PARALLEL METAHEURISTICS

A New Class of Algorithms

Edited by

Enrique Alba

WILEY-INTERSCIENCE

A JOHN WILEY & SONS, INC., PUBLICATION
PARALLEL METAHEURISTICS
PARALLEL METAHEURISTICS

A New Class of Algorithms

Edited by

Enrique Alba

A JOHN WILEY & SONS, INC., PUBLICATION
# Contents

Foreword xi
Preface xiii
Contributors xv

Part I INTRODUCTION TO METAHEURISTICS AND PARALLELISM 1

1 An Introduction to Metaheuristic Techniques 3
   Christian Blum, Andrea Roli, Enrique Alba
   1.1 Introduction 3
   1.2 Trajectory Methods 8
   1.3 Population-Based Methods 19
   1.4 Decentralized Metaheuristics 28
   1.5 Hybridization of Metaheuristics 29
   1.6 Conclusions 31
       References 31

2 Measuring the Performance of Parallel Metaheuristics 43
   Enrique Alba, Gabriel Luque
   2.1 Introduction 43
   2.2 Parallel Performance Measures 44
   2.3 How to Report Results 48
   2.4 Illustrating the Influence of Measures 54
   2.5 Conclusions 60
       References 60

3 New Technologies in Parallelism 63
   Enrique Alba, Antonio J. Nebro
   3.1 Introduction 63
   3.2 Parallel Computer Architectures: An Overview 63
   3.3 Shared-Memory and Distributed-Memory Programming 65


<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Authors</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>Parallel Evolution Strategies</td>
<td>Günter Rudolph</td>
<td>155</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7.1</td>
<td>Introduction</td>
<td></td>
<td>155</td>
</tr>
<tr>
<td>7.2</td>
<td>Deployment Scenarios of Parallel Evolutionary Algorithms</td>
<td></td>
<td>156</td>
</tr>
<tr>
<td>7.3</td>
<td>Sequential Evolutionary Algorithms</td>
<td></td>
<td>159</td>
</tr>
<tr>
<td>7.4</td>
<td>Parallel Evolutionary Algorithms</td>
<td></td>
<td>159</td>
</tr>
<tr>
<td>7.5</td>
<td>Conclusions</td>
<td></td>
<td>165</td>
</tr>
<tr>
<td></td>
<td>References</td>
<td></td>
<td>165</td>
</tr>
<tr>
<td>8</td>
<td>Parallel Ant Colony Algorithms</td>
<td>Stefan Janson, Daniel Merkle, Martin Middendorf</td>
<td>171</td>
</tr>
<tr>
<td>8.1</td>
<td>Introduction</td>
<td></td>
<td>171</td>
</tr>
<tr>
<td>8.2</td>
<td>Ant Colony Optimization</td>
<td></td>
<td>172</td>
</tr>
<tr>
<td>8.3</td>
<td>Parallel ACO</td>
<td></td>
<td>175</td>
</tr>
<tr>
<td>8.4</td>
<td>Hardware Parallelization of ACO</td>
<td></td>
<td>190</td>
</tr>
<tr>
<td>8.5</td>
<td>Other Ant Colony Approaches</td>
<td></td>
<td>195</td>
</tr>
<tr>
<td></td>
<td>References</td>
<td></td>
<td>197</td>
</tr>
<tr>
<td>9</td>
<td>Parallel Estimation of Distribution Algorithms</td>
<td>Julio Madera, Enrique Alba, Alberto Ochoa</td>
<td>203</td>
</tr>
<tr>
<td>9.1</td>
<td>Introduction</td>
<td></td>
<td>203</td>
</tr>
<tr>
<td>9.2</td>
<td>Levels of Parallelism in EDA</td>
<td></td>
<td>204</td>
</tr>
<tr>
<td>9.3</td>
<td>Parallel Models for EDAs</td>
<td></td>
<td>206</td>
</tr>
<tr>
<td>9.4</td>
<td>A Classification of Parallel EDAs</td>
<td></td>
<td>216</td>
</tr>
<tr>
<td>9.5</td>
<td>Conclusions</td>
<td></td>
<td>219</td>
</tr>
<tr>
<td></td>
<td>References</td>
<td></td>
<td>220</td>
</tr>
<tr>
<td>10</td>
<td>Parallel Scatter Search</td>
<td>F. García, M. García, B. Melián, J. A. Moreno-Pérez, J. M. Moreno-Vega</td>
<td>223</td>
</tr>
<tr>
<td>10.1</td>
<td>Introduction</td>
<td></td>
<td>223</td>
</tr>
<tr>
<td>10.2</td>
<td>Scatter Search</td>
<td></td>
<td>224</td>
</tr>
<tr>
<td>10.3</td>
<td>Parallel Scatter Search</td>
<td></td>
<td>225</td>
</tr>
<tr>
<td>10.4</td>
<td>Application of Scatter Search to the $p$-Median Problem</td>
<td></td>
<td>229</td>
</tr>
<tr>
<td>10.5</td>
<td>Application of Scatter Search to Feature Subset Selection</td>
<td></td>
<td>232</td>
</tr>
<tr>
<td>10.6</td>
<td>Computational Experiments</td>
<td></td>
<td>239</td>
</tr>
<tr>
<td>10.7</td>
<td>Conclusions</td>
<td></td>
<td>243</td>
</tr>
<tr>
<td></td>
<td>References</td>
<td></td>
<td>244</td>
</tr>
</tbody>
</table>
11 Parallel Variable Neighborhood Search
José A. Moreno-Pérez, Pierre Hansen, Nenad Mladenović
11.1 Introduction
11.2 The VNS Metaheuristic
11.3 The Parallelizations
11.4 Application of VNS for the p-median
11.5 Computational Experiments
11.6 Conclusions
References

