



ESSAYS AND SURVEYS IN METAHEURISTICS

Volume Contributors:

E. Aarts
B. Baker
M.P. Bastos
S. Binato
J. Błażewicz
P.C. Borges
P. Boucher
J. Brandão
H. Bravo
A. Candia
A. Carbonaro
C. Carreto
C.C.B. Cavalcante
V.F. Cavalcante
L. Cavique
I. Charon
Y.-J. Cho
V.-D. Cung
G. Desaulniers
J. Desrosiers
P. Festa
M. Gendreau
T. Grünert
A.-F.-W. Han
M.P. Hansen
P. Hansen
W.J. Hery
H.H. Hoos
O. Hudry
T. Ibaraki
S.H. Jacobson
J. Korst
D.M. Loewenstern
N. Maculan
Th. Mahnig
V. Maniezzo
S.L. Martins
T. Mautor
N. Mladenović
F. Montenegro
H. Mühlenbein
K. Nonobe
G.C. Oliveira
G. Plateau
C. Rego
M.G.C. Resende
C.C. Ribeiro
C. Roucairol
A.M. Salvador
M.M. Solomon
C.C. de Souza
T. Stützle
E.D. Taillard
I. Themido
S. Voß
R. Walkowiak
M. Wright
T. Yamada

Edited by
CELSO C. RIBEIRO
PIERRE HANSEN

SPRINGER SCIENCE+BUSINESS MEDIA, LLC

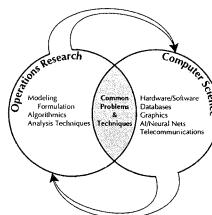
Essays and Surveys in Metaheuristics

OPERATIONS RESEARCH/COMPUTER SCIENCE INTERFACES SERIES

Series Editors

Professor Ramesh Sharda
Oklahoma State University

Prof. Dr. Stefan Voß
Technische Universität Braunschweig



Other published titles in the series:

Brown, Donald/Scherer, William T. / *Intelligent Scheduling Systems*

Nash, Stephen G./Sofer, Ariela / *The Impact of Emerging Technologies on Computer Science & Operations Research*

Barth, Peter / *Logic-Based 0-1 Constraint Programming*

Jones, Christopher V. / *Visualization and Optimization*

Barr, Richard S./ Helgason, Richard V./ Kennington, Jeffery L. / *Interfaces in Computer Science & Operations Research: Advances in Metaheuristics, Optimization, and Stochastic Modeling Technologies*

Ellacott, Stephen W./ Mason, John C./ Anderson, Iain J. / *Mathematics of Neural Networks: Models, Algorithms & Applications*

Woodruff, David L. / *Advances in Computational & Stochastic Optimization, Logic Programming, and Heuristic Search*

Klein, Robert / *Scheduling of Resource-Constrained Projects*

Bierwirth, Christian / *Adaptive Search and the Management of Logistics Systems*

Laguna, Manuel / González-Velarde, José Luis / *Computing Tools for Modeling, Optimization and Simulation*

Stilman, Boris / *Linguistic Geometry: From Search to Construction*

Sakawa, Masatoshi / *Genetic Algorithms and Fuzzy Multiobjective Optimization*

Essays and Surveys in Metaheuristics

Celso C. Ribeiro

Catholic University of Rio de Janeiro, Brazil

Pierre Hansen

École des Hautes Études Commerciales, Canada



Springer Science+Business Media, LLC

Library of Congress Cataloging-in-Publication Data

Essays and surveys in metaheuristics / [edited by] Celso C. Ribeiro, Pierre Hansen.

p. cm.-- (Operations research/computer science interfaces series ; ORCS 15)

Includes bibliographical references.

ISBN 978-1-4613-5588-5 ISBN 978-1-4615-1507-4 (eBook)

DOI 10.1007/978-1-4615-1507-4

1. Combinatorial optimization--Data processing. 2. Computer algorithms. I. Ribeiro, Celso C. II. Hansen, P. (Pierre) III. Series.

QA402.5 E88 2001

519.3--dc21

2001038701

Copyright © 2002 by Springer Science+Business Media New York

Originally published by Kluwer Academic Publishers in 2002

Softcover reprint of the hardcover 1st edition 2002

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, mechanical, photo-copying, recording, or otherwise, without the prior written permission of the publisher, Springer Science+Business Media, LLC.

Printed on acid-free paper.

Contents

Preface	ix
1 Selected Topics in Simulated Annealing <i>E. Aarts and J. Korst</i>	1
2 Reactive Tabu Search with Path-Relinking for the Steiner Problem in Graphs <i>M.P. Bastos and C.C. Ribeiro</i>	39
3 A GRASP for Job Shop Scheduling <i>S. Binato, W.J. Hery, D.M. Loewenstern, and M.G.C. Resende</i>	59
4 A Reactive GRASP for Transmission Network Expansion Planning <i>S. Binato and G.C. Oliveira</i>	81
5 Tabu Search for Two-Dimensional Irregular Cutting <i>J. Błażejewicz, A.M. Salvador, and R. Walkowiak</i>	101
6 A Study of Global Convexity for a Multiple Objective Travelling Sales- man Problem <i>P.C. Borges and M.P. Hansen</i>	129
7 A Lower Bound Based Meta-Heuristic for the Vehicle Routing Problem <i>J. Brandão</i>	151
8 A Simulated Annealing Approach for Minimum Cost Isolated Failure Immune Networks <i>A. Candia and H. Bravo</i>	169

9

9	A GRASP Interactive Approach to the Vehicle Routing Problem with Backhauls <i>C. Carreto and B. Baker</i>	185
10	Parallel Cooperative Approaches for the Labor Constrained Scheduling Problem <i>C.C.B. Cavalcante, V.F. Cavalcante, C.C. Ribeiro, and C.C. de Souza</i>	201
11	A Scatter Search Algorithm for the Maximum Clique Problem <i>L. Cavique, C. Rego, and I. Themido</i>	227
12	The Noising Methods: A Survey <i>I. Charon and O. Hudry</i>	245
13	Strategies for the Parallel Implementation of Metaheuristics <i>V.-D. Cung, S.L. Martins, C.C. Ribeiro, and C. Roucairol</i>	263
14	Accelerating Strategies in Column Generation Methods for Vehicle Routing and Crew Scheduling Problems <i>G. Desaulniers, J. Desrosiers, and M.M. Solomon</i>	309
15	GRASP: An Annotated Bibliography <i>P. Festa and M.G.C. Resende</i>	325
16	Recent Advances in Tabu Search <i>M. Gendreau</i>	369
17	Lagrangean Tabu Search <i>T. Grünert</i>	379
18	A GIDS Metaheuristic Approach to the Fleet Size and Mix Vehicle Routing Problem <i>A.F.-W. Han and Y.-J. Cho</i>	399
19	Developments of Variable Neighborhood Search <i>P. Hansen and N. Mladenović</i>	415
20	Analyzing the Performance of Local Search Algorithms Using Generalized Hill Climbing Algorithms <i>S.H. Jacobson</i>	441

