the model allowed the authors to propose and test certain hypotheses regarding metaheuristic behavior for the JSP, including the lack of benefit due to alternate initialization strategies. For completeness and contrast with corresponding results for static and quasi-dynamic cost models, we show in Figure 20.6 the predicted versus actual search costs for 6 job, 6 machine random JSPs. The  $r^2$  for the model is a remarkable 0.96, with predictions in nearly all cases within a factor of two of the observed values.



**Fig. 20.6** Scatter plot of the predicted versus actual search cost for 6 job, 6 machine random job-shop problems, using a Markov model; the least-squares fit line is super-imposed.

Most recently, Fournier [6] introduced a Markov model to account for a simple stochastic local search metaheuristic for MAX-SAT. In contrast to Hoos and Watson et al. the search space is aggregated in terms of solution quality. Each discrete value of solution quality (the number of unsatisfied clauses) is represented by a state in the Markov chain. The state transition matrix then specifies the probability of transitioning from a state with quality  $q$  to a neighboring state with solution quality  $q'$ . The resulting Markov chain is then simulated and compared against experimental results to assess model quality, which in turn provides information concerning the accuracy of the proposed metaheuristic dynamics.

Fournier's analysis focuses on a simple metaheuristic for MAX-SAT, called RSAT, which simply selects a neighbor at each iteration with a probability in proportion to the neighbor's quality. This is in contrast to Hoos and Watson et al. who analyze metaheuristics closely related to the state of the art for their respective problems. Further, Fournier's analysis is primarily concerned with average behavior over an ensemble of instances. In particular, the transition matrix is estimated from a large sample of instances, aggregated into a single ensemble estimate. Not unexpectedly, the accuracy of the model on a per-instance basis is limited, although the model (with some minor, noted exceptions) accurately captures metaheuristic dynamics at the ensemble level.

While limited, the research into dynamic models of metaheuristic behavior has lead to impressive advances—relative to simple models based on fitness landscape features—in cost model accuracy. This progression in accuracy is graphically illustrated in Figures 20.4 through 20.6. Although far from representing a general theory of local search metaheuristics, such dynamic models do provide the first steps in that general, key direction.

## 20.6 Conclusions

Despite the high level of research activity in local search metaheuristics over the last two decades, comparatively little progress has been made in the theoretical foundations of the field. Most research focuses either on the application of existing metaheuristics to new problems or the development of new metaheuristics. Ideas and techniques are routinely re-introduced and re-invented, and it is often difficult to assess the novelty and/or contribution of new research. Recently, the roots of a theory of local search have begun to emerge. The type of model discussed in this chapter, we believe, provides a basis for a more general theory of local search. Specifically, we have seen examples of how researchers have used fitness landscape analysis to better understand the mechanisms underlying metaheuristic search and how these mechanisms give rise to various observed behaviors. Model generalization to both other problems and a wider range of metaheuristics is a significant outstanding challenge. Similarly, the implications of these models for metaheuristic design are largely unknown and unexplored. Even with inefficient and ad hoc development methodologies, researchers have continued to make significant advances in the effectiveness of local search metaheuristics. By developing a generalized theory of local search, it should be possible to more precisely focus future research and, as a consequence, significantly accelerate the rate of advances in the field.

**Acknowledgments** Sandia is a multipurpose laboratory operated by Sandia Corporation, a Lockheed-Martin Company, for the United States Department of Energy under contract DE-AC04-94AL85000.

## References

1. Achlioptas, D., Gomes, C., Kautz, H., Selman, B.: Generating satisfiable problem instances. In: Ford, K. (ed.) Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-00), pp. 256–261. AAAI/MIT Press (2000)
2. Beveridge, J., Graves, C., Steinborn, J.: Comparing random starts local search with key feature matching. In: Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97) (1997)
3. Cheeseman, P., Kanefsky, B., Taylor, W.: Where the *Really* hard problems are. In: Proceedings of the 12th International Joint Conference on Artificial Intelligence (IJCAI-91), pp. 331–337 (1991)
4. Clark, D., Frank, J., Gent, I., MacIntyre, E., Tomov, N., Walsh, T.: Local search and the number of solutions. In: Freuder, E.C. (ed.) Proceedings of the 2nd International Conference on Principles and Practices of Constraint Programming (CP-96), pp. 119–133. Springer (1996)
5. Cohen, P.: Empirical Methods for Artificial Intelligence. MIT Press (1995)
6. Fournier, N.G.: Modelling the dynamics of stochastic local search on k-SAT. *J. Heuristics* **13**, 587–639 (2007)
7. Frank, J., Cheeseman, P., Stutz, J.: When gravity fails: local search topology. *J. Artif. Intell. Res.* **7**, 249–281 (1997)
8. Glover, F., Laguna, M.: Tabu Search. Kluwer, Boston, MA (1997)
9. Hogg, T., Huberman, B., Williams, C.: Special issue on frontiers in problem solving: phase transitions and complexity. *Artif. Intell.* **81**(1–2) (1996)
10. Hooker, J.: Testing heuristics: we have it all wrong. *J. Heuristics* **1**, 33–42 (1995)
11. Hoos, H.: A mixture-model for the behaviour of SLS algorithms for SAT. In: Proceedings of the 18th National Conference on Artificial Intelligence (AAAI-02), pp. 661–667. AAAI Press/MIT Press (2002)
12. Hoos, H., Stüzle, T.: Stochastic Local Search: Foundations and Applications. Morgan Kaufmann (2005)
13. Johnson, D., McGeoch, L.A.: The traveling salesman problem: a case study in local optimization. In: Local Search in Optimization, pp. 215–310. John Wiley (1997)
14. Jones, T.: Evolutionary algorithms, fitness landscapes, and search. Ph.D. thesis, Department of Computer Science, University of New Mexico (1995)
15. Jones, T., Forrest, S.: Fitness distance correlation as a measure of problem difficulty. In: Eschelman L. (ed.) Proceedings of the 6th International Conference on Genetic Algorithms, pp. 184–192. Morgan Kaufmann (1995)
16. Kauffman, S.: The Origins of Order. Oxford University Press (1993)
17. Kirkpatrick, S., Selman, B.: Critical behavior in the satisfiability of random boolean expressions. *Science* **264**, 1297–1301 (1994)
18. Kirkpatrick, S., Toulouse, G.: Configuration space analysis of traveling salesman problems. *J. Phys.* **46**, 1277–1292 (1985)
19. Lin, S., Kernighan, B.: An effective heuristic algorithm for the traveling salesman problem. *Oper. Res.* **21**, 498–516 (1973)
20. Lourenço, H., Martin, O., Stützle, T.: Iterated local search. In: Glover, F., Kochenberger, G. (eds.) Handbook of Metaheuristics. Kluwer (2003)
21. Mattfeld, D., Bierwirth, C., Kopfer, H.: A search space analysis of the job shop scheduling problem. *Ann. Oper. Res.* **86**, 441–453 (1999)
22. Mezard, M., Parisi, G.: A replica analysis of the traveling salesman problem. *J. Phys.* **47**, 1285–1296 (1986)
23. Monasson, R., Zecchina, R., Kirkpatrick, S., Selman, B., Troyansky, L.: Determining computational complexity for characteristic ‘phase transitions’. *Nature* **400**, 133–137 (1998)
24. Mühlenthaler, H., Georges-Schleuter, M., Krämer, O.: Evolution algorithms in combinatorial optimization. *Parallel Comput.* **7**, 65–85 (1988)
25. Nowicki, E., Smutnicki, C.: A fast taboo search algorithm for the job shop problem. *Manage. Sci.* **42**(6), 797–813 (1996)

26. Papadimitriou, C.: On selecting a satisfying truth assignment. In: Proceedings of the 33rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 91) (1991)
27. Parkes, A.: Clustering at the phase transition. In: Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97), pp. 340–345. AAAI/MIT Press (1997)
28. Prosser, P.: Binary constraint satisfaction problems: Some are harder than others. In: Proceedings of the 11th European Conference on Artificial Intelligence (ECAI-94), pp. 95–99 (1994)
29. Rana, S.: Local optima and genetic algorithms. Ph.D. thesis, Department of Computer Science, Colorado State University (1999)
30. Rana, S., Whitley, L.: Representation, search, and genetic algorithms. In: Proceedings of the 14th National Conference on Artificial Intelligence (AAAI-97). AAAI Press/MIT Press (1997)
31. Reeves, C.: Landscapes, operators and heuristic search. *Ann. Oper. Res.* **86**, 473–490 (1998)
32. Riedys, C., Stadler, P.: Combinatorial landscapes. Technical Report 01-03-014, Santa Fe Institute (2001)
33. Schneider, J., Froschhammer, C., Morgernstern, I., Husslein, T., Singer, J.: Searching for backbones—an efficient parallel algorithms for the traveling salesman problem. *Comput. Phys. Comm.* **96**, 173–188 (1996)
34. Schrag, R., Crawford, J.M.: Implicates and prime implications in random 3SAT. *Artif. Intell.* **88**, 199–222 (1996)
35. Singer, J., Gent, I., Smaill, A.: Backbone fragility and the local search cost peak. *J. Artif. Intell. Res.* **12**, 235–270 (2000)
36. Slaney, J., Walsh, T.: Backbones in optimization and approximation. In: Nebel, B. (ed.) Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI-01), pp. 254–259. Morgan Kaufmann (2001)
37. Sourlas, N.: Statistical mechanics and the traveling salesman problem. *Europhys. Lett.* **2**, 919–923 (1986)
38. Stadler, P.: Landscapes and their correlation functions. *J. Math. Chem.* **20**, 1–45 (1996)
39. Stützle, T.: Personal communication (2001)
40. Watson, J.P.: Problem difficulty for local search in job-shop scheduling. Ph.D. thesis, Department of Computer Science, Colorado State University (2003)
41. Watson, J.P., Beck, J.C., Howe, A., Whitley, L.: Problem difficulty for tabu search in job-shop scheduling. *Artif. Intell.* **143**(2), 189–217 (2003)
42. Watson, J.P., Whitley, L.D., Howe, A.E.: Linking search space structure, run-time dynamics, and problem difficulty: a step toward demystifying tabu search. *J. Artif. Intell. Res.* **24**, 221–261 (2005)
43. Weinberger, E.D.: Correlated and uncorrelated fitness landscapes and how to tell the differences. *Biol. Cybern.* **63**, 325–336 (1989)
44. Wright, S.: The roles of mutation, inbreeding, crossbreeding and selection in evolution. In: Jones, D. (ed.) International Proceedings of the 6th International Congress on Genetics, vol. 1, pp. 356–366 (1932)
45. Yokoo, M.: Why adding more constraints makes a problem easier for hill-climbing algorithms: Analyzing landscapes of CSPs. In: Proceedings of the 3rd International Conference on the Principles and Practice of Constraint Programming (CP-97), pp. 356–370. Springer (1997)