12 Parallel Simulated Annealing
M. Emin Aydin, Vecihi Yiğit
12.1 Introduction
12.2 Simulated Annealing
12.3 Parallel Simulated Annealing
12.4 A Case Study
12.5 Summary
References

13 Parallel Tabu Search
Teodor Gabriel Crainic, Michel Gendreau, Jean-Yves Potvin
13.1 Introduction
13.2 Tabu Search
13.3 Parallelization Strategies for Tabu Search
13.4 Literature Review
13.5 Two Parallel Tabu Search Heuristics for Real-Time Fleet Management
13.6 Perspectives and Research Directions
References

14 Parallel Greedy Randomized Adaptive Search Procedures
Mauricio G.C. Resende, Celso C. Ribeiro
14.1 Introduction
14.2 Multiple-Walk Independent-Thread Strategies
14.3 Multiple-Walk Cooperative-Thread Strategies
14.4 Some Parallel GRASP Implementations
14.5 Conclusion
References
15 Parallel Hybrid Metaheuristics
Carlos Cotta, El-Ghazali Talbi, Enrique Alba

15.1 Introduction
15.2 Historical Notes on Hybrid Metaheuristics
15.3 Classifying Hybrid Metaheuristics
15.4 Implementing Parallel Hybrid Metaheuristics
15.5 Applications of Parallel Hybrid Metaheuristics
15.6 Conclusions

References

16 Parallel Multiobjective Optimization
Antonio J. Nebro, Francisco Luna, El-Ghazali Talbi, Enrique Alba

16.1 Introduction
16.2 Parallel Metaheuristics for Multiobjective Optimization
16.3 Two Parallel Multiobjective Metaheuristics
16.4 Experimentation
16.5 Conclusions and Future Work

References

17 Parallel Heterogeneous Metaheuristics
Francisco Luna, Enrique Alba, Antonio J. Nebro

17.1 Introduction
17.2 Heterogeneous Metaheuristics Survey
17.3 Taxonomy of Parallel Heterogeneous Metaheuristics
17.4 Frameworks for Heterogeneous Metaheuristics
17.5 Concluding Remarks
17.6 Annotated Bibliography

References

Part III THEORY AND APPLICATIONS

18 Theory of Parallel Genetic Algorithms
Erick Cantú-Paz

18.1 Introduction
18.2 Master-Slave Parallel GAs
18.3 Multipopulation Parallel GAs
18.4 Cellular Parallel GAs
18.5 Conclusions

References
19 Parallel Metaheuristics Applications
   Teodor Gabriel Crainic, Nourredine Hail

   19.1 Introduction
   19.2 Parallel Metaheuristics
   19.3 Graph Coloring
   19.4 Graph Partitioning
   19.5 Steiner Tree Problem
   19.6 Set Partitioning and Covering
   19.7 Satisfiability Problems
   19.8 Quadratic Assignment
   19.9 Location Problems
   19.10 Network Design
   19.11 The Traveling Salesman Problem
   19.12 Vehicle Routing Problems
   19.13 Summary
   References

20 Parallel Metaheuristics in Telecommunications
   Sergio Nesmachnow, Héctor Cancela, Enrique Alba, Francisco Chicano

   20.1 Introduction
   20.2 Network Design
   20.3 Network Routing
   20.4 Network Assignment and Dimensioning
   20.5 Conclusions
   References

21 Bioinformatics and Parallel Metaheuristics
   Oswaldo Trelles, Andrés Rodriguez

   21.1 Introduction
   21.2 Bioinformatics at a Glance
   21.3 Parallel Computers
   21.4 Bioinformatic Applications
   21.5 Parallel Metaheuristics in Bioinformatics
   21.6 Conclusions
   References

Index
Foreword

Metaheuristics are powerful classes of optimization techniques that have gained a lot of popularity in recent years. These techniques can provide useful and practical solutions for a wide range of problems and application domains. The power of metaheuristics lies in their capability in dealing with complex problems with no or little knowledge of the search space, and thus they are particularly well suited to deal with a wide range of computationally intractable optimizations and decision-making applications.

Rather simplistically, one can view metaheuristics as algorithms that perform directed random searches of possible solutions, optimal or near optimal, to a problem until a particular termination condition is met or after a predefined number of iterations. At the first instance, this can be seen as a drawback because the search for a solution may take too much time to an extent that renders the solution impractical.

Fortunately, many classes of metaheuristics are inherently parallelizable and this led researchers to develop parallelization techniques and efficient implementations. Of course, in some metaheuristics, parallelization is much easier to achieve than in others, and with that comes issues of implementation on actual parallel platforms. In earlier implementations the master-slave paradigm was the preferred model used to run metaheuristics and still is a valid approach for many classes of these algorithms. However, due to the great variety of computer architectures (shared memory processors, clusters, grids, etc.) other approaches have been developed and more concerted work is needed in this direction. Moreover, another important issue is that of the development of parallelization tools and environments that ease the use of metaheuristics and extend their applicability range.