21		
Ant Colony Optimization: An Overview <i>V. Maniezzo and A. Carbonaro</i>		469
22		
Intensification Neighborhoods for Local Search Methods <i>T. Mautour</i>		493
23		
New Heuristics for the Euclidean Steiner Problem in R^n <i>F. Montenegro, N. Maculan, G. Plateau, and P. Boucher</i>		509
24		
Mathematical Analysis of Evolutionary Algorithms <i>H. Mühlenbein and Th. Mahnig</i>		525
25		
Formulation and Tabu Search Algorithm for the Resource Constrained Project Scheduling Problem <i>K. Nonobe and T. Ibaraki</i>		557
26		
Analysing the Run-Time Behaviour of Iterated Local Search for the Travelling Salesman Problem <i>T. Stützle and H.H. Hoos</i>		589
27		
POPMUSIC — Partial Optimization Metaheuristic under Special Intensification Conditions <i>E.D. Taillard and S. Voss</i>		613
28		
Subcost-Guided Simulated Annealing <i>M. Wright</i>		631
29		
A Pruning Pattern List Approach to the Permutation Flowshop Scheduling Problem <i>T. Yamada</i>		641

Preface

Finding exact solutions to many combinatorial optimization problems in business, engineering, and science still poses a real challenge, despite the impact of recent advances in mathematical programming and computer technology. New fields of applications, such as computational biology, electronic commerce, and supply chain management, bring new challenges and needs for algorithms and optimization techniques. Metaheuristics are master procedures that guide and modify the operations of subordinate heuristics, to produce improved approximate solutions to hard optimization problems with respect to more simple algorithms. They also provide fast and robust tools, producing high-quality solutions in reasonable computation times.

The field of metaheuristics has been fast evolving in recent years. Techniques such as simulated annealing, tabu search, genetic algorithms, scatter search, greedy randomized adaptive search, variable neighborhood search, ant systems, and their hybrids are currently among the most efficient and robust optimization strategies to find high-quality solutions to many real-life optimization problems. A very large number of successful applications of metaheuristics are reported in the literature and spread throughout many books, journals, and conference proceedings. A series of international conferences entirely devoted to the theory, applications, and computational developments in metaheuristics has been attracting an increasing number of participants, from universities and the industry.

Kluwer published two books ^{1,2} which grew out of the two first *Metaheuristics International Conferences* (MIC'95 held in Breckenridge, United States, in 1995; MIC'97 held in Sophia-Antipolis, France, in 1997). Similarly, this new book grows out of the *Third Metaheuristics International Conference* held in Angra dos Reis, Brazil, in July 1999, chaired by Celso C. Ribeiro (Catholic

¹S. Voss, S. Martello, I.H. Osman, and C.Roucairol (editors), *Meta-heuristics: Advances and Trends in Local Search Paradigms for Optimization*, Kluwer Academic Publishers, 1999.

²I.H. Osman and J.P. Kelly (editors), *Meta-heuristics: Theory and Applications*, Kluwer Academic Publishers, 1996.

University of Rio de Janeiro, Brazil) and Pierre Hansen (École des Hautes Études Commerciales, Montréal, Canada) and attended by approximately 150 participants.

Essays and Surveys in Metaheuristics is not only a conference proceedings. This book innovates with respect to the two previous ones, in containing a series of surveys on recent developments of the main metaheuristics. Well-known specialists have written surveys on the following subjects: simulated annealing (E. Aarts and J. Korst, The Netherlands), noising methods (I. Charon and O. Hudry, France), parallel implementation of metaheuristics (V.-D. Cung and C. Roucairol, France, and S.L. Martins and C.C. Ribeiro, Brazil), greedy randomized adaptive search procedures (P. Festa, Italy, and M.G.C. Resende, USA), tabu search (M. Gendreau, Canada), variable neighborhood search (P. Hansen and N. Mladenovic, Canada), ant colonies (V. Maniezzo and A. Carbonaro, Italy), and evolutionary algorithms (H. Mühlenbein and Th. Mahnig, Germany).

Several further essays address issues or variants of metaheuristics: use of metaheuristics within exact algorithms (G. Desaulniers and J. Desrosiers, Canada, and M.M. Solomon, USA), performance of metaheuristics (S.H. Jacobson, USA), intensification neighborhoods (T. Mautor, France), Lagrangean tabu search (T. Grünert, Germany), partial optimization (E. Taillard, Switzerland, and S. Voss, Germany), and subcost guided simulated annealing (M. Wright, UK).

Finally, a large part of the book is devoted to innovative or successful applications of metaheuristics to classical or new combinatorial optimization problems: job-shop, flow-shop, and resource-constrained scheduling (S. Binato, Brazil, W.J. Hery, D.M. Loewenstern, and M.G.C. Resende, USA; T. Yamada, Japan; C.C.B. Cavalcante, V.C. Cavalcante, C.C. Ribeiro, and C.C. de Souza, Brazil; K. Nonobe and T. Ibaraki, Japan), the Steiner problem in graphs (M.P. Bastos and C.C. Ribeiro, Brazil; F. Montenegro and N. Maculan, Brazil, G. Plateau and P. Boucher, France), vehicle routing (J. Brandão, Portugal; C. Carreto, Portugal, and B. Baker, England; A.F.-W. Han and Y.-J. Cho, Taiwan), travelling salesman (P.C. Borges, Portugal, and M.P. Hansen, Danemark; T. Stützle and H.H. Hoos, Germany), network reliability and design (A. Candia and H. Bravo, Chile; S. Binato and G.C. Oliveira, Brazil), 2-dimensional cutting (J. Blazewicz, A.M. Salvador, and R. Walkowiak, Poland), and maximum clique (L. Cavique, C. Rego, and I. Themido, Portugal).

As the organizers of MIC'99, we wish to acknowledge FAPERJ - *Fundação Carlos Chagas Filho de Amparo à Pesquisa do Estado do Rio de Janeiro*, Brazil, for the generous financial support for the conference, which strongly contributed to the participation of many researchers and graduate students. We are also thankful to the staff of the Department of Computer Science of the Catholic University of Rio de Janeiro, for their administrative support, and to Simone Martins and Geraldo Veiga, for their help with the editorial work involving the pre-conference volume of extended abstracts, from which this book emerged. As the editors of *Essays and Surveys in Metaheuristics*, we are very grateful

to Carole Dufour, for her secretarial assistance along the refereeing procedure, and to Nicole Paradis, for her patience and splendidous editorial work, both of them helped by Francine Benoît. We are thankful to GERAD - *Groupe d'études et de recherche en analyse des décisions* and AT&T Labs Research, for the good environment and appropriate conditions offered for the conclusion of the editorial work of this book. Finally, we are also grateful to Gary Folven, for his encouragements and support at different stages of the production of this book, and to the anonymous and generous work of almost one hundred devoted referees.