# Chapter 21

## Comparison of Metaheuristics

John Silberholz and Bruce Golden

**Abstract** Metaheuristics are truly diverse in nature—under the overarching theme of performing operations to escape local optima, algorithms as different as ant colony optimization, tabu search, harmony search, and genetic algorithms have emerged. Due to the unique functionality of each type of metaheuristic, comparison of metaheuristics is in many ways more difficult than other algorithmic comparisons. In this chapter, we discuss techniques for meaningful comparison of metaheuristics. We discuss how to create and classify instances in a new testbed and how to make sure other researchers have access to the problems for future metaheuristic comparisons. Further, we discuss the disadvantages of large parameter sets and how to measure complicating parameter interactions in a metaheuristic’s parameter space. Last, we discuss how to compare metaheuristics in terms of both solution quality and runtime.

### 21.1 Introduction

Metaheuristics are truly diverse in nature—under the overarching theme of performing operations to escape local optima (we assume minima in this chapter without loss of generality), algorithms as different as ant colony optimization [12], tabu search [16], and genetic algorithms [23] have emerged. Due to the unique functionality of each type of metaheuristic, comparison of metaheuristics is in many ways more difficult than other algorithmic comparisons.

---

John Silberholz

Center for Scientific Computing and Mathematical Modeling, University of Maryland, College Park, MD 20740, USA  
e-mail: [josilber@umd.edu](mailto:josilber@umd.edu)

Bruce Golden

R.H. Smith School of Business, University of Maryland, College Park, MD 20740, USA  
e-mail: [bgolden@rhsmith.umd.edu](mailto:bgolden@rhsmith.umd.edu)

In this chapter, we discuss techniques for meaningful comparison of metaheuristics. In Section 21.2, we discuss how to create and classify instances in a new testbed and how to make sure other researchers have access to the problems for future metaheuristic comparisons. In Section 21.3, we discuss the disadvantages of large parameter sets and how to measure complicating parameter interactions in a metaheuristic’s parameter space. Last, in Sections 21.4 and 21.5, we discuss how to compare metaheuristics in terms of both solution quality and runtime.

## 21.2 The Testbed

It seems natural that one of the most important parts of a comparison among heuristics is the testbed on which the heuristics are tested. As a result, the testbed should be the first consideration when comparing two metaheuristics.

### 21.2.1 Using Existing Testbeds

When comparing a new metaheuristic to existing ones, it is advantageous to test on the problem instances already tested by previous papers. Then, results will be comparable on a by-instance basis, allowing relative gap calculations between the two heuristics. Additionally, trends with regard to specific types of problem instances in the testbed can be made, making analysis of the new metaheuristic simpler.

### 21.2.2 Developing New Testbeds

While ideally testing on an existing testbed would be sufficient, there are many cases when this is either insufficient or not possible. For instance, when writing a metaheuristic for a new problem, there will be no testbed for that problem, so a new one will need to be developed. In addition, even on existing problems where heuristic solutions were tested on non-published, often randomly generated problem instances, such as those presented in [15, 25], a different testbed will need to be used. Last, if the existing testbed is insufficient (often due to being too small to effectively test a heuristic), a new one will need to be developed.

#### 21.2.2.1 Goals in Creating the Testbed

The goals of a problem suite include mimicking real-world problem instances while providing test cases that are of various types and difficulty levels.

One key requirement of the testbed that is especially important in the testing of metaheuristics is that large problem instances must be tested. For small instances, optimal solution techniques often run in reasonable runtimes while giving the advantage of a guaranteed optimal solution. It is, therefore, critical that metaheuristic testing occurs on the large problems for which optimal solutions could not be calculated in reasonable runtimes. As discussed in [19], it is not enough to test on small problem instances and extrapolate the results for larger instances; algorithms can perform differently in both runtime and solution quality on large problem instances.

While it is desirable that the new testbed be based on problem instances found in industrial applications of the problem being tested (like the TSPLIB, [28]), it is typically time intensive to do this sort of data collection. Often real-world data is proprietary and, therefore, difficult to obtain. Furthermore, the problem instances generated will typically be small in number and size. For instance, real-world problem instances used for testing on the generalized traveling salesman problem proposed in [13] all had fewer than 50 nodes. In addition, there were only two real-world problem instances proposed; nearly all of the problem instances used in that paper did not come from real-world data, and all of the larger problem instances were artificial.

As a result, it is generally more reasonable to create a testbed based on existing well-known problem instances than it is to create one from scratch. For instance, many testbeds have been successfully made based on the TSPLIB. In the case of the generalized traveling salesman problem, [13] established a well-used testbed based on a simple extension to TSPLIB problem instances. Another example of such an extension can be found in [2], in which the authors tested several different modified versions of 10 benchmark VRP problems and reported computational results on each variation.

### 21.2.2.2 Accessibility of New Test Instances

When creating a new testbed, the focus should be on providing others access to the problem instances. This will allow other researchers to more easily make comparisons, ensuring the problem instances are widely used. One way to ensure this would be to create a simple generating function for the problem instances. For instance, the clustering algorithm proposed in [13] that converted TSPLIB instances into clustered instances for the generalized traveling salesman problem was simple, making it easy for others to create identical problem instances. Additionally, publishing the problem instances tested is another effective way to make them accessible. This was an effective technique used, for instance, in [7, 31].

In developing a new testbed, capturing real aspects of a problem is important. For instance, in the problem instances found in [13], the clustering algorithm placed nodes in close proximity to each other in the same cluster, capturing real-life characteristics of this problem.

### 21.2.2.3 Geometrically Constructed Problem Instances

One problem in the analysis of metaheuristics, as discussed in more detail in Section 21.4 is finding errors for the algorithms. Even when using advanced techniques, it is typically difficult to determine optimal solutions for large problem instances. A way to minimize the difficulty in this step is to use geometrically constructed solutions for which optimal or near-optimal solutions are apparent. This removes the burden on the metaheuristics designer to also implement an exact approach, relaxation results, or a tight lower bound. Instead, the designer can use the specially designed problem instances and provide a good estimate of the error of each metaheuristic tested.

A number of papers in the literature have used this approach. For instance, in [6], problem instances for the split delivery vehicle routing problem were generated with customers in concentric circles around the depot, making estimation of optimal solutions possible visually. Other examples of this approach are found in [5, 20–22].

### 21.2.3 Problem Instance Classification

Regardless of whether an existing or new testbed is used, classifying the problem instances being tested is critical to the proper analysis of heuristics. Differentiating factors between problem instances should be noted prior to any experimentation, and heuristic performance on each type of problem instance should be discussed. A good example of such analysis is found in [17], an experimental evaluation of heuristics for the resource-constrained project scheduling problem. That paper split problem instances by three key problem instance parameters, the network complexity, resource factor, and resource strength, analyzing the effects of each on the performance of the heuristics. Especially in testbeds based on real-world data, this classification of problem instances and subsequent analysis could help algorithm writers in industry with a certain type of dataset to determine which method will work the best for them.

## 21.3 Parameters

Though deciding upon a quality testbed is critical when comparing solution qualities and runtimes, it is also important to compare the actual algorithms. This can be accomplished in part by considering the complexity of the algorithms; if two algorithms produce similar results but one is significantly simpler than the other, then the simpler of the two is a superior algorithm. Algorithms with a low degree of complexity have a number of advantages, including being simple to implement in an industrial setting, being simple to reimplement by researchers, and being simpler to explain and analyze.

A number of measures of simplicity exist. Reasonable metrics include the number of steps of pseudocode needed to describe the algorithm or the number of lines of code needed to implement the algorithm. However, these metrics are not particularly useful, as they vary based on programming language and style in the case of the lines of code metric and pseudocode level of detail in the case of the pseudocode length metric. A more meaningful metric for algorithmic complexity is the number of parameters used in the algorithm.

Parameters are the configurable components of an algorithm that can be changed to alter the performance of that algorithm. Parameters can either be set statically (for instance, creating a genetic algorithm with a population size of 50) or based on the problem instance (for instance, creating a genetic algorithm with a population size of  $5\sqrt{n}$ , where  $n$  is the number of nodes in the problem instance). In either of these cases, the constant value of the parameter or the function of problem instance attributes used to generate the parameter must be predetermined by the algorithm designer.

Each major type of metaheuristic has a number of parameters that must be set before algorithm execution. Consider Table 21.1, which lists basic parameters required for major types of metaheuristics. Though these are guidelines for the minimum number of parameters typical in different types of algorithms, in practice, most metaheuristics have more parameters. For instance, a basic tabu search procedure can have just one parameter, the tabu list length. However, some procedures have many more than that one parameter. The tabu search for the vehicle routing problem presented in [33] uses 32 parameters. Likewise, algorithms can have fewer than the “minimum” number of parameters by combining parameters with the same value. For instance, the genetic algorithm for the minimum label spanning tree problem in [32] uses just one parameter, which functions both to control the population size and to serve as a termination criterion.