Professor Alba’s new book, Parallel Metaheuristics, is a well-timed and worthy effort that provides a comprehensive and balanced blend of topics, implementations, and case studies. This volume will prove to be a very valuable resource for researchers and practitioners interested in using metaheuristics to solve problems in their respective disciplines. The book also serves as a repository of significant reference material as the list of references that each chapter provides will serve as a useful source of further study.

Professor Albert Y. Zomaya
CISCO Systems Chair, Professor of Internetworking
The University of Sydney, Australia

May 2005
This Page Intentionally Left Blank
Preface

The present book is the result of an ambitious project to bring together the various visions of researchers in both the parallelism and metaheuristic fields, with a main focus on optimization. In recent years, devising parallel models of algorithms has been a healthy field for developing more efficient optimization procedures. What most people using these algorithms usually miss is the important idea that parallel models that run in multiple computers are quite modified versions of the sequential solvers they have in mind. This of course means that not only the resulting algorithm is faster in wall clock time, but also that the underlying algorithm performing the actual search is a new one. These new techniques have their own dynamics and properties, many of them coming from the kind of separate decentralized search that they perform, while many others come from their parallel execution.

Creating parallel metaheuristics is just one way for improving an algorithm. Other different approaches account for designing hybrid algorithms (merging ideas from existing techniques), creating specialized operations for the problem at hand, and a plethora of fruitful research lines of the international arena. However, designing parallel metaheuristics has an additional load of complexity, since doing it appropriately implies that the researcher must have background knowledge from the two combined fields: parallelism and metaheuristics. Clearly, this is difficult, since specialization is a must nowadays, and these two fields are naturally populated by often separate groups of people. Thus, many researchers in mathematics, engineering, business, physics, and pure computer science deal quite appropriately with the algorithms, but have no skills in parallelism. Complementary, many researchers in the field of parallelism are quite skilled with parallel software tools, distributed systems, parallel languages, parallel hardware, and many other issues of high importance in complex applications; but the problem arises since these researchers often do not have deep knowledge in metaheuristics. In addition, there are also researchers who are application-driven in their daily work; they only want to apply the techniques efficiently, and do not have the time or resources (nor maybe the interest) in the algorithms themselves nor in parallelism, just in the application.

This book is intended to serve all of them, and this is why I initially said that it tries to fulfill an ambitious goal. The reader will have to judge to which extent this goal is met in the contents provided in the different chapters. Most chapters contain a methodological first part dealing with the technique, in order to settle its expected behavior and the main lines that could lead to its parallelization. In a second part, chapters discuss how parallel models can be derived for the technique to become
more efficient and what are the implications for the resulting algorithms. Finally, some experimental analysis is included in each chapter in order to help understand the advantages and limits of each proposal from a practical point of view. In this way, researchers whose specialities are in either domain can profit from the contents of each chapter. This is the way in which the central part of the book, entitled *Parallel Metaheuristic Models* (Chapters 5 to 17) was conceived.

There are of course some exceptions to this general chapter structure to make the book more complete. I added four initial chapters introducing the two fields (Chapters 1 to 4) and four trailing chapters dealing with theory and applications (Chapters 18 to 21). The resulting structure has three building blocks that offer an opportunity to the reader to select the parts or chapters he/she is more interested in. The four initial chapters are targeted to a broad sector of readers that want to know in a short time what are the most important topics and issues in metaheuristics and in parallelism, both dealt together or separately. In the third part, also included is an invited chapter on theoretical issues for Parallel Genetic Algorithms (a widely used metaheuristic) and three more chapters dealing with applications of these algorithms. Since the spectrum of potential applications is daunting, I decided to to devote a chapter to complex applications in general to reach a large audience, plus two additional ones on interesting, influent, and funded research lines internationally, that is telecommunications and bioinformatics.

The whole work is targeted to a wide set of readers, ranging from specialists in parallelism, optimization, application-driven research, and even graduate courses or beginners with some curiosity of the advances and latest techniques in parallel metaheuristics. Since it is an edited volume, I was able to profit from well-known international researchers as well as from new research lines on related topics started recently; this is an important added value that a non edited book could not show.

I would like to end this introduction with my profound acknowledgment to all authors contributing a chapter to this book, since any merit this work could deserve must be credited to them. Also, I thank the research group in my university in Málaga for all their effort and help in this project. I also appreciate the support received from Wiley during the whole editing process as well as the decisive endorsement of Professor A. Zomaya to make true this idea. To all of them, thank you very much.