Holmdel and Montréal, April 2001

*Celso C. Ribeiro
Pierre Hansen*

1 SELECTED TOPICS IN SIMULATED ANNEALING

Emile Aarts^{1,2} and Jan Korst¹

¹Philips Research Laboratories

Prof. Holstlaan 4, 5656 AA Eindhoven, The Netherlands

emile.aarts@philips.com

²Eindhoven University of Technology

P.O. Box 513, 5600 MB Eindhoven, The Netherlands

jan.korst@philips.com

Abstract: We review a number of selected topics that were published in the simulated annealing literature during the past decade. The emphasis of the presentation is on theoretical and general results. The presentation of the novel features include generalized convergence results, new performance properties, improved variants, genetic hybrids, and approaches to general mathematical programming models.

1.1 INTRODUCTION

Simulated annealing belongs to a class of randomized local search algorithms that can be used to handle hard combinatorial optimization problems. The fundamentals of the simulated annealing approach were introduced in the 1980s by Kirkpatrick et al. [66] and Černý [25] following an analogy with the physical annealing process which was used to find low-energy states of solids. During the first decade of its existence the approach has been extensively studied both from a theoretical as well as from an application point of view, which has led to a wealth of interesting results on the convergence of the algorithm and on its performance when applied to a large variety of problems. A remarkable fact is also the wide-spread use of the algorithm in fields like molecular physics, biology, and chemistry, where the approach has been accepted in the meantime as a general tool.

In this chapter we review the developments in simulated annealing over the past decade. A variety of new and interesting results have been published recently, and we have made a selection for this book chapter that contains primarily theoretical and general results. Consequently, we will pay only little attention to the large stack of applied results that has recently been published. Our bibliometric search yielded more than a thousand applied papers on simulated annealing that were published over the past decade dealing with a very large range of subjects. A review of these results would call for an encyclopedic approach or an annotated bibliography, which is beyond the scope of this book chapter. We restrict ourselves to the review of a number of issues from practice that have a more or less general nature.

The selected topics presented in this book chapter are organized along the following lines. First, we briefly review *basic simulated annealing*, and its *asymptotic convergence*, where basic refers to the use of the classical Metropolis acceptance criterion. We also review results on *generalized simulated annealing*, where the generalization refers to the use of more general acceptance criteria and the corresponding framework for the theoretical analysis. Next, we review results on *performance analysis* and *cooling schedules*. *Issues from practice* are only briefly reviewed where we restrict ourselves to results that have a general nature. Many researchers have investigated variants of simulated annealing that are aimed at improved performance, especially with respect to time efficiency. *Variants of simulated annealing* include deterministic annealing, threshold accepting, rescaled annealing, and simulated jumping. Simulated annealing has primarily been applied to combinatorial optimization problems. However, during the past decade simulated annealing has also been applied to several other types of optimization problems including *global optimization*, *multi-criteria optimization*, and *discrete stochastic optimization*. Of these three, we review global optimization in more detail. In addition, we consider how simulated annealing can be used to govern self-organization in *neural networks*. Finally, we briefly mention some results on *choosing neighborhoods*, on *parallel annealing*, on *combined approaches*, with an emphasis on the combination with genetic algorithms, and on *quantum annealing*.

1.2 BASIC SIMULATED ANNEALING

The use of simulated annealing presupposes the definition of a combinatorial optimization problem and a neighborhood. A *combinatorial optimization problem* is a set of problem instances where each *instance* is a pair (\mathcal{S}, f) with \mathcal{S} the set of feasible solutions and $f : \mathcal{S} \rightarrow \mathbb{Z}$ a cost function that assigns a cost value to each solution. The problem is to find a *globally optimal solution*, i.e., an $i^* \in \mathcal{S}$ such that $f(i^*) \leq f(i)$, for all $i \in \mathcal{S}$. Furthermore, $f^* = f(i^*)$ denotes the optimal cost value, and $\mathcal{S}^* = \{i \in \mathcal{S} \mid f(i) = f^*\}$ denotes the set of optimal solutions. The solution set is often represented by a set of *decision variables*, whose values can have certain ranges, and a solution is then represented by a value assignment of the variables. A *neighborhood function* is a mapping $\mathcal{N} : \mathcal{S} \rightarrow 2^{\mathcal{S}}$, which defines for each solution $i \in \mathcal{S}$ a set $\mathcal{N}(i) \subseteq \mathcal{S}$ of solutions

that are in some sense close to i . The set $\mathcal{N}(i)$ is called the *neighborhood* of solution i , and each $j \in \mathcal{N}(i)$ is called a *neighbor* of i . We shall assume that $i \in \mathcal{N}(i)$ for all $i \in \mathcal{S}$. A solution $\hat{i} \in \mathcal{S}$ is *locally optimal (minimal)* with respect to \mathcal{N} if

$$f(\hat{i}) \leq f(j) \quad \text{for all } j \in \mathcal{N}(\hat{i}).$$

The set of locally optimal solutions is denoted by $\hat{\mathcal{S}}$.

Roughly speaking, simulated annealing starts off with an initial solution in \mathcal{S} and then continually tries to find better solutions by searching neighborhoods and applying a stochastic acceptance criterion. This is schematically laid out in Figure 1.1.

```

procedure SIMULATED_ANNEALING;
begin
    INITIALIZE ( $i_{start}$ );
     $i := i_{start}$ ;
     $k := 0$ ;
    repeat
        GENERATE ( $j$  from  $\mathcal{N}(i)$ );
        if  $f(j) - f(i) < t_k$  then  $i := j$ ;
         $k := k + 1$ ;
    until STOP;
end;

```

Figure 1.1 Pseudocode of the basic simulated annealing algorithm.

The procedure INITIALIZE selects a start solution from \mathcal{S} , the procedure GENERATE selects a solution from the neighborhood of the current solution, and the procedure STOP evaluates a stop criterion that determines termination of the algorithm. Simulated annealing algorithms continually select a neighbor of a current solution and compare the difference in cost between these solutions to a threshold. If the cost difference is within the threshold, the neighbor replaces the current solution. Otherwise, the search continues with the current solution. The sequence $(t_k | k = 0, 1, 2, \dots)$ denotes the thresholds, where t_k is used at iteration k of the algorithm, and is given by a random variable with expected value $\mathbb{E}(t_k) = c_k \in \mathbb{R}^+$, $k = 0, 1, 2, \dots$. The t_k 's follow a probability distribution function F_{c_k} over \mathbb{R}^+ . Simulated annealing uses randomized thresholds with values between zero and infinity, and the probability of a threshold t_k being at most $y \in \mathbb{R}^+$ is given by $\mathbb{P}_{c_k}\{t_k \leq y\} = F_{c_k}(y)$. This implies that each neighboring solution can be chosen with a finite probability to replace the current solution.