### 21.3.1 Parameter Space Visualization and Tuning

Metaheuristics using many parameters are more complex than procedures with few parameters for a number of reasons. First, the effort needed to tune or understand these parameters is far greater as the number of parameters increases. A brute-force technique for parameter tuning involves testing  $m$  parameter values for each of the  $n$  parameters, a procedure that should test  $n^m$  configurations over a subset of the problem instances. Assuming we choose to test just three values for each parameter, we must test nine configurations for an algorithm with two parameters and 2,187 values for an algorithm with seven parameters. While this number of configurations is likely quite reasonable, the number needed for a 32-parameter algorithm, 1,853,020,188,851,841, is clearly not reasonable. The size of the parameter space for an algorithm with a large number of parameters expands in an exponential manner, making the search for a good set of parameters much more difficult as the number of parameters increases. While, of course, there are far better ways to search for good parameter combinations than brute-force search, the size of the search space

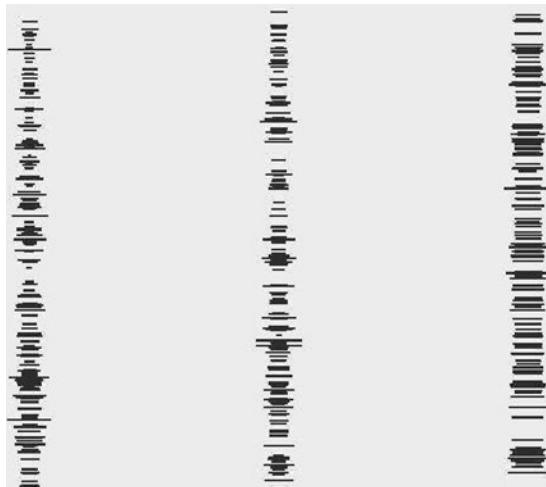
**Table 21.1** Popular metaheuristics and their standard parameters.

Name	Parameters
Ant colony optimization	Pheromone evaporation parameter Pheromone weighting parameter
Genetic algorithm	Crossover probability Mutation probability Population size
Harmony search	Distance bandwidth Memory size Pitch adjustment rate Rate of choosing from memory
Simulated annealing	Annealing rate Initial temperature
Tabu search	Tabu list length
Variable neighborhood search	None

still increases exponentially with the number of parameters, meaning a large number of parameters makes this search much more difficult.

Larger numbers of parameters also make the parameter space much harder to visualize or understand. As a motivating example, consider the relative ease with which the parameter space of an algorithm with two parameters can be analyzed. We analyzed the two-parameter metaheuristic due to [30] for solving the generalized orienteering problem on a few random problems from the TSPLIB-based large-scale orienteering problem dataset considered in that paper. To analyze this algorithm, we chose a number of parameter configurations in which each parameter value was close to the parameter values used in that paper. For each parameter set, the algorithm was run 20 times on each of five randomly selected problem instances from all the TSPLIB-based instances used. The optimal solutions are known for each of the five problem instances tested.

The resulting image, shown in Figure 21.1, is a testament to the simplicity of analysis of an algorithm with just two parameters. In this figure, different values of the parameter  $i$  are shown on the  $x$ -axis, while different values of the parameter  $t$  are shown on the  $y$ -axis. Parameter  $i$  is an integral parameter with small values, so results are plotted in three columns representing the three values tested for that parameter: three, four, and five. For each parameter set (a pair of  $i$  and  $t$ ), a horizontal line is plotted with width normalized by the average error of the algorithm over the 20 runs for each of the five problem instances tested. A narrow width corresponds to an average error near the best performance of the testing, which is 2.53%, while a wide width corresponds to an average error near the worst performance of the testing, which is 4.08%. In a dense parameter space, the same sort of visualization could be gleaned by coloring dots with colors relating to the error or by presenting a three-dimensional depiction, where the  $z$ -coordinate is the error.



**Fig. 21.1** Depiction of solution quality of a metaheuristic for the generalized orienteering problem over its two-dimensional parameter space. The  $x$ -axis is the parameter  $i$  at three separate values and the  $y$ -axis is the parameter  $t$  over a large range of values. The widths in the figure represent error of the algorithm; a small width represents a small error.

It is immediately clear that the two lower values tested for  $i$ , 3 and 4, are superior to the higher value of 5 on the problem instances tested. Further, it appears that higher values of  $t$  are preferred over lower ones for all of the values of  $i$  tested, ignoring a single outlier with higher error for low  $i$  and high  $t$ .

This sort of simplistic visual analysis becomes more difficult as the dimensionality of the parameter space increases. It is certainly possible to visualize a three-dimensional parameter space in which the color at each point represents the solution quality of the algorithm with that parameter set, though this technique suffers from difficulties in viewing the interior points in a cubic parameter space with the exterior points in the way. Though visualizations of four-dimensional spaces do exist (see, for instance, [18]), the visualizations do not provide information that is nearly as intuitive, decreasing the simplicity with which the parameter space can be visualized. Certainly no simple visualizations are available for 32-dimensional parameter spaces.

### 21.3.2 Parameter Interactions

This is not the only downside of metaheuristics with a large number of parameters. Another shortcoming is apparent in the susceptibility of a large parameter set to exhibit complex parameter interactions. These complex interactions might lead to, for instance, multiple locally optimal solutions in the parameter space in terms of solution quality. In a more practical optimization sense, this concept of parameter

interaction implies that optimizing parameters individually or in small groups will become increasingly ineffective as the total number of parameters increases.

Parameter interaction is a topic that has been documented in a variety of works. For instance, in [10] the authors observe non-trivial parameter interactions in genetic algorithms with just three parameters. These authors note that the effectiveness of a given parameter mix is often highly based on the set of problem instances considered and the function being optimized, further noting the interdependent nature of the parameters. To a certain extent, it is often very difficult to avoid parameter interactions such as these. In the case of genetic algorithms, for instance, a population size parameter, crossover probability parameter, and mutation probability parameter are typically used, meaning these algorithms will typically have at least the three parameters considered by Deb and Agrawal. However, there have been genetic algorithms developed that operate using only one parameter [32] or none [29], actually eliminating the possibility of parameter interactions.

Though to some degree there is parameter interaction in algorithms with a small number of parameters, we believe that the level of interaction increases dramatically with the number of parameters. To our knowledge, no research has been done on the effects of the number of parameters in a metaheuristic or heuristic on the parameter interactions for that algorithm. However, we propose a simple experiment to test this hypothesis.

First, the experimenter should select a combinatorial optimization problem for which a large number of metaheuristics have been developed. Reasonable choices might be the traveling salesman problem or the vehicle routing problem. Next, the experimenter should obtain implementations of a number of those metaheuristics, preferably of different types (genetic algorithm, tabu search, simulated annealing, ant colony optimization, etc.) and necessarily with a range of number of parameters.

The next step would be to test the parameter interactions using methods designed for this purpose on a representative set of problem instances for the problem considered. One method that could capture parameter interactions of any order would be a full-factorial design, in which a reasonable maximum and minimum value is selected for each parameter and each combination of high and low values for each parameter is tested. However, the number of configurations tested with this method is exponential; a 32-parameter algorithm would require 4,294,967,296 configurations to be tested, which is almost certainly not reasonable. Even a 10-parameter algorithm, which is not uncommon in metaheuristics today, would require tests on over 1,000 configurations, likely a computational burden.

Thus, a better design might be the Plackett–Burman method [27], which requires a number of configurations that is linear in the number of parameters considered. Though this method is limited in that it can only show second-order parameter interactions (the interactions between pairs of parameters), this is not an enormous concern as most parameter interactions are of the second-order variety [24].

In either of these two designs, the number and magnitude of parameter interactions will be measured for each of the algorithms, and a comparison of the intensity

of the interactions will be possible. We believe that not only will the number and magnitude of second-order interactions increase as the size of the parameter set increases, but the same will be true for the higher-order interactions measured through the full-factorial design (if it is possible to use this design).

### 21.3.3 Fair Testing Involving Parameters

Though the effect of parameters on algorithmic simplicity is an important consideration, it is not the only area of interest in parameters while comparing metaheuristics. The other major concern is one of fairness in parameter tuning—if one algorithm is tuned very carefully to the particular set of problem instances on which it is tested, this can make comparisons on these instances unfair. Instead of tuning parameters on all the problem instances used for testing, a fairer methodology for parameter setting involves choosing a representative subset of the problem instances to train parameters on, to avoid overtraining the data. The complementary subset can be used for testing and comparing metaheuristics. A full description of one such methodology can be found in [9].

## 21.4 Solution Quality Comparisons

While it is important to gather a meaningful testbed and to compare the metaheuristics in terms of simplicity by considering their number of parameters, one of the most important comparisons involves solution quality. Metaheuristics are designed to give solutions of good quality in runtimes better than those of exact approaches. To be meaningful, a metaheuristic must give acceptable solutions, for some definition of acceptable.

Depending on the application, the amount of permissible deviation from the optimal solution varies. For instance, in many long-term planning applications or applications critical to a company’s business plan the amount of permissible error is much lower than in optimization problems used for short-term planning or for which the solution is tangential to a company’s business plans. Even for the same problem, the amount of permissible error can differ dramatically. For instance, a parcel company planning its daily routes to be used for the next year using the capacitated vehicle routing problem would likely have much less error tolerance than a planning committee using the capacitated vehicle routing problem to plan the distribution of voting materials in the week leading up to Election Day.

As a result, determining a target solution quality for a combinatorial optimization problem is often difficult or impossible. Thus, when comparing metaheuristics it is not sufficient to determine if each heuristic meets a required solution quality threshold; comparison among the heuristics is necessary.

### 21.4.1 Solution Quality Metrics

To compare two algorithms in terms of solution quality, a metric to represent the solution quality is needed. In this discussion of the potential metrics to be selected, we assume that solution quality comparisons are being made over the same problem instances. Comparisons over different instances are generally weaker, as the instances being compared often have different structures and almost certainly have different optimal values and difficulties.