*My final words are of course for my family: my wife, Ana, and my children, Enrique and Ana, the three lights that are always guiding my life, anytime, anywhere.*

ENRIQUE ALBA

Málaga, Spain
May 2005
Contributors

E. ALBA University of Málaga, Spain
Departamento de Lenguajes y Ciencias de la
Computación
E.T.S.I. Informática (3-2-12)
Campus de Teatinos, 29071 Málaga (Spain)
eat@lcc.uma.es

M. E. AYDIN London South Bank University, UK
London South Bank University, BCIM
103 Borough Rd., London, SE1 0AA (UK)
aydinme@lсу.ac.uk

C. BLUM Polytechnic University of Catalunya, Spain
Dept. de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
Jordi Girona 1-3, C6119 Campus Nord
E-08034 Barcelona (Spain)
cblum@lsi.upc.es

H. CANCELFA University of La República, Uruguay
Facultad de Ingeniería
J. Herrera y Reissig 565,
Montevideo 11300 (Uruguay)
cancela@fing.edu.uy

E. CANTÚ-PAZ Lawrence Livermore National Laboratory, USA
Center for Applied Scientific Computing
Lawrence Livermore National Laboratory
7000 East Avenue, L-561
Livermore, CA 94550 (USA)
cantupaz@llnl.gov

F. CHICANO University of Málaga, Spain
Departamento de Lenguajes y Ciencias de la
Computación
E.T.S.I. Informática (3-3-4)
Campus de Teatinos, 29071 Málaga (Spain)
chicano@lcc.uma.es

C. COTTA University of Málaga, Spain
Departamento de Lenguajes y Ciencias de la
Computación
E.T.S.I. Informática (3-2-49)
Campus de Teatinos, 29071 Málaga (Spain)
ccottap@lcc.uma.es

T. CRAINIC Transport Research Center and University of Québec at Montreal, Canada
Département Management et Technologie
Université du Québec à Montréal
315, rue Sainte-Catherine est, local R-2380
Montréal QC H2X 3X2 (Canada)
theo@crt.umontreal.ca

B. DORRONSORO University of Málaga, Spain
Departamento de Lenguajes y Ciencias de la
Computación
E.T.S.I. Informática (3-3-4)
Campus de Teatinos, 29071 Málaga (Spain)
dorronsoro@uma.es

F. FERNÁNDEZ University of Extremadura, Spain
Centro Universitario de Mérida
Universidad de Extremadura
C/ Sta. Teresa de Jornet, 38
06800 Mérida (Spain)
fcfdez@unex.es

F. GARCÍA University of La Laguna, Spain
Departamento de Estadística, I.O. y Computación
Universidad de La Laguna
38271 La Laguna (Spain)
fegarcia@ull.es

M. GARCÍA University of La Laguna, Spain
Departamento de Estadística, I.O. y Computación
Universidad de La Laguna
38271 La Laguna (Spain)
mgarcia@ull.es
M. GENDREAU  Transport Research Center and University of Montreal, Canada
Centre de Recherche sur les Transports
Université de Montréal
C.P. 6128, succ. Centre-ville Montréal-Québec
(Canada)
michelg@crt.umontreal.ca

N. HAIL  Transport Research Center and University of Montreal, Canada
Centre de Recherche sur les Transports
Université de Montréal
C.P. 6128, succ. Centre-ville Montréal-Québec
(Canada)
hail@crt.umontreal.ca

P. HANSEN  GERAD and HEC Montreal, Canada
3000, ch. de la Cote-Sainte Catherine
Montreal (Quebec) H3T 2A7 (Canada)
Pierre.Hansen@gerad.ca

S. JANSON  University of Leipzig, Germany
Parallel Computing and Complex Systems Group
Faculty of Mathematics and Computer Science
University of Leipzig
Augustusplatz 10/11 D-04109 Leipzig (Germany)
janson@informatik.uni-leipzig.de

F. LUNA  University of Málaga, Spain
Departamento de Lenguajes y Ciencias de la
Computación
E.T.S.I. Informática (3-3-4)
Campus de Teatinos, 29071 Málaga (Spain)
flv@lcc.uma.es

G. LUQUE  University of Málaga, Spain
Departamento de Lenguajes y Ciencias de la
Computación
E.T.S.I. Informática (3-3-4)
Campus de Teatinos, 29071 Málaga (Spain)
gabriel@lcc.uma.es

J. MADERA  University of Camagüey, Cuba
Department of Computing
Circunvalación Norte km. 5 1/2
Camagüey (Cuba)
jmadera@inf.reduc.edu.cu

N. MELAB  University of Lille, France
Laboratoire d'Informatique Fondamentale de Lille
UMR CNRS 8022, Cité scientifique
59655 Villeneuve d'Ascq cedex (France)
melab@lif.fr

B. MELIÁN  University of La Laguna, Spain
Departamento de Estadística, I.O. y Computación
Universidad de La Laguna
38271 La Laguna (Spain)
mmbelian@ull.es

D. MERKLE  University of Leipzig, Germany
Parallel Computing and Complex Systems Group
Faculty of Mathematics and Computer Science
University of Leipzig
Augustusplatz 10/11 D-04109 Leipzig (Germany)
merkle@informatik.uni-leipzig.de

M. MIDDENDORF  University of Leipzig, Germany
Parallel Computing and Complex Systems Group
Faculty of Mathematics and Computer Science
University of Leipzig
Augustusplatz 10/11 D-04109 Leipzig (Germany)
middendorf@informatik.uni-leipzig.de

N. MLADENOVIC  Mathematical Institute (SANU), Belgrade
Mathematical Institute, SANU
30000 Belgrade
Serbia and Montenegro
nenad@mi.sanu.ac.yu

J. A. MORENO-PÉREZ  University of La Laguna, Spain
Departamento de Estadística, I.O. y Computación
Universidad de La Laguna
38271 La Laguna (Spain)
jamoreno@ull.es