The basic simulated annealing version of Kirkpatrick et al. [66] and Černý [25] take for F_{c_k} the negative exponential distribution with parameter $1/c_k$. This choice is identical to the following *acceptance criterion*. For any two solutions $i, j \in \mathcal{S}$ the probability of accepting j from i at the k th iteration is given by

$$\mathbb{P}_{c_k} \{\text{accept } j\} = \begin{cases} 1 & \text{if } f(j) \leq f(i) \\ \exp\left(\frac{f(i) - f(j)}{c_k}\right) & \text{if } f(j) > f(i). \end{cases} \quad (1.1)$$

The parameter c_k is used in the simulated annealing algorithm as a *control parameter*, and it plays an important role in the convergence of the algorithm. A characteristic feature of simulated annealing is that, besides accepting improvements in cost, it also accepts to a limited extent deteriorations in cost. Initially, at large values of c , large deteriorations are accepted; as c decreases, only smaller deteriorations are accepted, and finally, as the value of c approaches 0, no deteriorations are accepted at all. Arbitrarily large deteriorations are accepted with positive probability; for these deteriorations the acceptance probability is however small.

The origin of simulated annealing and the choice of the acceptance criterion can be found in the physical annealing process. In *condensed matter physics*, *annealing* is a thermal process for obtaining low energy states of a solid in a *heat bath*. It consists of the following two steps: first, the temperature of the heat bath is increased to a maximum value at which the solid melts; second, the temperature is carefully decreased until the particles of the melted solid arrange themselves in the ground state of the solid. In the liquid phase all particles of the solid arrange themselves randomly. In the ground state the particles are arranged in a highly structured lattice and the energy of the system is minimal.

The physical annealing process can be modeled successfully by computer simulation methods based on *Monte Carlo techniques*; see Binder [19]. Here, we discuss one of the early techniques proposed by Metropolis et al. [86], who gave a simple algorithm for simulating the evolution of a solid in a heat bath to *thermal equilibrium*. Their algorithm is based on Monte Carlo techniques, and generates a sequence of states of the solid in the following way. Given a current state i of the solid with *energy* E_i , a subsequent state j is generated by applying a perturbation mechanism, which transforms the current state into a next state by a small distortion, for instance by displacement of a single particle. The energy of the next state is E_j . If the *energy difference*, $E_j - E_i$, is less than or equal to 0, the state j is accepted as the current state. If the energy difference is greater than 0, the state j is accepted with a probability that is given by

$$\exp\left(\frac{E_i - E_j}{k_B T}\right),$$

where T denotes the *temperature* of the heat bath and k_B is a physical constant known as the *Boltzmann constant*. The acceptance rule described above is known as the *Metropolis criterion*, the corresponding algorithm as the *Metropolis algorithm*.

If the temperature is lowered sufficiently slowly, the solid can reach thermal equilibrium at each temperature. Thermal equilibrium is characterized by the *Boltzmann distribution*, which relates the probability of the solid of being in a state i with energy E_i to the temperature T , and is given by

$$\mathbb{P}_T\{\mathbf{X} = i\} = \frac{\exp(-E_i/k_B T)}{\sum_j \exp(-E_j/k_B T)},$$

where \mathbf{X} is a random variable denoting the current state of the solid and the summation extends over all possible states. As we show in the following sections, the Boltzmann distribution plays an essential role in the analysis of the simulated annealing algorithm.

The analogy between the annealing of a physical many-particle system and the application of simulated annealing to a combinatorial optimization problem is obvious from the following equivalences.

- Solutions in a combinatorial optimization problem are equivalent to states of the physical system.
- The cost of a solution is equivalent to the energy of a state.
- Transitions to neighbors are equivalent to state changes.
- The control parameter is equivalent to the temperature.

Since the introduction of simulated annealing the physics analogy has played an important role in both the analysis of the algorithm and the development of novel algorithmic concepts which we will touch upon below.

1.3 ASYMPTOTIC CONVERGENCE

A simulated annealing algorithm can be viewed as a sampling process whose outcomes are neighboring solutions. This class of processes can be mathematically modeled using finite Markov chains; see e.g. Feller [39], Isaacson and Madsen [59], and Seneta [104].

Let \mathcal{O} denote a set of possible outcomes of a sampling process. A *Markov chain* is a sequence of *trials*, satisfying the *Markov property* which states that the probability of the outcome of a given trial depends only on the outcome of the previous trial. Let $\mathbf{X}(k)$ be a random variable denoting the outcome of the k th trial. Then the *transition probability* at the k th trial for each pair of outcomes $i, j \in \mathcal{O}$ is defined as

$$P_{ij}(k) = \mathbb{P}\{\mathbf{X}(k) = j | \mathbf{X}(k-1) = i\}. \quad (1.2)$$

The matrix $P(k)$, whose elements are given by (1.2), is the *transition matrix*, and it is called *inhomogeneous* if the transition probabilities depend on the trial number k , and *homogeneous* if they do not depend on the trial number.

In the case of simulated annealing, a trial corresponds to a transition, and the set of outcomes is given by the finite set of solutions. Furthermore, the outcome

of a trial only depends on the outcome of the previous trial, which implies the Markov property. Consequently, we obtain for simulated annealing the following formulation. Let (\mathcal{S}, f) be an instance of a combinatorial optimization problem and \mathcal{N} a neighborhood function. Then the *transition probabilities* for the simulated annealing algorithm are defined as

$$\forall i, j \in \mathcal{S} : P_{ij}(k) = \begin{cases} G_{ij}(c_k) A_{ij}(c_k) & \text{if } j \neq i \\ 1 - \sum_{l \in \mathcal{S}, l \neq i} G_{il}(c_k) A_{il}(c_k) & \text{if } j = i, \end{cases} \quad (1.3)$$

where $G_{ij}(c_k)$ denotes the *generation probability*, i.e., the probability of generating a solution j from a solution i , and $A_{ij}(c_k)$ denotes the *acceptance probability*, i.e., the probability of accepting a solution j that is generated from solution i . The $G_{ij}(c_k)$'s and $A_{ij}(c_k)$'s of (1.3) are conditional probabilities, i.e., $G_{ij}(c_k) = \mathbb{P}_c\{\text{generate } j \mid i\}$ and $A_{ij}(c_k) = \mathbb{P}_c\{\text{accept } j \mid i, j\}$. The corresponding matrices $G(c_k)$ and $A(c_k)$ are the *generation matrix* and *acceptance matrix*, respectively. They need not be stochastic.