Of course, the best metric to use in solution quality comparison is the deviation of the solutions returned by the algorithms from optimality. Finding the average percentage error over all problems is common practice, as this strategy gives equal weight to each problem instance (instead of, for instance, giving preference to problem instances with larger optimal solution values).

However, using this metric requires knowledge of the optimal solution for every problem instance tested. However, this is a presupposition that likely cannot always be made. If optimal solutions are available for every problem instance tested upon, the problem instances being considered are likely not large enough, since exact algorithms can provide solutions in reasonable runtimes.

This introduces the need for new metrics that can provide meaningful information without access to the optimal solution for all (or potentially any) problem instances. Two popular metrics that fit this description are deviation from best-known solutions for a problem and deviation between the algorithms being compared.

Deviation from best-known solution or tightest lower bound can be used on problems for which an optimal solution was sought but optimal solutions were not obtained for some problem instances within a predetermined time limit. In these cases, deviation from best-known solution or tightest relaxation is meaningful because for most problem instances the best-known solution or tightest relaxation will be an optimal solution. An example of the successful application of this approach can be found in [14]. In that paper, a metaheuristic, optimal solution, and relaxation of that optimal solution are all created. Though the optimal solution was not run on the largest problem instances due to the excessive runtime required, the low error of the metaheuristic from the optimal solution on the smaller problems (0.25%) reinforces moderate deviations from the relaxed solutions over all problem instances (6.09%).

The metric can also be used for problems for which no optimal solution has been published, though the resulting deviations are less meaningful. It is unclear to a reader how well the algorithm performs without an understanding of how close the best-known solutions or tight lower bounds are to optimal solutions.

Though it also addresses the issue of not having access to optimal solutions, a metric of deviation between the algorithms being compared operates differently—any evaluation of solution quality is done in relation to the other algorithm(s) being considered. This method has the advantage of making the comparison between the algorithms very explicit—all evaluations, in fact, compare the two or more algorithms. However, these comparisons lack any sense of the actual error of solutions. Regardless of how an algorithm fares against another algorithm, its actual error as

compared to the optimal solution is unavailable using this metric. Therefore, using a metric of deviation from another algorithm loses much of its meaningfulness unless accompanied by additional information, such as optimal solutions for some of the problem instances, relaxation results for the problem instances, or deviation from tight lower bounds (to give a sense of the global optimality of the algorithms).

### ***21.4.2 Multiobjective Solution Quality Comparisons***

Though this section has focused on solution quality comparisons of single-objective heuristics, much work has also been done on the comparison of heuristics seeking to optimize multiple objective functions. For a detailed overview of multiobjective optimization and some of the difficulties encountered in comparing multiobjective metaheuristics, see [8]. For an example of the application of metaheuristics to multiobjective optimization problems, see [26].

## **21.5 Runtime Comparisons**

While it is necessary that a metaheuristic demonstrates good solution quality to be considered viable, having a fast runtime is another critical necessity. If metaheuristics did not run quickly, there would be no reason to choose these approaches over exact algorithms.

At the same time, runtime comparisons are some of the most difficult comparisons to make. This is fueled by difficulties in comparing runtimes of algorithms that compiled with different compilers (using different compilation flags) and executed on different computers, potentially on different testbeds.

### ***21.5.1 The Best Runtime Comparison Solution***

The best solution is, of course, to get the source code for the algorithm, compile it on the same computer with the same compilation flags as your own code, and run both algorithms on the same computer. This is certainly the best solution in terms of runtime comparison, as the runtimes for a given problem are then directly comparable. Further, assuming the code can be obtained, this is a relatively simple way to compare the solution qualities. However, this technique for comparing algorithm runtimes is often not possible.

One case in which it is not possible is if the algorithms were programmed in different languages. This implies that their runtimes are not necessarily directly comparable. Though attempts have been made to benchmark programming languages in terms of solution qualities (see, for instance, [4]), these benchmarks are susceptible

to the type of program being run, again rendering any precise comparison difficult. Further invariants in these comparisons include compiler optimizations. The popular C compiler gcc has over 100 optimization flags that can be set to fine-tune the performance of a C program. As most papers do not report compiler optimization flags along with computational results, it would be difficult to obtain the exact scalar multiplier for a C program without additional information. Therefore, while the technique of obtaining a scalar multiplier between programming languages will almost certainly allow comparisons accurate to within an order of magnitude between algorithms coded in different programming languages, these methods cannot provide precise comparisons.

### ***21.5.2 Other Comparison Methods***

It is sometimes not possible to obtain the source code for the algorithm to which we compare. The source code may have been lost (especially in the case of older projects) or the authors may be unwilling to share their source code. While this does make runtime comparison harder, it does not excuse authors from performing these computations—they are critical to the comparison of two algorithms. Two major approaches remain for a researcher to compare runtimes between the two algorithms, each with advantages and disadvantages.

The first is to reimplement another researcher’s code in the same language as your code, running it on the same computer on the same problem instances. This has the advantage of actually running the same algorithm on the same hardware with the same compiler on the same computer, all positive attributes of a comparison. However, this approach suffers from two major weaknesses. First, some algorithms are not clear on certain details of the approach, making an exact reimplementation difficult. While statistical tests can be used to prove that solution qualities returned by the two algorithms are not statistically significantly different between the two implementations, this makes direct comparison of the results more difficult. Second, there is no guarantee that the approach used to reimplement another researcher’s code is really similar to their original code. For instance, the other researcher may have used a clever data structure or algorithm to optimize a critical part of the code, yielding better runtime efficiency. As there is little incentive for a researcher to perform the hard work of optimizing the code to compare against, but much incentive to optimize one’s own code, we believe it is fair to say that reimplementations typically overstate the runtime performance of a new algorithm over an existing one (see [3] for a humorous view of issues such as these).

The other approach does not suffer from these weaknesses. In this approach, published results of an algorithm over a publicly available dataset are compared to a new algorithm’s results on the same dataset. While the dataset being tested is the same and the algorithms being compared are the algorithms as implemented by their developers, the computer used to test these instances is different, and the compiler and compiler flags used are likely also not the same. This approach has the

advantage of simplicity for the researcher—no reimplementation of other algorithms is needed. Further, the implementations of each algorithm are the implementations of their authors, meaning there are no questions about implementation as there were in the reimplementations approach. However, the problem then remains to provide a meaningful comparison between the two runtimes. Researchers typically solve this issue by using computer runtime comparison tables such as the one found in [11] to derive conservative runtime multipliers between the two algorithms. These comparison tables are built by running a benchmarking algorithm (in the case of [11], this algorithm is a system of linear equations solved using LINPACK) and comparing the time to completion for the algorithm. However, it is well known that these sorts of comparisons are imprecise and highly dependent on the program being benchmarked, and the very first paragraph of the benchmarking paper makes sure to mention the limitations of this sort of benchmarking: “The timing information presented here should in no way be used to judge the overall performance of a computer system. The results only reflect one problem area: solving dense systems of equations.” Hence, the multipliers gathered in this way can only provide a rough idea of runtime performance, clearly a downside of the approach.

### 21.5.3 Runtime Growth Rate

Regardless of the comparison method used to compare algorithms’ runtimes, the runtime growth rate can be used as a universal language for the comparison of runtime behaviors of two algorithms. While upper bounds on runtime growth play an important role in the discussion of heuristic runtimes, metaheuristic analysis often does not benefit from these sorts of metrics. Consider, for instance, a genetic algorithm that terminates after a fixed number of iterations without improvement in the solution quality of the best solution to date. No meaningful worst-case analysis can be performed, as there could be many intermediate best solutions encountered during the metaheuristic’s execution. Even in metaheuristics where such analysis is possible (for instance, a genetic algorithm with a fixed number of generations before termination), the worst-case runtime will often not be representative of how the algorithm will actually perform on problem instances, decreasing its value. As a result, the worst-case runtime is a bad choice for asymptotic analysis.

A much better approach for asymptotic analysis is fitting a curve to the runtimes measured for each of the algorithms. Regression analysis is a well-known technique that matches functions to a set of measurement points, minimizing the sum-of-squares error of the matching. These asymptotic results help indicate how an algorithm might perform as the problem size increases. Though there is no guarantee that trends will continue past the endpoint of the sampling (motivating testing on large problem instances), asymptotic runtime trends are key to runtime analyses. Even if one algorithm runs slower than another on small- or medium-sized problem instances, a favorable asymptotic runtime suggests the algorithm may well perform better on large-sized problem instances, where metaheuristics are most helpful.

### 21.5.4 An Alternative to Runtime Comparisons

Though the focus thus far has been on runtime comparisons, there are other forms of computational complexity comparison that do not rely on runtimes. One of the most intriguing, counting the number of representative operations the algorithm uses, is discussed in [1]. In this scheme, the number of a selected set of bottleneck operations is compared without any regard for the total execution time of the algorithms being compared.

There are several clear advantages to this approach over runtime comparisons. As described in [1], it removes the invariants of compiler choice, programer skill, and power of computation platform, providing complexity measures that are easier to replicate by other researchers. However, this approach suffers from the fact that it is often difficult to identify good operations that each algorithm being compared will implement. The only function sure to be implemented by every procedure is the evaluation of the function being optimized. As a result, comparisons of this type often only compare on the optimization function, losing information about other operations, which could potentially be more expensive or more frequently used. As a result, in the context of metaheuristic comparison this technique is best if used along with more traditional runtime comparisons.

## 21.6 Conclusion

We believe following the procedures described in this chapter will increase the quality of metaheuristic comparisons. In particular, choosing an appropriate testbed and distributing it so other researchers can access it will result in more high-quality comparisons of metaheuristics, as researchers will test on the same problem instances. In addition, expanding the practice of creating geometric problem instances with easy-to-visualize optimal or near-optimal solutions will increase understanding of how metaheuristics perform in a global optimization sense.