J. M. MORENO-VEGA  University of La Laguna, Spain
Departamento de Estadística, I.O. y Computación
Universidad de La Laguna
38271 La Laguna (Spain)
jmmoreno@ull.es

A. J. NEBRO  University of Málaga, Spain
Departamento de Lenguajes y Ciencias de la
Computación
E.T.S.I. Informática (3-2-15)
Campus de Teatinos, 29071 Málaga (Spain)
antonio@lcc.uma.es

S. NESMACHNOW  University of La República, Uruguay
Facultad e Ingeniería
J. Herrera y Reissig 565,
Montevideo 11300 (Uruguay)
sergion@fing.edu.uy

A. OCHOA  ICIMAF, Cuba
Institute of Cybernetics, Mathematics and Physics
Calle 15 No. 551 e/ C y D
10400 La Habana (Cuba)
ochoa@icmf.infn.cu
This Page Intentionally Left Blank
Part I

Introduction to Metaheuristics and Parallelism
1 An Introduction to Metaheuristic Techniques

CHRISTIAN BLUM$^{1}$, ANDREA ROLI$^{2}$, ENRIQUE ALBA$^{3}$

$^{1}$Universitat Politècnica de Catalunya, Spain
$^{2}$Università degli Studi "G.D'Annunzio", Italy
$^{3}$Universidad de Málaga, Spain

1.1 INTRODUCTION

In optimization we generally deal with finding among many alternatives a best (or good enough) solution to a given problem. Optimization problems occur everywhere in our daily life. Each one of us is constantly solving optimization problems, such as finding the shortest way from our home to our workplace subject to traffic constraints or organizing our agenda. (Most) human brains are pretty good in efficiently finding solutions to these daily problems. The reason is that they are still tractable, which means that their dimension is small enough to process them. However, these types of problems also arise in much bigger scales, such as, for example, making most beneficial use of the airplane fleet of an airline with the aim of saving fuel and parking costs. These kinds of problems are usually so high-dimensional and complex that computer algorithms are needed for tackling them.

Optimization problems can be modelled by means of a set of decision variables with their domains and constraints concerning the variable settings. They naturally divide into three categories: (i) the ones with exclusively discrete variables (i.e., the domain of each variable consists of a finite set of discrete values), (ii) the ones with exclusively continuous variables (i.e., continuous variable domains), and (iii) the ones with discrete as well as continuous variables. As metaheuristics were originally developed for optimization problems from class (i), we restrict ourselves in this introduction to this class of problems, which is also called the class of combinatorial optimization problems, or CO problems. However, much can be said and extended to continuous and other similar domains.

According to Papadimitriou and Steiglitz [114], a CO problem $\mathcal{P} = (S, f)$ is an optimization problem in which is given a finite set of objects $S$ and an objective function $f : S \rightarrow \mathbb{R}^+$ that assigns a positive cost value to each of the objects $s \in S$. 
The goal is to find an object of minimal cost value. The objects are typically integer numbers, subsets of a set of items, permutations of a set of items, or graph structures. An example is the well-known travelling salesman problem (TSP [92]).

Other examples of CO problems are assignment problems, timetabling, and scheduling problems. Due to the practical importance of CO problems, many algorithms to tackle them have been developed. These algorithms can be classified as either complete or approximate algorithms. Complete algorithms are guaranteed to find for every finite size instance of a CO problem an optimal solution in a bounded time (see [114, 111]). Yet, for CO problems that are \( \mathcal{NP} \)-hard [63], no polynomial time algorithm exists, assuming that \( \mathcal{P} \neq \mathcal{NP} \). Therefore, complete methods need exponential computation time in the worst-case. This often leads to computation times too high for practical purposes. Thus, the use of approximate methods to solve CO problems has received more and more attention in the last 30 years. In approximate methods we sacrifice the guarantee of finding optimal solutions for the sake of getting good solutions in a significantly reduced amount of time. Among the basic approximate methods we usually distinguish between constructive heuristics and local search methods.

1.1.1 Constructive Heuristics

Constructive heuristics are typically the fastest approximate methods. They generate solutions from scratch by adding opportunely defined solution components to an initially empty partial solution. This is done until a solution is complete or other stopping criteria are satisfied. For the sake of simplicity, we henceforth assume that a solution construction stops in case the current (partial) solution cannot be further extended. This happens when no completion exists such that the completed solution is feasible, i.e., it satisfies the problem constraints. In the context of constructive heuristics, solutions and partial solutions are sequences \( \langle c_1, \ldots, c_k \rangle \) composed of solution components \( c_j \) from a finite set of solution components \( \mathcal{C} \) (where \( |\mathcal{C}| = n \)). This kind of solution is throughout the chapter denoted by \( s \), respectively \( s^p \) in case of partial solutions.

Constructive heuristics have first to specify the set of possible extensions for each feasible (partial) solution \( s^p \). This set, henceforth denoted by \( \mathcal{N}(s^p) \), is a subset of \( \mathcal{C} \setminus \{ c | c \in s^p \} \). At each construction step one of the possible extensions is chosen until \( \mathcal{N}(s^p) = \emptyset \), which means either that \( s^p \) is a solution or that \( s^p \) is a partial solution that cannot be extended to a feasible solution. The algorithmic framework of a constructive heuristic is shown in Algorithm 1. A notable example of a constructive heuristic is a greedy heuristic, which implements procedure ChooseFrom(\( \mathcal{N}(s^p) \)) by applying a weighting function. A weighting function is a function that, sometimes

---

1 Note that minimizing over an objective function \( f \) is the same as maximizing over \(-f\). Therefore, every CO problem can be described as a minimization problem.