Basic simulated annealing assumes the following probabilities:

- Generation probability

$$\forall i, j \in \mathcal{S} : G_{ij}(c_k) = G_{ij} = \begin{cases} \frac{1}{|\mathcal{N}(i)|} & \text{if } j \in \mathcal{N}(i) \\ 0 & \text{otherwise.} \end{cases} \quad (1.4)$$

- Acceptance probability

$$\forall i, j \in \mathcal{S} : A_{ij}(c_k) = \exp\left(-\frac{[f(j) - f(i)]^+}{c_k}\right), \quad (1.5)$$

where, for all $a \in \mathbb{R}$, $a^+ = \max(0, a)$.

Thus, the generation probabilities are chosen to be uniformly distributed over the neighborhoods $\mathcal{N}(i)$, and the acceptance probabilities follow the Metropolis criterion.

Stationarity is an essential property of Markov chains for the convergence of simulated annealing. Under mild conditions on the transition probabilities associated with the Markov chains, the probability distribution of the outcomes after an infinite number of trials satisfy specific stationarity properties reflected by the stationary distribution which is defined as follows. A *stationary distribution* of a finite homogeneous Markov chain with transition matrix P on a set of outcomes \mathcal{O} is defined as the stochastic $|\mathcal{O}|$ -vector q , whose components are given by

$$q_i = \lim_{k \rightarrow \infty} \mathbb{P}\{\mathbf{X}(k) = i\}, \quad \text{for all } i \in \mathcal{O},$$

which is independent of $\mathbf{X}(0)$. Thus, q is the left eigenvector of P with eigenvalue 1. Clearly, in the case of simulated annealing, as P depends on c , q depends on c , i.e., $q = q(c)$. We now can formulate one of the most important convergence results of basic simulated annealing, which was first published by

Lundy and Mees [84]; see also Aarts and Korst [1] and Van Laarhoven and Aarts [74].

Theorem 1. *Let (\mathcal{S}, f) be an instance of a combinatorial optimization problem, \mathcal{N} a neighborhood function, and $P(k)$ the transition matrix of the homogeneous Markov chain associated with basic simulated annealing defined by (1.3), (1.4), and (1.5), with $c_k = c$, for all k . Furthermore, let the following connectivity condition be satisfied*

$$\forall i, j \in \mathcal{S} \exists p \geq 1 \exists l_0, l_1, \dots, l_p \in \mathcal{S}$$

with $l_0 = i, l_p = j$, and $G_{l_k l_{k+1}} > 0$, $k = 0, 1, \dots, p - 1$. (1.6)

Then the associated homogeneous Markov chain has a stationary distribution $q(c)$, whose components are given by

$$q_i(c) = \frac{|\mathcal{N}(i)| \exp(-f(i)/c)}{\sum_{j \in \mathcal{S}} |\mathcal{N}(j)| \exp(-f(j)/c)} \quad \text{for all } i \in \mathcal{S}, \quad (1.7)$$

and

$$q_i^* \stackrel{\text{def}}{=} \lim_{c \downarrow 0} q_i(c) = \begin{cases} \frac{1}{|\mathcal{S}^*|} & \text{if } i \in \mathcal{S}^* \\ 0 & \text{otherwise,} \end{cases} \quad (1.8)$$

where \mathcal{S}^ denotes the set of optimal solutions.*

The distribution given by (1.7) is the equivalent of the Boltzmann distribution in the Monte Carlo simulations of the physical annealing process mentioned in Section 1.1.

As a result of Theorem 1 we have that

$$\lim_{c \downarrow 0} \lim_{k \rightarrow \infty} \mathbb{P}_c \{ \mathbf{X}(k) \in \mathcal{S}^* \} = 1, \quad (1.9)$$

which reflects the basic property of the simulated annealing algorithm, i.e., the guarantee that the algorithm asymptotically finds an optimal solution. Furthermore, (1.9) expresses the characteristic of the homogeneous model for basic simulated annealing, viz., first take the limit of the homogeneous Markov chain for an infinite number of trials, and next, take the limit for the control parameter to zero. In the inhomogeneous model, these two limits are combined to a single one.

Many authors have analyzed the convergence of basic simulated annealing in terms of inhomogeneous Markov chains, i.e., they assume a model in which the value of the control parameter may be changed after each transition. Necessary and sufficient conditions for asymptotic convergence in this case have been derived by Hajek [50]. To discuss this result we need the following definitions.

For any two solutions $i, j \in \mathcal{S}$, j is *reachable at height h* from i if there exists a sequence of solutions $i = l_0, l_1, \dots, l_p = j \in \mathcal{S}$ with $G_{l_k l_{k+1}} > 0$ for $k = 0, \dots, p - 1$ and $f(l_k) \leq h$ for all $k = 0, \dots, p$.

The *depth* $d(\hat{i})$ of a local optimum \hat{i} is the smallest positive number x , such that there is a solution $j \in \mathcal{S}$ with $f(j) < f(\hat{i})$ that is reachable at height $f(\hat{i}) + x$ from \hat{i} . By definition, for an optimal solution i^* , $d(i^*) = \infty$.

We now can formulate Hajek's result.

Theorem 2. *Let $(c_k | k = 0, 1, \dots)$ be a sequence of values of the control parameter defined as*

$$c_k = \frac{\Gamma}{\log(k+2)}, \quad k = 0, 1, \dots,$$

for some constant Γ . Then asymptotic convergence of the simulated annealing algorithm, using the transition probabilities of (1.3), (1.4) and (1.5), is guaranteed if and only if

- *the Markov chain is irreducible,*
- *i is reachable from j at height h if and only if j is reachable from i at height h , for arbitrary $i, j \in \mathcal{S}$ and h , and*
- *the constant Γ satisfies $\Gamma \geq D$, where*

$$D = \max_{i \in \hat{\mathcal{S}} \setminus \mathcal{S}^*} d(\hat{i}), \quad (1.10)$$

i.e., D is the depth of the deepest local, non-global minimum.

Kern [64] has addressed the problem of calculating the value of D . In particular, he showed for a number of problems that it is unlikely that D can be calculated in polynomial time for arbitrary instances. Kern also presents bounds on the value of D for several combinatorial optimization problems.

Finally, we mention that similar asymptotic convergence results have been obtained for discrete stochastic optimization problems, where the cost function is of a stochastic nature, for example caused by noise. First results date back to Gelfand and Mitter [44]. By assuming that in iteration k the noise is normally distributed with mean zero and variance σ_k^2 , Gelfand and Mitter impose conditions on σ_k^2 to ensure asymptotic convergence. For more recent results, we refer to Alkhamis et al. [5] and Alrefaei and Andradottir [6], and references therein.