Furthermore, it is important to recognize that the number of algorithm parameters has a direct effect on the complexity of the algorithm and on the number of parameter interactions, which complicates analysis. If the number of parameters is considered in the analysis of metaheuristics, this will encourage simpler, easier-to-analyze procedures.

Finally, good techniques in solution quality and runtime comparisons will ensure fair and meaningful comparisons are carried out between metaheuristics, producing the most meaningful and unbiased results possible.

## References

1. Ahuja, R., Orlin, J.: Use of representative operation counts in computational testing of algorithms. *INFORMS J. Comput.* **8**(3), 318–330 (1996)

2. Archetti, C., Feillet, D., Hertz, A., Speranza, M.G.: The capacitated team orienteering and profitable tour problems. *J. Oper. Res. Soc.* **60**(6), 831–842 (2009)
3. Bailey, D.: Twelve ways to fool the masses when giving performance results on parallel computers. *Supercomput. Rev.* **4**(8), 54–55 (1991)
4. Bull, M., Smith, L., Pottage, L., Freeman, R.: Benchmarking Java against C and Fortran for scientific applications. In: ACM 2001 Java Grande/ISCOPE Conference, pp. 97–105. ACM, New York (2001)
5. Chao, I.M.: Algorithms and solutions to multi-level vehicle routing problems. Ph.D. thesis, University of Maryland, College Park, MD (1993)
6. Chen, S., Golden, B., Wasil, E.: The split delivery vehicle routing problem: applications, algorithms, test problems, and computational results. *Networks* **49**, 318–329 (2007)
7. Christofides, N., Eilon, S.: An algorithm for the vehicle dispatching problem. *Oper. Res. Q.* **20**(3), 309–318 (1969)
8. Coello, C.: Evolutionary multi-objective optimization: a historical view of the field. *IEEE Comput. Intel. Mag.* **1**(1), 28–36 (2006)
9. Coy, S., Golden, B., Rungger, G., Wasil, E.: Using experimental design to find effective parameter settings for heuristics. *J. Heuristics* **7**(1), 77–97 (2001)
10. Deb, K., Agarwal, S.: Understanding interactions among genetic algorithm parameters. In: Foundations of Genetic Algorithms, pp. 265–286. Morgan Kauffman, San Mateo, CA (1998)
11. Dongarra, J.: Performance of various computers using standard linear equations software. Tech. rep., University of Tennessee (2009)
12. Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT, Cambridge (2004)
13. Fischetti, M., Salazar González, J.J., Toth, P.: A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Oper. Res.* **45**(3), 378–394 (1997)
14. Gamvros, I., Golden, B., Raghavan, S.: The multilevel capacitated minimum spanning tree problem. *INFORMS J. Comput.* **18**(3), 348–365 (2006)
15. Gendreau, M., Laporte, G., Semet, F.: A tabu search heuristic for the undirected selective travelling salesman problem. *Eur. J. Oper. Res.* **106**(2–3), 539–545 (1998)
16. Glover, F.: Tabu search: a tutorial. *Interfaces* **20**(4), 74–94 (1990)
17. Hartmann, S., Kolisch, R.: Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. *Eur. J. Oper. Res.* **127**(2), 394–407 (2000)
18. Hollasch, S.: Four-space visualization of 4d objects. Ph.D. thesis, Arizona State University, Tempe, Arizona (1991)
19. Jans, R., Degraeve, Z.: Meta-heuristics for dynamic lot sizing: a review and comparison of solution approaches. *Eur. J. Oper. Res.* **177**(3), 1855–1875 (2007)
20. Li, F., Golden, B., Wasil, E.: Very large-scale vehicle routing: new test problems, algorithms, and results. *Comput. Oper. Res.* **32**(5), 1165–1179 (2005)
21. Li, F., Golden, B., Wasil, E.: The open vehicle routing problem: algorithms, large-scale test problems, and computational results. *Comput. Oper. Res.* **34**(10), 2918–2930 (2007)
22. Li, F., Golden, B., Wasil, E.: A record-to-record travel algorithm for solving the heterogeneous fleet vehicle routing problem. *Comput. Oper. Res.* **34**(9), 2734–2742 (2007)
23. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs. Springer, New York (1996)
24. Montgomery, D.: Design and Analysis of Experiments. Wiley, New York (2006)
25. Nummela, J., Julstrom, B.: An effective genetic algorithm for the minimum-label spanning tree problem. In: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, pp. 553–557. ACM, New York (2006)
26. Paquete, L., Stützle, T.: Design and analysis of stochastic local search for the multiobjective traveling salesman problem. *Comput. Oper. Res.* **36**(9), 2619–2631 (2009)
27. Plackett, R., Burman, J.: The design of optimum multifactorial experiments. *Biometrika* **33**, 305–325 (1946)
28. Reinelt, G.: TSPLIB—a traveling salesman problem library. *ORSA J. Comput.* **3**(4), 376–384 (1991)

29. Sawai, H., Kizu, S.: Parameter-free genetic algorithm inspired by “disparity theory of evolution”. In: Eiben, A., Bäck, T., Schoenauer, M., Schwefel H.P., (eds.) Parallel Problem Solving from Nature – PPSN V, *LNCS*, vol. 1498, pp. 702–711. Springer Berlin / Heidelberg (1998)
30. Silberholz, J., Golden, B.: The effective application of a new approach to the generalized orienteering problem. *J. Heuristics* **16**(3), 393–415 (2010)
31. Wang, Q., Sun, X., Golden, B.L.: Using artificial neural networks to solve generalized orienteering problems. In: Dagli, C., Akay, M., Chen, C., Fernández, B., Ghosh J., (eds.) Intelligent Engineering Systems Through Artificial Neural Networks, vol. 6, pp. 1063–1068. ASME Press, New York (1996)
32. Xiong, Y., Golden, B., Wasil, E.: A one-parameter genetic algorithm for the minimum labeling spanning tree problem. *IEEE Trans. Evol. Comput.* **9**(1), 55–60 (2005)
33. Xu, J., Kelly, J.: A network flow-based tabu search heuristic for the vehicle routing problem. *Transp. Sci.* **30**(4), 379–393 (1996)

# Subject Index

- A**
- Absorbing state, 580, 609
  - Acceptance criterion, 3, 159, 369–370, 374, 377–380, 382–385, 387–388, 392, 415, 459
  - Acceptance probability, 3, 6, 8–9, 14, 17–18, 22, 24–25
  - Activation bit, 328, 330, 333, 350–351, 355
  - Adaptive memory, 267, 527–528, 558, 560
  - Affine shaker, 552, 561
  - Affinity, 423–424, 426, 428, 432–434, 440–442
  - Agent, 141–142, 159–160, 165–166, 210–211, 227, 231, 338, 424, 427, 437, 473, 545
  - Aggregation, 158, 526, 579, 619
  - Ant colony optimization (ACO), 43, 167, 227–254, 335, 389, 392, 469, 479, 483, 485, 522, 559, 576, 581, 584–586, 591, 593, 600–601, 625, 630, 632
  - Antenna design, 162, 214
  - Antibody, 423, 427, 434–436, 440–442
  - Antigen, 423–427, 430, 432–444, 545
  - Ant system, 165, 228, 234–239, 253, 578, 581, 586
  - Anytime algorithm, 554
  - Arc routing, 339
  - Artificial immune systems (AIS), 7, 44, 421–445
  - Artificial intelligence, 44, 62, 185, 187, 212, 449, 471, 517, 545, 607, 617
  - Aspiration, 47, 55, 333–334
  - Assembly line balancing, 228, 252
  - Asymptotic analysis, 637
  - Asymptotic approximation, 581
  - Asynchronous communication, 504, 517, 519, 521
  - Attractor, 133, 533, 546, 551, 553, 578, 606, 610–611, 613
  - basin, 606, 610–611, 613
- B**
- Augmented objective, 324–327, 329–330, 332, 344
  - Automated theorem proving, 449
  - Average case, 163, 166, 579, 582, 588
- Backtracking, 17, 196, 229, 322, 462, 477, 485–486
- Basin of attraction, 365, 367
- B-cell, 422–425, 433, 435–436
- Beam search, 160, 251, 483
- Behavioral model, 600, 607
- Belief search, 165–166
- Benchmark, 15, 55, 81, 237, 240, 242, 244, 270, 303, 327, 335–338, 349, 379, 382, 385, 387, 416, 437, 456–457, 461, 480, 561–562, 584, 600, 610, 627, 635, 637
- Best-first, 484
- Best-improvement, 231, 500, 551
- Bias function, 290, 292–293
- Big valley, 456
- Bin packing, 212, 270, 340, 412, 416, 454, 456–457, 462–463, 489
- Bioinformatics, 160, 162, 207–208, 211–212, 245
- Blackboard, 165, 517, 519, 523, 526
- Black-box, 364, 574–575, 587, 590–591
- Boltzmann distribution, 3, 583
- Bounding, 11, 14, 482–483
- Branch-and-bound, 72–73, 160, 251–252, 415, 471, 499, 501
- Branch-and-price, 489
- Breakout method, 556
- Building block, 115–116, 124, 290, 310, 450–451, 453, 562