2 Note that constructive heuristics exist that may add several solution components at the same time to a partial solution. However, for the sake of simplicity, we restrict our description of constructive heuristics to the ones that add exactly one solution component at a time.
Algorithm 1 Constructive heuristic

\[ s^p = \langle \rangle \]

Determine \( \mathcal{N}(s^p) \)

while \( \mathcal{N}(s^p) \neq \emptyset \) do

\( c \leftarrow \text{ChooseFrom}(\mathcal{N}(s^p)) \)

\( s^p \leftarrow \text{extend } s^p \text{ by appending solution component } c \)

Determine \( \mathcal{N}(s^p) \)

end while

output: constructed solution

depending on the current (partial) solution, assigns at each construction step a heuristic value \( \eta(c) \) to each solution component \( c \in \mathcal{N}(s^p) \). Greedy heuristics choose at each step one of the extensions with the highest value.

For example, a greedy heuristic for the TSP is the Nearest Neighbor Heuristic. The set of solution components is the set of nodes (cities) in \( G = (V, E) \). The algorithm starts by selecting a city \( i \) at random. Then, the current partial solution \( s^p \) is extended at each of \( n - 1 \) construction steps by adding the closest city \( j \in \mathcal{N}(s^p) = V \setminus s^p \). Note that in the case of the Nearest Neighbor Heuristic the heuristic values, which are chosen as the inverse of the distances between the cities, do not depend on the current partial solution. Therefore, the weighting function that assigns the heuristic values is called \textit{static}. In cases in which the heuristic values depend on the current partial solution, the weighting function is called \textit{dynamic}.

### 1.1.2 Local Search Methods

As mentioned above, constructive heuristics are often very fast, yet they often return solutions of inferior quality when compared to local search algorithms. Local search algorithms start from some initial solution and iteratively try to replace the current solution by a better one in an appropriately defined neighborhood of the current solution, where the neighborhood is formally defined as follows:

**Definition 1** A neighborhood structure is a function \( \mathcal{N} : S \rightarrow 2^S \) that assigns to every \( s \in S \) a set of neighbors \( \mathcal{N}(s) \subseteq S \). \( \mathcal{N}(s) \) is called the neighborhood of \( s \). Often, neighborhood structures are implicitly defined by specifying the changes that must be applied to a solution \( s \) in order to generate all its neighbors. The application of such an operator that produces a neighbor \( s' \in \mathcal{N}(s) \) of a solution \( s \) is commonly called a move.

A neighborhood structure together with a problem instance defines the topology of a so-called search (or fitness) landscape \([134, 84, 61, 123]\). A search landscape can be visualized as a labelled graph in which the nodes are solutions (labels indicate their objective function value) and arcs represent the neighborhood relation between solutions. A solution \( s^* \in S \) is called a globally minimal solution (or global minimum) if for all \( s \in S \) it holds that \( f(s^*) \leq f(s) \). The set of all globally
AN INTRODUCTION TO METAHEURISTIC TECHNIQUES

Algorithm 2 Iterative improvement local search

\[
\begin{align*}
& s \leftarrow \text{GenerateInitialSolution()} \\
& \text{while } \exists s' \in \mathcal{N}(s) \text{ such that } f(s') < f(s) \text{ do} \\
& \quad s \leftarrow \text{ChooseImprovingNeighbor}(\mathcal{N}(s)) \\
& \text{end while} \\
& \text{output: } s
\end{align*}
\]

minimal solutions is henceforth denoted by \( S^* \). The introduction of a neighborhood structure enables us to additionally define the concept of \textit{locally} minimal solutions.

\textbf{Definition 2} A \textbf{locally minimal solution (or local minimum)} with respect to a neighborhood structure \( \mathcal{N} \) is a solution \( \hat{s} \) such that \( \forall s \in \mathcal{N}(\hat{s}) : f(\hat{s}) \leq f(s) \). We call \( \hat{s} \) a \textbf{strict locally minimal solution} if \( \forall s \in \mathcal{N}(\hat{s}) : f(\hat{s}) < f(s) \).

The most basic local search method is usually called \textit{iterative improvement local search}, since each move is only performed if the resulting solution is better than the current solution. The algorithm stops as soon as it finds a local minimum. The high level algorithm is sketched in Algorithm 2.

There are two major ways of implementing function \( \text{ChooseImprovingNeighbor}(\mathcal{N}(s)) \). The first way is called \textit{first-improvement}. A first-improvement function scans the neighborhood \( \mathcal{N}(s) \) and returns the first solution that is better than \( s \). In contrast, a \textit{best-improvement} function exhaustively explores the neighborhood and returns one of the solutions with the lowest objective function value. An iterative improvement procedure that uses a first-improvement function is called \textit{first-improvement local search}, respectively \textit{best-improvement local search} (or steepest descent local search) in the case of a best-improvement function. Both methods stop at local minima. Therefore, their performance strongly depends on the definition of a neighborhood structure \( \mathcal{N} \).