1.4 GENERALIZED SIMULATED ANNEALING

Several authors have addressed the convergence of simulated annealing for more general forms of the generation and acceptance probabilities than the probabilities of (1.4) and (1.5) which are used in basic simulated annealing. Especially the use of more general forms of the acceptance probability have been extensively studied with the aim to find probabilities different from the exponential form of the Metropolis criterion that exhibit a similar asymptotic convergence behavior. First results in this respect were published by Lundy and Mees [84] who extended the results of Theorem 1 to a more general class of acceptance

probabilities. More recently this result was further refined by Schuur [102] who proved the following theorem.

Theorem 3. *Let $P(k)$ be the transition matrix of the homogeneous Markov chain associated with the simulated annealing algorithm defined by (1.3) with $c_k = c$, for all k and generation probabilities given by (1.4) and satisfying the connectivity condition of (1.6). Furthermore, let the acceptance probabilities be defined as*

$$A_{ij}(c) = H(c, f(i), f(j)) \cdot \min \left(1, \frac{\varphi(c, f(j))}{\varphi(c, f(i))} \right),$$

where $\varphi : (0, \infty) \times \mathbb{R} \rightarrow (0, \infty)$ and $H : (0, \infty) \times \mathbb{R} \times \mathbb{R} \rightarrow (0, 1]$ are two functions such that for $c > 0$, and $x, y \in \mathbb{R}$: $H(c, x, y) = H(c, y, x)$, and

$$\forall x, y \in \mathbb{R} : x > y \Rightarrow \lim_{c \downarrow 0} \frac{\varphi(c, y)}{\varphi(c, x)} = 0. \quad (1.11)$$

Then the Markov chain has a unique stationary distribution $q(c)$, whose components are given by

$$q_i(c) = \frac{\varphi(c, f(i))}{\sum_{j \in S} \varphi(c, f(j))} \text{ for all } i \in S, \quad (1.12)$$

and

$$\lim_{c \downarrow 0} q_i(c) = q_i^*,$$

where q_i^* is given by (1.8).

As a corollary to Theorem 3 it is argued that the only well-behaved function $\varphi(c, f(j))$ that satisfies (1.11) is of the form

$$\varphi(c, f(j)) = \exp(\gamma(c)f(j)), \quad (1.13)$$

where $\gamma : (0, \infty) \rightarrow (0, \infty)$ and $\lim_{c \downarrow 0} \gamma(c) = \infty$.

Kesidis and Wong [65] and Romeo and Sangiovanni-Vincentelli [101] provide arguments for the assertion that the fastest convergence to the stationary distribution of (1.12) with (1.13) and $\gamma = c^{-1}$ is given by the acceptance probabilities of (1.5).

Anily and Federgruen [13] extended the asymptotic convergence results of generalized annealing to include the inhomogeneous case by adding the following requirement

$$\sum_{k=0}^{\infty} (\underline{A}(c_k))^n = \infty, \quad (1.14)$$

where n denotes the maximum number of steps needed to reach an optimal solution from any arbitrary solution, and

$$\underline{A}(c) = \min_{i,j} \{ A_{ij}(c) \mid i \in S, j \in N(i) \}.$$

Then

$$\lim_{k \rightarrow \infty} \mathbb{P}\{\mathbf{X}(k) \in \mathcal{S}^*\} = 1.$$

Several authors have considered conditions for convergence of generalized simulated annealing to include the inhomogeneous case. Most of the approaches follow the same line of reasoning. Anily and Federgruen [14], for instance, impose the following restrictions on the generation and acceptance probabilities.

- The generation probabilities are given by the connectivity condition of (1.6), i.e.,

$$\forall i, j \in \mathcal{S} \exists p \geq 1 \exists l_0, l_1, \dots, l_p \in \mathcal{S}$$

$$\text{with } l_0 = i, l_p = j, \text{ and } G_{l_k l_{k+1}} > 0, \quad k = 0, 1, \dots, p-1. \quad (1.15)$$

- The acceptance probability is an asymptotically monotone function satisfying

$$\lim_{c \downarrow 0} A_{ij}(c) = 0 \text{ if } f(j) > f(i) \text{ and } \lim_{c \downarrow 0} A_{ij}(c) = 1 \text{ if } f(j) \leq f(i), \text{ and} \quad (1.16)$$

$$\sum_{k=0}^{\infty} (\underline{A}(c_k))^p = \infty, \quad (1.17)$$

where p denotes the maximum number of steps needed to reach j from a i in (1.15), and

$$\underline{A}(c) = \min_{i,j} \{A_{ij}(c) \mid i \in \mathcal{S}, j \in \mathcal{N}(i)\}.$$

Cruz and Dorea [31] argue that the conditions of (1.15), (1.16), and (1.17) are hard to verify from a practical point of view, and show that they can be replaced with the following more easy conditions.

- The generation probability matrix is irreducible, which is realized by assuming the connectivity condition of (1.6), and furthermore, $G_{ii} > 0$ for all $i \in \mathcal{S}$.
- The acceptance probabilities satisfy

$$\lim_{c \downarrow 0} A_{ij}(c) = 0 \text{ if } f(j) > f(i) \text{ and } \lim_{c \downarrow 0} A_{ij}(c) = 1 \text{ if } f(j) \leq f(i), \text{ and}$$

$$\text{for } f(l) > f(j) > f(i) : \lim_{c \downarrow 0} \frac{A_{il}(c)}{A_{ij}(c)} = 0 \text{ and } \lim_{c \downarrow 0} \frac{A_{il}(c)}{A_{jl}(c)} = 0.$$

Del Moral and Miclo [32] present conditions and rigorous proofs of convergence of generalized simulated annealing with time dependent cost functions. They extend their treatment to asymptotic convergence results for population based simulated annealing approaches including parallel simulated annealing and stochastic genetic algorithms.

The connectivity condition of condition (1.6) implies that the generation probabilities and the neighborhoods should be such that between any two solutions there exists a path along neighboring solutions that can be obtained with finite probability. This condition can be relaxed by the following more general necessary and sufficient weak connectivity condition

$$\forall i \in \mathcal{S} \quad \exists i^* \in \mathcal{S}^*, p \geq 1 \quad \exists l_0, l_1, \dots, l_p \in \mathcal{S},$$

with $l_0 = i, l_p = i^*$, and $G_{l_k l_{k+1}} > 0, \quad k = 0, 1, \dots, p - 1.$ (1.18)

According to this condition it should be possible to construct a finite sequence of transitions with non-zero generation probability, leading from an arbitrary solution i to some optimal solution i^* . For the proof of the validity of this condition, a distinction must be made between *transient* and *recurrent* solutions, where a solution is called transient if the probability that the Markov chain ever returns to that solution equals zero, and recurrent if the Markov chain may return to the solution with a positive probability (Feller [39]). Furthermore, the stationary distribution of (1.7) does not apply any more and should be replaced by a *stationary matrix* $Q(c)$ whose elements q_{ij} denote the probability of finding a solution j after an infinite number of transitions, starting from a solution i . A more detailed treatment of this is beyond the scope of this chapter. The interested reader is referred to Connors and Kumar [30], Gidas [47], and Van Laarhoven et al. [75].