**C**

- Calibration, 55, 413, 497–498, 505, 529  
 Candidate list, 48–49, 99, 269, 284–291, 300, 310, 376, 509  
 Capacity constraint, 144, 339, 405, 477, 488  
 Capital budgeting, 558  
 Car sequencing, 245, 339, 388, 486–487  
 Case-based reasoning, 451, 454  
 Cellular automata, 438  
 Central limit, 604  
 Chaos theory, 533  
 Chromosome, 112–113, 116, 120–123, 126–127, 132–133, 166, 334, 461, 477, 483, 564  
 Classification, 159–160, 245, 267, 273–274, 437, 442–444, 449–465, 470–474, 503–504, 513, 549, 559, 561, 575–576, 601, 607–610, 628  
 Classifier system, 441, 452, 454, 456–457  
 Clause-weighting, 556–557  
 Closure, 194  
 Cluster analysis, 62  
 Clustering, 62, 106, 161–162, 271, 382, 388, 436, 559, 564, 627  
 Coarse-grained, 252, 473, 503, 521–523  
 Column generation, 52, 486, 488–489  
 Compatibility constraint, 341  
 Competitive testing, 600  
 Complexity analysis, 164, 476, 579  
 Compression, 561  
 Computer art, 213  
 Computer vision, 214  
 Conceptual framework, 428, 430, 438–439, 445  
 Confidence interval, 590, 610  
 Conformance testing, 563  
 Constraint  
     programming, 52, 252, 335, 339, 408–409, 470–471  
     satisfaction, 322, 336, 339, 342, 353, 556, 560  
     violation, 51, 342, 352, 354  
 Continuous optimization, 1, 31, 52, 63, 247–248, 391, 552, 554, 558, 561–562  
 Control  
     parameter, 197, 582–583  
     strategy, 471, 473  
 Convergence, 1, 4–5, 7–20, 22–25, 31–32, 63, 92, 119, 125, 127, 151, 154, 217, 234, 247, 253–254, 267, 272, 286, 305, 356, 514, 516–517, 525, 552, 554, 574, 577–579, 585, 593

- Cooling schedule, 2, 4–5, 7, 13, 15–16, 20, 22, 25–29, 31, 458, 512, 553–555

- Cooperative search, 515–530, 532–533  
 Correlation, 161, 382, 391, 442–444, 589–590, 612, 614–617

- Costimulation, 425  
 Coupled neighborhoods, 412  
 Credit assignment, 456  
 Crew scheduling, 29, 559  
 Cross-entropy optimization, 576, 578  
 Crossover, 91, 102–103, 112–113, 115, 117–118, 120, 124–127, 129–133, 150–152, 156, 190–192, 194, 197, 199–201, 206, 217, 334, 391, 422, 433, 479–480, 483, 508, 514, 575, 585, 630, 632

- Curve fitting, 207–209  
 Cutting, 73, 80, 211, 422, 454, 478, 487–488, 525  
 Cutting plane, 488  
 Cyclic exchange, 404  
 Cycling, 44, 46–48, 66, 301, 368, 403, 501, 514, 576

**D**

- Danger signal, 425–426, 437, 442–443  
 Data modelling, 207–209  
 Data parallelism, 502  
 Data structure, 163, 193, 195, 203–204, 286, 326, 443, 516–517, 549–550, 636  
 Decay, 32, 410, 443, 556  
 Decision problem, 52, 144–145, 147, 156, 164, 275, 582  
 Decoder, 473, 478–479, 490  
 Decomposition, 16, 30, 62, 65, 69–70, 72, 269, 289, 386, 409, 464, 473–474, 476, 482, 486–489, 498, 502–507, 510–513, 528, 531  
 Degeneracy, 22, 131, 429, 438  
 Dendritic cell, 422, 425, 428, 429, 437, 442–444  
 Depth-first, 484  
 Descent, 18, 62, 64–66, 68, 70, 270–271, 301–302, 364, 369, 383, 401, 413–414, 456, 460, 475, 481, 500, 551–552, 556, 603–604, 606  
 Detector, 429, 431–434, 439  
 Diffusion, 518, 522, 526, 532, 533, 548  
 Disjunctive constraint, 558  
 Dispatching, 507, 527, 563  
 Disposable heuristic, 451, 461  
 Distance metric, 367, 434  
 Distributed computing, 502

- Diversification, 24, 42, 50–51, 54, 56, 88–89, 266, 269, 273, 279, 291–293, 302, 307, 356, 368, 372, 377–380, 382, 387, 389–391, 393, 408, 411–412, 474–475, 478, 487, 501, 516, 520, 523, 525–527, 529–530, 548–553, 555, 559–560
- Diversity, 89–90, 92, 96–97, 100, 120, 125, 127–129, 133, 154, 191, 217, 273–279, 295, 298, 300–301, 335, 388, 391–392, 413, 433, 443, 514, 516, 522, 524–525, 528–529, 543, 545, 564
- DNA sequencing, 162, 228, 245, 476
- Domain decomposition, 502–504, 510–513, 531
- Double-bridge, 372–373, 375–376, 380, 383–384
- Drawing, 106, 123, 213, 309
- Drift analysis, 581
- Dynamic programming, 160, 337, 471, 479–480, 486–487
- Dynamic system, 117, 533, 549
- Dynasearch, 337, 385, 486
- E**
- Edge recombination, 528
- Ejection chain, 45, 49, 94, 302, 325, 403, 520
- Electric power distribution, 562
- Elite solution, 52, 88, 94, 100–103, 267, 269, 274, 293–295, 300–301, 303, 305, 307, 310, 509, 521, 525, 527–528, 532, 560, 577
- Elitist strategy, 128, 237–238, 247
- Energy, 2–3, 9, 22, 26, 28, 31–32, 80, 208, 336, 553, 561, 564, 583
- Entropy, 16, 23, 27, 154, 253, 576, 578
- Enumeration, 73, 144, 388, 392, 486, 499, 503, 554, 590
- Epitope, 435–436, 440–441
- Estimation-of-distribution algorithm, 576
- Evaluation safety, 194–195
- Evolutionary computation, 117–118, 141, 206, 213–215, 235, 334, 461, 470
- Evolution strategy, 109, 112, 154, 158, 334, 337, 525
- Exchange, 124, 192, 211, 230–231, 252–254, 276–277, 303, 305, 307, 345–346, 374, 385, 387, 402, 404, 411, 473, 487, 504, 512, 515–517, 519–523, 525–527, 529–530, 532–533
- Exploitation, 142, 150, 164, 238–239, 244, 247, 272, 433, 457, 503, 577–578, 585, 593
- Exploration, 46, 48, 54, 88, 102, 111, 151, 154, 234, 238–239, 247, 251, 266, 301, 335, 393, 453, 457, 464, 470, 498–499, 501, 503, 505, 509, 515–516, 519–522, 524, 532–533, 545, 548, 552, 555, 564, 573, 578, 585
- Extremal graph, 79–80
- F**
- Facility layout, 302, 335
- Factorial design, 632–633
- Fast programmable gate array, 561
- Feature selection, 161–162, 208, 212, 335, 509, 526
- Feedback, 5, 29, 236, 438, 451–453, 459–460, 544–545, 550, 554–557, 559, 562
- Filter-and-fan, 403
- Filtering, 9, 301, 399, 424, 435, 444, 489, 512
- Fine-grained, 252, 507, 509, 520
- Finite-time performance, 15–18, 21, 27
- First-improvement, 231, 370, 385, 500, 552
- Fitness, 7, 29, 112–114, 116, 120–123, 125, 127, 129, 132, 147–150, 152, 156–157, 159, 186, 190–191, 194–201, 205, 207, 209, 213, 215–218, 334–335, 456, 477, 503, 508, 514, 521, 546, 557, 574–575, 577, 580, 584–592
- landscape, 147–149, 456, 574, 588, 589, 591, 599–621
- Fixed-parameter tractability, 164–165
- Fleet management, 512
- Functional parallelism, 502
- Fuzzy logic, 471
- G**
- Game theory, 210–211
- Genetic algorithm, 6, 15, 17, 19–20, 30, 44, 52, 88, 91, 102–104, 109–133, 142, 160, 162, 166, 192, 212, 254, 270, 272, 302, 310, 333–334, 337, 422, 427–428, 430, 433, 441, 450, 454, 456, 469, 502, 508, 514, 522, 528, 547, 558, 575, 600–601, 625, 629–630, 632, 637
- Genetic programming, 185–219, 450–452, 461–464, 600
- Genomic median, 564
- Genotype, 111–112, 115, 120, 130, 145, 205–206, 478, 483
- Gradient, 29, 218, 253, 270, 482, 557, 619
- Grammar, 159, 198–202, 215, 219
- Graph
- coloring, 28, 161, 245, 476, 486, 489, 582
  - partitioning, 27–28, 476, 514, 521, 523–524, 526, 559
  - theory, 8, 52, 62–63, 79–80, 82, 163, 559
- Gray-box, 591