1.1.3 Metaheuristics

In the 1970s, a new kind of approximate algorithm has emerged which basically tries to combine basic heuristic methods in higher level frameworks aimed at efficiently and effectively exploring a search space. These methods are nowadays commonly called \textit{metaheuristics}. The term \textit{metaheuristic}, first introduced in [66], derives from the composition of two Greek words. \textit{Heuristic} derives from the verb \textit{heuriskein} (\textit{eýpírškein}) which means "to find", while the suffix \textit{meta} means "beyond, in an upper level". Before this term was widely adopted, metaheuristics were often called \textit{modern heuristics} [122]. The class of metaheuristic algorithms includes—\textit{but is not restricted to}—ant colony optimization (ACO), evolutionary computation (EC) including genetic algorithms (GAs), iterated local search (ILS), simulated annealing (SA), and tabu search (TS). For books and surveys on metaheuristics see [19, 69, 148].

\footnote{In alphabetical order.}
The different descriptions of metaheuristics found in the literature allow us to extract some fundamental properties by which metaheuristics are characterized:

- Metaheuristics are strategies that "guide" the search process.
- The goal is to efficiently explore the search space in order to find (near-) optimal solutions.
- Techniques which constitute metaheuristic algorithms range from simple local search procedures to complex learning processes.
- Metaheuristic algorithms are approximate and usually non-deterministic.
- They may incorporate mechanisms to avoid getting trapped in confined areas of the search space.
- The basic concepts of metaheuristics can be described on an abstract level (i.e., not tied to a specific problem)
- Metaheuristics are not problem-specific.
- Metaheuristics may make use of domain-specific knowledge in the form of heuristics that are controlled by the upper level strategy.
- Today's more advanced metaheuristics use search experience (embodied in some form of memory) to guide the search.

In short we may characterize metaheuristics as high level strategies for exploring search spaces by using different methods. Of great importance hereby is that a dynamic balance is given between diversification and intensification. The term diversification generally refers to the exploration of the search space, whereas the term intensification refers to the exploitation of the accumulated search experience. These terms stem from the tabu search field [70] and it is important to clarify that the terms exploration and exploitation are sometimes used instead, for example in the evolutionary computation field [51]. The balance between diversification and intensification is important, on one side to quickly identify regions in the search space with high quality solutions and on the other side not to waste too much time in regions of the search space which either are already explored or do not provide high quality solutions. Blum and Roli elaborated on the importance of the two concepts in their recent survey on metaheuristics [19].

The search strategies of different metaheuristics are highly dependent on the philosophy of the metaheuristic itself. There are several different philosophies apparent in the existing metaheuristics. Some of them can be seen as "intelligent" extensions of local search algorithms. The goal of this kind of metaheuristic is to escape from local minima in order to proceed in the exploration of the search space and to move on to find other hopefully better local minima. This is for example the case in tabu search, iterated local search, variable neighborhood search and simulated annealing. These metaheuristics (also called trajectory methods) work on one or several neighborhood structure(s) imposed on the search space. We can find a different philosophy in algorithms such as ant colony optimization and evolutionary computation. They incorporate a learning component in the sense that they implicitly or explicitly try to learn correlations between decision variables to identify high quality areas in the search space. This kind of metaheuristic performs, in a sense, a biased sampling
of the search space. For instance, in evolutionary computation this is achieved by recombination of solutions and in ant colony optimization by sampling the search space at each iteration according to a probability distribution.

There are different ways to classify and describe metaheuristic algorithms. Depending on the characteristics selected to differentiate among them, several classifications are possible, each of them being the result of a specific viewpoint (see for example, [136]). The classification into nature-inspired vs. non nature-inspired metaheuristics, into memory-based vs. memory-less methods, or into methods that either use a dynamic or a static objective function, is possible. In this chapter we describe the most important metaheuristics according to the single-point vs. population-based search classification, which divides metaheuristics into trajectory methods and population-based methods. This choice is motivated by the fact that this categorization permits a clearer description of the algorithms. Moreover, a successful hybridization is obtained by the integration of single-point search algorithms in population-based ones.

As mentioned at the beginning of this section, metaheuristic algorithms were originally developed for solving CO problems. However, in the meanwhile they are also successfully applied to continuous optimization problems. Examples are simulated annealing algorithms such as [128] or differential evolution [135] and [4, 25, 27] from the evolutionary computation field. Tabu search algorithms such as [13, 26] were among the first metaheuristic algorithms to be applied to continuous problems. Among the most recent metaheuristic approaches are ant colony optimization algorithms such as [46, 99, 131]. Some of the above mentioned algorithms are based on the well-known Nelder-Mead simplex algorithm for continuous optimization [110], while others are developed after new ideas on real parameter management coming from the mathematical programming field. However, for the rest of this introduction we will focus on metaheuristic approaches for CO problems, since including in each section discussion on real optimization could end in a chapter of quite difficult organization and reading.

The structure of this chapter is as follows. Section 1.2 and Section 1.3 are devoted to a description of nowadays most important metaheuristics. Section 1.2 describes the most relevant trajectory methods and in Section 1.3 we outline population-based methods. In Section 1.4 we give an overview over the different decentralized methods, which are metaheuristics without a central control, and we conclude in Section 1.5 with an overview on metaheuristic hybridizations.