Van Laarhoven et al. [75] claim that the weak connectivity condition of (1.18) assures asymptotic convergence to optimality for applications of simulated annealing to the job shop scheduling problem with neighborhoods applying reversals of arcs on the critical path in the directed graph that can be associated with the well-known disjunctive graph representation in the job shop scheduling problem. Kolonko [69] disproves this assertion with a counter example showing that simulated annealing converges to suboptimal solutions. The proof of Van Laarhoven et al. [75] assumes symmetry of the generation probabilities which is not assured by the reversal neighborhoods in the job shop scheduling problem.

1.5 PERFORMANCE ANALYSIS

In the previous sections we discussed the asymptotic convergence of simulated annealing. Convergence to the set of optimal solutions can be guaranteed only after an infinite number of transitions. In any finite-time implementation one obtains an approximation of the asymptotic convergence behavior.

Seneta [104] has proved that the speed of convergence of a finite homogeneous Markov chain is determined by the second largest eigenvalue of the associated transition matrix according to the following expression

$$\|a(k) - q(c)\| = \mathcal{O}(k^s |\lambda_2(c)|^k), \quad (1.19)$$

where $a(k)$ denotes the probability distribution of the outcomes after k trials, $q(c)$ the corresponding stationary distribution, and $\lambda_2(c)$ ($0 < |\lambda_2(c)| < 1$) the second largest eigenvalue of transition matrix $P(c)$ with multiplicity m_2 ,

and $s = m_2 - 1$. However, computation of $\lambda_2(c)$ is impracticable, due to the large size of the matrix $P(c)$. Aarts and Van Laarhoven [3] have determined the following approximation of the norm in (1.19). Let ε denote an arbitrarily small positive number. Then

$$\|a(k) - q(c)\| < \varepsilon, \quad (1.20)$$

if

$$k > K \left(1 + \frac{\ln(\frac{1}{2}\varepsilon)}{\ln(1 - \gamma^K(c))} \right), \quad (1.21)$$

where $\gamma(c) = \min_{i,j \in S} \{P_{ij}(c) \mid P_{ij}(c) > 0\}$ and $K = |S|^2 - 3|S| + 3$. From (1.20) and (1.21) it follows that the stationary distribution is only approximated arbitrarily closely, if the number of transitions is at least quadratic in the size of the solution space. Moreover, as $|S|$ is exponential in the problem size for most combinatorial optimization problems, arbitrarily close approximation of the stationary distribution leads to an exponential-time execution of simulated annealing.

Mitra et al. [87] obtained the following performance bound for the asymptotic convergence of the inhomogeneous simulated annealing algorithm. Let the transition probabilities of the inhomogeneous Markov chain associated with simulated annealing be defined by (1.3), (1.4) and (1.5), and let the sequence of control parameter values be given by $c_k = \frac{\Gamma}{\log(k+k_0)}$, $k = 0, 1, \dots$ for some positive k_0 , and $\Gamma > r\Delta$, where $\Delta = \max_{i \in S} \max_{j \in N(i)} \{|f(j) - f(i)|\}$ and $r = \min_{i \in S \setminus \hat{S}} \max_{j \in S} d(i, j)$, were the *distance* $d(i, j)$ between two solutions $i, j \in S$ is defined as the length d of the shortest sequence of solutions (l_0, l_1, \dots, l_d) , with $l_0 = i, l_d = j$, and $P_{l_m l_{m+1}}(c) > 0, l_m \in S, m = 0, 1, \dots, d-1$, and \hat{S} denotes the set of all locally minimal solutions.

Furthermore, let q^* be the uniform probability distribution on the set of optimal solutions defined by (1.8). Then for $k \rightarrow \infty$,

$$\|a(k) - q^*\| < \varepsilon,$$

for an arbitrarily small positive number ε , if

$$k = O\left(\varepsilon^{-\max(a,b)}\right),$$

where

$$a = \frac{r^{r\Delta/\Gamma}}{w^r} \quad \text{and} \quad b = \frac{r\Delta}{\hat{f} - f^*},$$

with $\hat{f} = \min_{i \in S \setminus S^*} f(i)$ and $w = \min_{i \in S} \min_{j \in N(i)} G_{ij}$.

Evaluation of this bound for particular problem instances typically leads to a number of transitions that is larger then the size of the solution space and thus to an exponential-time execution for most problems. For instance, in the case of the traveling salesman problem Aarts and Korst [1] show that

$$k = O\left(n^{n^{2n-1}}\right).$$

Note that $|S| = O(n^n)$ and hence complete enumeration of all solutions would take less time than approximating an optimal solution arbitrarily closely by the simulated annealing algorithm.

Several authors have investigated possibilities of speeding up the convergence of optimal simulated annealing for specific problems by taking into account the combinatorial structure of the problem at hand. Sorkin [106] proved that if the neighborhoods of a problem exhibit certain fractal properties, than the time complexity of optimal simulated annealing is polynomial. More specific, he showed that, for problems with properly scaled cost functions between 0 and 1, and a fractal neighborhood structure, a solution of expected cost no greater than ε can be found in a time bounded by a polynomial in $1/\varepsilon$, where the exponent of the polynomial depends on the fractal.

Stander and Silverman [107] discuss a simple global optimization problem and propose an optimal method for lowering the value of the control parameter based on the use of dynamic programming techniques. The resulting time complexity is still exponential but the method provides optimal choices for the initial and final values of the control parameter. Christoph and Hoffman [27] address the scaling behavior of optimal annealing. They found that *dominating barriers* exist at which the value of the control parameter must be lowered more slowly than in between the barriers.

Rajasekaran and Reif [98] improved the convergence rate of optimal annealing by exploiting a special property of the cost function, if present, which they call *small-separability*. Based on this concept, they developed an algorithm called *nested annealing*, which is a simple modification of the classical simulated annealing algorithm obtained by assigning different control parameter values to different regions. For a specific class of problems in computer vision and circuit layout they proved that the time complexity of their optimal simulated algorithm is $2^{O(\sqrt{n})}$ instead of $2^{\Omega(n)}$, where n refers to the size of the problem instance at hand.