- Great deluge, 458  
 Greedy heuristic, 231, 268, 284, 406, 415, 487, 489  
 Greedy randomized adaptive search procedure (GRASP), 94, 102–103, 105, 267, 269, 272, 274–275, 283–311, 388, 390, 392, 470, 474, 601  
 Grid computing, 272, 303, 307–309  
 Ground state, 583  
 Growth function, 145–147  
 Guided local search (GLS), 321–356, 386, 389, 460, 501, 557  
 Guiding function, 145, 147–148, 154
- H**  
 Halin graph, 405  
 Hamming distance, 73–74, 78, 95–97, 100, 146, 268, 287, 295, 387, 549, 584  
 Hashing table, 301  
 Heuristic generation, 461–464  
 Heuristic selection, 451–461  
 Hill-climbing, 1–2, 14–15, 17–22, 24, 27, 110, 270, 522  
 History, 2, 44, 51, 91, 206, 235–240, 273, 277, 290, 356, 364, 369, 374, 377–378, 380, 385, 390, 411, 526, 546, 548, 551, 553, 555, 559, 590  
 Hit-and-run algorithm, 5  
 Human guided search, 471  
 Hybrid, 20, 30, 100–101, 142, 151, 160, 162, 272–273, 289, 292, 301, 302, 307–308, 334–335, 337–338, 392, 435–436, 453, 470–471, 473, 477, 478, 480–481, 487, 489–490, 560  
 Hybridization, 52, 100–106, 142, 250–251, 302, 335, 471, 483, 522  
 Hyper-heuristic, 212, 414, 449–465  
 Hypermutation, 423–425, 433, 436, 445
- I**  
 Idiotope, 435, 441  
 Idiotypic network, 422, 426–427, 428–429, 435–436, 439–442, 445  
 Image processing, 210  
 Immune system, 7, 44, 421–445  
 Independent set, 305, 334, 480, 582  
 Inequality constraint, 72, 353–354  
 Infectious nonself, 425–426  
 Initial population, 101, 113, 118–119, 153, 188–190, 200, 218, 302, 375, 481–482, 502, 514, 525  
 Innate system, 422, 425  
 Intensification, 24, 42, 49–50, 54–55, 88, 92, 94, 103, 267, 269, 292–295, 300–302, 305, 307, 310, 354, 368, 377–380, 382, 390, 393, 411, 474–475, 478, 501, 516, 529, 549, 560  
 Interchange, 45, 65, 126, 146, 385–386, 404, 411  
 Interior point, 481  
 Island model, 143, 272, 473, 520, 523  
 Iterated local search (ILS), 302, 363–393, 414, 456, 470, 480, 487, 501, 546, 552, 576, 603  
 Iterative improvement, 230–231, 250, 372, 381, 387, 389, 390
- K**  
 $k$ -cardinality tree, 66, 479–480, 488  
 Kernelization, 476  
 K-means, 433  
 Knapsack, 73, 88, 129, 144, 161, 245, 268, 270, 340, 476–477, 480–482, 521, 558, 563  
 $k$ -Opt, 343, 345, 484, 603
- L**  
 Lagrangian relaxation, 477–478  
 Landscape, 22–23, 28, 116, 145–149, 156, 338, 352–354, 356, 454, 456, 564, 574, 587–591, 599–621  
 Large neighborhood, 22–23, 52, 160, 388, 399–417, 473, 475, 486–488, 490  
 Lateral thinking, 547–548  
 LeadingOnes, 584–586  
 Learning, 69, 110, 159–161, 163–164, 166, 185, 202, 208, 212, 215, 219, 231, 238, 245, 253, 267–268, 273, 290–293, 334, 388, 424, 429–431, 436, 438, 441, 450–453, 456, 458, 461, 464, 528, 532, 543–564, 581, 585–586  
 rate, 556, 581, 585–586  
 Linear programming, 46, 72–75, 163, 471, 478  
 Lin-Kernighan, 156, 163, 302, 337, 343, 364, 373, 380, 383–384, 387, 403, 479–480  
 Local access network, 475  
 Local branching, 73–74, 78–79, 484–485, 488, 591  
 Local search (LS), 1–2, 15–22, 43, 50, 63–70, 73, 75–79, 94–96, 142–143, 148–149, 152, 153–165, 228, 230–231, 233–234, 242–243, 247–248, 250–251, 267, 270–276, 284–290, 294–297, 301–302, 305, 310, 321–356, 363–393, 412–414, 433, 445, 450, 452, 456, 458, 461–462, 464, 470–471, 474–475, 480, 484–488, 499–502, 507, 509, 511–512, 520, 523, 545–564, 576, 589–590, 599–621

- Location, 4, 44–45, 62–72, 155, 200, 209, 291, 302, 309, 346–348, 374–375, 388, 458, 509, 521, 525, 558, 563  
Logic programming, 166, 339  
Look-ahead, 509  
Lot sizing, 28, 62, 161
- M**  
Machine learning, 110, 160–161, 185, 208, 212, 215, 219, 429–430, 436, 438, 451–461, 464, 543–545, 557–558, 561  
Manufacturing, 161, 212, 309  
Marginal conditional validity, 268  
Markov chain, 2–3, 5, 7–8, 10–14, 16–17, 19–20, 23, 31, 116–117, 302, 364, 377, 383, 552, 580, 607, 609, 619–620  
Martingale, 13, 581  
Matching, 29, 161, 404–405, 423, 431, 434–435, 440–441, 480, 548, 584, 637  
Mathematical programming, 6, 28, 62, 71, 248, 251, 502, 591–592  
Maximum diversity, 92, 274–279  
MAX-SAT, 105, 160–161, 287, 289, 296, 338, 342, 353–354, 374, 382, 387–388, 555, 602–603, 606, 611–612, 614, 619–621  
Memetic algorithm, 141–167, 272, 337, 389, 391–392, 473–475, 480  
Memory, 19, 44, 46, 49, 50, 92, 203, 218, 236–237, 243, 266–269, 273–277, 279, 290, 293, 305, 310, 341, 347, 365, 367–369, 371, 377–378, 381, 390–391, 393, 427, 433, 436, 473, 478, 508, 517, 519, 523–533, 546, 548, 552–553, 555, 557–558, 560, 564, 574–575, 577, 578, 580–582, 585, 587, 590, 619, 630  
Merging, 163, 384–385, 473, 476, 479–480  
Metropolis, 3, 9–10, 27, 585  
Migration, 5, 253, 521, 523, 525  
Minimax principle, 590  
Monte-Carlo, 9, 21, 31, 65, 267  
Multi-level, 476, 514, 526, 532  
Multi-objective, 5–7, 211, 217–218, 268, 272, 393, 435, 521, 584, 593  
Multi-point constructive search, 477  
Multi-search, 498, 503–504, 509–510, 513–515, 522, 531–532  
Multi-stage, 475–477, 480, 486, 490  
Multi-start, 24, 265–279, 285, 293, 389–392, 470, 513, 532  
Mutation, 20, 110, 112–113, 115, 117, 120, 126–128, 131–133, 148–150, 152, 154–155, 191–193, 197, 199–200, 205–206, 215, 217, 339, 433–434, 477, 480–482, 508, 575, 585–586, 630, 632
- N**  
Natural language parsing, 340  
Nearest neighbor, 230, 376  
Needle-in-a-haystack (NIAH), 584  
Neighborhood  
    function, 2, 21, 23–24, 30  
    structure, 23, 42, 44–46, 49, 53–54, 56, 63, 68, 94, 146, 154, 230, 285, 294, 365, 367, 389, 392, 404, 475, 479, 487–488, 575–576, 586, 588–590  
Nested neighborhoods, 385  
Nested partitioning, 485  
Network design, 62, 73, 299, 305, 388, 414, 475, 487, 514, 521, 525–526  
Network flow, 402, 404–405  
Neural network, 117, 161, 205, 215, 245, 422, 428, 444, 471, 545, 558, 560–561  
No-free-lunch theorem, 115, 142, 574, 587–589  
Noise, 18, 22, 26, 197, 249, 291–292, 556, 574, 592–593  
Noising, 17–19, 22, 291, 410  
Nonlinear programming, 63, 70, 77–79, 471  
Number partitioning, 28, 582–583, 616
- O**  
Offspring, 91, 102, 112–113, 120, 124–125, 128, 130, 150, 156, 186, 191–193, 195, 200, 238, 272, 479–480, 483, 507, 530  
OneMax, 582, 584–586, 591  
Online, 120, 158, 343, 452–454, 458, 461–463, 544–547, 550–551  
Optical system design, 214  
 $\lambda$ -Opt, 500  
1-Opt, 331, 353–355  
2-Opt, 156, 328, 331, 343–345, 349, 373–374, 380, 383, 401–402, 529, 603  
3-Opt, 111, 343, 345, 371, 373, 376, 379–380, 383, 529  
4-Opt, 380, 384  
Or-opt, 529  
Overtraining, 633
- P**  
Packing, 70, 73, 105, 212, 270, 291, 296, 339–340, 412, 416, 454, 456–458, 460–463, 478, 480, 483, 489  
Pallet loading, 483  
Parallelism, 29, 114–115, 163, 302, 498–505, 507–508, 510  
Parallelization, 133, 252, 303, 307, 473, 498, 503, 505–510, 513–514, 520–521, 527–528, 531–532

- Parameter  
calibration, 55, 498  
interaction, 632  
space, 626, 629–631  
tuning, 291, 302, 546, 557, 629, 633
- Paratope, 435–436, 440–441
- Partial solution, 229, 233, 237, 242, 248, 251, 271, 284, 286, 290–291, 386, 408, 454, 456, 477, 483, 530
- Particle swarm optimization, 44, 158, 167
- Partition function, 583
- Path exchange, 487
- Pathogen associated molecular pattern (PAMP), 425–426
- Path relinking, 87–106, 267–269, 294–301, 303, 305, 310–311, 502, 509, 525, 528, 602
- Pattern recognition, 161, 422, 430, 433, 437, 561
- Penalty, 22, 51, 76, 277, 323–325, 335–336, 341, 344–345, 347–348, 350, 354, 356, 388, 557
- Permutation, 25, 30–31, 103, 119, 131, 146, 237, 241, 246, 254, 331, 343, 346, 349–351, 371, 385, 477, 478, 588, 613
- Persistent attractiveness, 268
- Perturbation, 14, 30–31, 62, 292, 302, 368–377, 379–385, 387–393, 452–453, 457–461, 464, 546, 551–552, 554, 603
- Phase transition, 29, 553, 582–583, 609, 611, 616–617
- Phenotype, 111–112, 120, 130, 145, 205–206, 217, 478, 482
- Pheromone, 227–228, 230–250, 252–254, 508–509, 512, 521, 576, 578, 581, 630
- Pickup and delivery, 414–415, 460, 560
- Plackett-Burman method, 632
- P-median, 65, 480, 509, 514, 520, 524
- Policy, 127–128, 252–253, 549
- Pool, 100–101, 121, 142, 154, 293, 295, 297, 300–301, 303, 305, 307, 431, 450, 459–460, 476, 517, 525, 529
- Population, 6, 7, 19, 20, 91–93, 101–103, 112–115, 118–119, 121–129, 141–142, 148–149, 152–160, 166, 186–190, 193, 195, 197, 199–200, 202, 204, 215–218, 231, 239, 252, 300–302, 334, 364, 375, 384, 388, 391–392, 423, 431, 434, 438, 443–444, 456, 458, 477, 481–483, 498, 502, 504–509, 514–515, 519–532, 547, 575–577, 580, 601, 629–630, 632
- Portfolio optimization, 162
- Positional bias, 124, 456
- Potential function, 581
- Power system, 162, 291, 309
- Premature convergence, 119, 125, 127, 154, 286, 514, 578–579
- Pricing, 489–490
- Primitive set, 187, 189, 192, 194–195
- Probabilistic feedback, 5
- Probabilistic tabu search, 19
- Process control, 211
- Production planning, 514, 563
- Prohibition, 547–550, 552, 556, 559–561
- Propagation, 244, 252, 486, 489–490, 560
- Protein folding, 228, 245, 555
- Proximate optimality principle (POP), 152–155, 197, 294, 474
- Pseudo-utility ratio, 482
- Q**
- Quadratic assignment problem (QAP), 27, 30–31, 111, 251, 267, 302, 335, 341–342, 346–347, 374, 388, 480, 510, 558
- Quantum computing, 207
- Quasirandom sample, 271
- R**
- Randomization, 88, 91, 94, 98–99, 163, 273, 292, 301, 412, 500, 606
- Random restart, 15–16, 268, 366, 368–369, 371–372, 374–376, 378–380, 387
- Random search, 115, 577–578, 587–588
- Random walk, 43, 369, 377, 552, 556, 589–590, 617
- Ranking, 122–123, 524
- Reactive search, 374, 387, 543–564
- Real time, 52, 343, 393, 437–438, 512, 561, 563
- Real-world problem, 340, 476, 545, 558, 562, 593, 626–627
- Receptor, 423–426, 429–430, 433, 435, 438
- Recombination, 110–112, 115, 120, 127, 142–143, 149–152, 154, 159–160, 163–166, 191–193, 206, 391–392, 473, 479–483, 528, 601
- Reference set, 89–92, 94, 102, 509
- Regression, 198, 206–209, 607–610, 637
- Reinforcement learning, 231, 238, 253, 441, 452, 545, 557
- Relaxation, 51–52, 160, 342, 405, 473–474, 477–478, 481–482, 485, 488, 490, 628, 634–635
- Reliability, 6, 62, 563
- Relocate, 401–402, 406
- Removal, 80, 166, 405, 407, 414, 461