1.2 TRAJECTORY METHODS

In this section we outline metaheuristics referred to as trajectory methods. The term trajectory methods is used because the search process performed by these methods is characterized by a trajectory in the search space. Most of these methods are extensions
of simple iterative improvement procedures (see Section 1.1.2), whose performance is usually quite unsatisfactory. They incorporate techniques that enable the algorithm to escape from local minima. This implies the necessity of termination criteria other than simply reaching a local minimum. Commonly used termination criteria are a maximum CPU time, a maximum number of iterations, a solution $s$ of sufficient quality, or reaching the maximum number of iterations without improvement.

### 1.2.1 Simulated Annealing

Simulated Annealing (SA) is commonly said to be the oldest among the metaheuristics and surely one of the first algorithms that had an explicit strategy to escape from local minima. The origins of the algorithm are in statistical mechanics (see the Metropolis algorithm [101]). The idea of SA was provided by the annealing process of metal and glass, which assume a low energy configuration when cooled with an appropriate cooling schedule. SA was first presented as a search algorithm for CO problems in [87] and [23]. In order to avoid getting trapped in local minima, the fundamental idea is to allow moves to solutions with objective function values that are worse than the objective function value of the current solution. Such a move is often called an uphill move. At each iteration a solution $s' \in \mathcal{N}(s)$ is randomly chosen. If $s'$ is better than $s$ (i.e., has a lower objective function value), then $s'$ is accepted as new current solution. Otherwise, $s'$ is accepted with a probability which is a function of a temperature parameter $T_k$ and $f(s') - f(s)$. Usually this probability is computed following the Boltzmann distribution:

$$p(s' | T_k, s) = e^{-\frac{f(s') - f(s)}{T_k}}. \quad (1.1)$$

The dynamic process described by SA is a Markov chain [52], as it follows a trajectory in the state space in which the successor state is chosen depending only on the incumbent one. This means that basic SA is memory-less. However, the use of memory can be beneficial for SA approaches (see for example [24]). The algorithmic framework of SA is described in Algorithm 3. The components are explained in more detail in the following.

**GeneratenInitialSolution():** The algorithm starts by generating an initial solution that may be randomly or heuristically constructed.

**SetlnitialTemperature():** The initial temperature is chosen such that the probability for an uphill move is quite high at the start of the algorithm.

**AdaptTemperature($T_k$):** The temperature $T_k$ is adapted at each iteration according to a cooling schedule (or cooling scheme). The cooling schedule defines the value of $T_k$ at each iteration $k$. The choice of an appropriate cooling schedule is crucial for the performance of the algorithm. At the beginning of the search the probability of accepting uphill moves should be high. Then, this probability should be gradually
Algorithm 3 Simulated Annealing (SA)

\[ s \leftarrow \text{GenerateInitialSolution()} \]
\[ k \leftarrow 0 \]
\[ T_k \leftarrow \text{SetInitialTemperature()} \]

while termination conditions not met do

\[ s' \leftarrow \text{PickNeighborAtRandom}(\mathcal{N}(s)) \]

if \( f(s') < f(s) \) then

\[ s \leftarrow s' \]
\[ \{ s' \text{ replaces } s \} \]

else

Accept \( s' \) as new solution with probability \( p(s' \mid T_k, s) \) \{Eq. (1.1)\}

end if

AdaptTemperature\((T_k)\)

\[ k \leftarrow k + 1 \]

end while

output: best solution found

decreased during the search. Note that this is not necessarily done in a monotonic fashion.

Theoretical results on non-homogeneous Markov chains \([1]\) state that under particular conditions on the cooling schedule, the algorithm converges in probability to a global minimum for \( k \to \infty \). More precisely:

\[ \exists r \in \mathbb{R}^+ \quad \text{s.t.} \quad \lim_{k \to \infty} p(\text{global minimum found after } k \text{ steps}) = 1 \]

\[ \iff \sum_{k=1}^{\infty} e^{-\frac{r}{T_k}} = \infty. \]

A particular cooling schedule that fulfills the hypothesis for the convergence is the one that follows a logarithmic law. Hereby, \( T_k \) is determined as \( T_k \leftarrow \frac{r}{\log(k+c)} \) (where \( c \) is a constant). Unfortunately, cooling schedules which guarantee the convergence to a global optimum are not feasible in applications, because they are too slow for practical purposes. Therefore, faster cooling schedules are adopted in applications. One of the most popular ones follows a geometric law: \( T_k \leftarrow \alpha \cdot T_{k-1} \), where \( \alpha \in (0, 1) \), which corresponds to an exponential decay of the temperature.

The cooling schedule can be used for balancing between diversification and intensification. For example, at the beginning of the search, \( T_k \) might be constant or linearly decreasing in order to sample the search space; then, \( T_k \) might follow a rule such as the geometric one in order to make the algorithm converge to a local minimum at the end of the search. More successful variants are non-monotonic cooling schedules (e.g., see \([94, 113]\)). Non-monotonic cooling schedules are characterized by alternating phases of cooling and reheating, thus providing an oscillating balance between diversification and intensification.

The cooling schedule and the initial temperature should be adapted to the particular problem instance considered, since the cost of escaping from local minima depends