- Repair, 80, 151, 206, 284–285, 322, 400, 406–416, 460, 474, 481, 488  
 Resource allocation, 341–342, 348, 350, 563  
 Restart, 15–16, 49–50, 151, 155, 268, 270, 328, 330, 366, 368–372, 374–376, 378–380, 382, 387–388, 392, 520, 547  
 Reusable heuristic, 451, 461  
 Risk analysis, 162  
 Robot, 186, 193–197, 208–209, 213, 429, 436, 438, 441–442, 553  
 Rostering, 29, 161, 459  
 Ruin and recreate, 409  
 Runtime comparison, 635–638  
 Rural postman, 337
- S**
- Safe signal, 442–443  
 Sampling, 5, 9, 21, 23, 49, 51, 55, 92, 119, 121, 123, 190, 271–273, 287, 291, 366–369, 382, 392, 443, 485, 552, 554, 560, 585, 587, 591–592, 637  
 Satisfiability (SAT), 98, 105, 147, 296, 305, 338, 461–463, 489, 521, 555, 560, 601–602  
 Scatter search, 87–106, 163, 267, 272, 294, 389, 391–392, 470, 481, 498, 502, 504, 509, 514, 602  
 Scenario, 143, 157, 245, 336, 341, 454, 545–546, 590–591  
 Scheduling, 24, 27–31, 52, 62, 106, 125, 160–161, 228, 241–242, 245–246, 249, 252, 266, 270–272, 291, 309, 322, 337–338, 340–342, 348–351, 375, 381, 385–388, 403–405, 414, 416, 454, 458–459, 461, 474–477, 483, 486–487, 499, 510, 513, 525, 559, 563, 582, 611, 628  
 Schema, 114–118  
 Search  
   threads, 303, 305, 503–504, 510, 513–517, 519, 521–528, 530, 532–533  
   trajectory, 548–549, 551  
   tree, 164, 251, 475, 483–484, 489  
 Security, 210, 431–432, 438, 442  
 Selection pressure, 122–123, 127, 191  
 Self-adaptation, 159, 206  
 Self-nonsense, 423–427, 429–431, 437  
 Self-tuning, 18, 289, 544, 562–563  
 Sequencing, 18, 131, 162, 228, 245, 339, 388, 476, 486–487, 563  
 Set covering, 95–96, 98, 161, 241–243, 245, 269, 289, 309, 480, 514, 527, 559  
 Shaking, 65, 67–68, 74, 77  
 Shared memory, 473, 508  
 Shortest path, 6, 292, 404, 487  
 Signal processing, 29, 207, 210, 426, 437, 443  
 Simulated annealing (SA), 1–32, 100, 115, 142, 217, 271, 301, 322, 337, 369, 372, 377, 381, 383, 387, 389, 408, 414–415, 458–461, 469, 498, 501, 507–508, 512, 514, 521, 524, 531, 548, 553, 558, 575–576, 600, 604, 616, 630, 632  
 Simulation, 22–23, 31, 210–211, 266, 421, 437, 471, 554, 563, 590, 609  
 Smoothing, 22, 354, 557, 560  
 Soft constraint, 352–354, 454, 544  
 Soft sensor, 208  
 Spanning tree, 161, 268, 302, 307–308, 404, 482, 487–488, 559, 584–585, 629  
 Speedup, 270, 303–305, 311, 505, 508, 529  
 Squeaky wheel, 462  
 Stagnation, 47, 154, 247, 379, 384, 557, 564, 619  
 State space, 27, 580, 582, 601–602, 605, 619  
 Statistical mechanics, 9, 43, 116, 582–583, 590  
 Statistics, 4, 9, 16, 30–31, 43, 54, 115–116, 119–121, 212, 244, 270, 458, 471, 516, 519, 527–528, 545, 552, 556, 582–583, 590, 607–609, 636  
 Steiner tree, 161, 292, 309–311, 482  
 Step size, 4, 555, 562  
 Stimulation, 424, 427, 436, 439–441  
 Stochastic search, 119, 366–369, 552, 561, 573–593  
 Strategic oscillation, 50–51, 268, 375, 390, 554  
 Supply chain, 161, 488  
 Support vector machine, 561  
 Suppression, 436, 440–441  
 Surrogate, 22, 51–52, 54, 129, 156  
 Swap, 45, 96, 126, 149, 192, 208, 331, 346, 350–351, 385, 402, 404, 510, 559  
 Swarm intelligence, 254, 584  
 Synchronization, 503–504, 509, 512–513, 519–522, 525–526, 562  
 Synchronous communication, 504, 517  
 System dynamics, 578

**T**

Tabu list, 19, 27, 42, 44, 46–48, 334, 354, 356, 387–388, 412, 459, 512, 560, 576, 629–630

Tabu search, 6, 15, 17, 19–21, 24, 41–56, 73, 94, 100, 115, 251, 267–269, 272–274, 276–277, 279, 293–294, 301–302, 310, 322, 333–335, 337, 339, 347, 354, 356, 374–375, 377, 381–382, 387, 389, 454–456, 458–460, 469, 474–475, 477–478, 486–489, 498, 501, 507, 509–510, 512–514, 516, 520–521, 523–531, 548, 550–551, 554, 556, 558–561, 563–564, 573, 600, 607, 611–614, 618–620, 625, 629–630, 632

Tabu tenure, 46–47, 549, 559

T-cell, 422, 424–428, 430, 433, 437–439

Team orienteering, 337

Telecommunications, 62, 160, 162, 233, 243, 245, 309, 514, 521, 559, 563

Temperature, 2–4, 6–7, 9, 11–13, 15–16, 18–19, 21–22, 24–27, 29–32, 369, 377, 386–387, 389, 408, 508, 514, 553–555, 578, 583–585, 630

Termination, 5, 48, 113, 119–120, 148, 197, 335, 356, 369, 407, 457, 464, 481–482, 485, 520, 629, 637

Testbed, 626–628, 633, 635, 638

Threshold, 17–18, 20–21, 24, 48, 120, 154, 270, 286, 293, 301, 335, 377, 429, 431–432, 439–440, 564, 607, 633

Threshold accepting (TA), 17–18, 20–21, 24, 335

Tie-breaking rule, 550

Timetabling, 29, 160–161, 245, 309, 452, 454–456, 458–459, 460, 483

Tolerance, 63, 272, 423–425, 429, 431, 633

Transition probability, 3, 10, 12–13, 27, 32, 581, 609, 619

Transportation, 44, 46, 249, 309, 460, 559

Traveling salesman, 17–18, 22, 27–28, 30–32,

73, 142, 228, 245, 270, 272, 284, 400, 403, 476, 478–480, 486–487, 510, 546, 602, 612, 615–616, 627, 632

Trial-and-error, 586

Type consistency, 194

**U**

Utility, 147, 206, 323–326, 332, 335–336, 338, 345, 350, 354, 356, 459, 601

**V**

Valid inequality, 488

Value-ordering, 477

Variable depth, 325, 386–387, 402–403, 414, 559

Variable fixing, 476–477

Variable neighborhood, 24, 61–82, 100, 301, 310–311, 375, 389, 413, 470, 475, 481, 486–488, 498, 501, 514, 523, 551, 554, 576, 600, 630

Variable-ordering, 477

Variable-sample, 592

Vector quantization, 561

Vehicle routing, 42–43, 45, 52, 62, 245, 249–250, 271, 335, 339, 341, 350, 388, 400, 403, 412–413, 458, 460–461, 475–476, 487–488, 500, 507, 527, 529, 552, 560, 562–563, 628–629, 632–633

Vertex cover, 164, 476, 582, 584

Very large-scale neighborhood, 402–405

Visualization, 486, 629–631

VLSI design, 162, 340

Volume algorithm, 482

**W**

Warping, 557

Worst case, 152, 163, 228, 243, 290, 322, 560, 579, 582, 590–591, 608–609, 615, 637