Nolte and Schrader [90, 91] have derived both positive and negative results on the convergence rate of simulated annealing for the graph 3-coloring problems. For a specific type of graphs, they show that the expected number of iterations before hitting on an optimal solution is exponential in the number of vertices. Alternatively, for certain random graphs the authors show convergence to optimality with high probability in sublinear time. This is remarkable since the number of steps of the algorithm is smaller than the number of edges.

Steinhöfel et al. [108] study the performance of a logarithmic cooling schedule for the job shop scheduling problem. They prove a run time bound of $O(\log^{1/\rho} 1/\delta) + 2^{O(l_{\max})}$ to obtain with probability $1 - \delta$ the optimal makespan, where ρ denotes a positive constant and l_{\max} the maximum number of consecutive transitions that increase the cost.

1.6 COOLING SCHEDULES

We now consider finite-time implementations of the algorithm, which, as a consequence of the above, can no longer guarantee to find an optimal solution,

but may result in much faster executions of the algorithm without giving in too much on the solution quality. A *finite-time* implementation of simulated annealing is obtained by generating a sequence of homogeneous Markov chains of finite length at descending values of the control parameter. For this, a set of parameters must be specified that govern the convergence of the algorithm. This set of parameters is referred to as a cooling schedule. More precisely, a cooling schedule specifies

- an *initial value* of the control parameter,
- a *decrement function* for lowering the value of the control parameter,
- a *final value* of the control parameter specified by a *stop criterion*, and
- a finite *length* of each homogeneous Markov chain.

Typical cooling schedules in simulated annealing start off at sufficiently large values of c_k , allowing acceptance of virtually all proposed transitions. Next, the decrement function and the Markov chain lengths are chosen such that at the end of each individual Markov chain, the probability distribution of the solutions is close to the stationary distributions, which is referred to as *quasi-equilibrium*. Since at large values of c_k the probability distribution of the solutions equals the stationary distribution by definition, cf. Theorem 1, one may expect that the cooling schedule enables the probability distribution to ‘closely follow’ the stationary distributions, so as to arrive eventually, as $c_k \downarrow 0$, close to q^* , the uniform distribution on the set of optimal solutions given by (1.8). It is intuitively clear that large decrements in c_k require longer Markov chain lengths in order to restore quasi-equilibrium at the next value c_{k+1} of the control parameter. Thus, there is a trade-off between large decrements of the control parameter and small Markov chain lengths. Usually, one chooses small decrements in c_k to avoid extremely long chains, but alternatively, one could use large values for the Markov chain length L_k in order to be able to make large decrements in c_k .

The search for adequate cooling schedules has been the subject of many studies over the past years. For extensive reviews we refer to Van Laarhoven and Aarts [74], Collins et al. [29], and Romeo and Sangiovanni-Vincentelli [101]. Below, we discuss some recent results.

Optimal schedules. Recently, researchers have been investigating optimal finite-time schedules, where optimal refers to the best average cost obtained in finite time. Strenski and Kirkpatrick [109] analyze a small instance of a graph partitioning problem and use an approach based on evaluating exactly the probability distributions of outcomes of the Markov chain associated with the simulated annealing algorithm. They find that different schedules, including iterative improvement, may be optimal depending on the employed schedule length. When a sufficiently long schedule is employed, annealing replaces iterative improvement as the optimal schedule. Furthermore, they observe that optimal schedules may be non-monotone. This result is rather unexpected since

the convergence proofs of simulated annealing suggest a monotone lowering of the control parameter value; see for instance Aarts and Korst [1]. Nevertheless, it is in accordance with earlier theoretical results obtained by Hajek and Sasaki [51], who found for a small artificial problem that the control parameter values of an optimal annealing schedule are all either 0 or ∞ . Related results are derived for fixed-temperature annealing by Cohn and Fielding [28] and Fielding [41].

The approach of Strenski and Kirkpatrick [109] has been further pursued by Boese and Kahng [20]. They introduce the concept of *best-so-far* versus *where-you-are*. More specifically, they use an acceptance criterion based on the cost of the best solution found so far, instead of the cost of the current solution. They determine optimal cooling schedules for two small instances of the traveling salesman and the graph partitioning problem and found that optimal sequences of control parameter values may not be monotone. The analysis of optimal finite-time schedules is interesting, but the results obtained so far are only proved to hold for extremely small instances. At present it is not clear which impact they have on larger instances. One might argue that the whimsical structure of small instances may introduce artifacts that are absent from the more regularly structured large instances. In that case the non-monotonicity results would only hold for a specific class of small problem instances.

Heuristic schedules. Most of the existing work on cooling schedules presented in the literature deals with heuristic schedules. We distinguish between two broad classes: static and dynamic schedules. In a *static cooling schedule* the parameters are fixed; they cannot be changed during execution of the algorithm. In a dynamic cooling schedule the parameters are adapted during execution of the algorithm. Below we review some examples.

Static cooling schedules. The original cooling schedule introduced by Kirkpatrick et al. [66] is known as the *geometric schedule* and applies the rules.

Initial value of the control parameter. $c_0 = \Delta$, where Δ is the maximal difference in cost between any two neighboring solutions. Since exact calculation of Δ is quite time consuming in many cases one often resort to simple estimates.

Lowering the control parameter value. $c_{k+1} = \alpha \cdot c_k$, $k = 0, 1, \dots$, where α is a positive constant smaller than but close to 1. Typical values lie between 0.8 and 0.99.

Final value of the control parameter. The final value is fixed at some small value that is related to the smallest possible difference in cost between two neighboring solutions.

Markov chain length. The length of Markov chains is fixed by some number that is related to the size of the neighborhoods.

Dynamic cooling schedules. There exist many extensions of the simple static schedule presented above that lead to a dynamic schedule. For instance, a sufficiently large value of c_0 may be obtained by requiring that the *initial*

acceptance ratio χ_0 , defined as the number of accepted transitions at c_0 , is close to 1. This can be achieved by starting off at a small positive value of c_0 and multiplying it with a constant factor, larger than 1, until the corresponding value of χ_0 , which is calculated from a number of generated transitions, is close to 1. Typical values of χ_0 lie between 0.9 and 0.99. An adaptive calculation of the final value of the control parameter may be obtained by terminating the execution of the algorithm at a c_k -value for which the value of the cost function of the solution obtained in the last trial of a Markov chain remains unchanged for a number of consecutive chains. Clearly such a value exists for each local minimum that is found. The length of Markov chains may be determined by requiring that at each value c_k a minimum number of transitions is accepted. However, since transitions are accepted with decreasing probability, one would obtain $L_k \rightarrow \infty$ for $c_k \downarrow 0$. Therefore, L_k is usually bounded by some constant L_{\max} to avoid extremely long Markov chains for small values of c_k .

In addition to this basic dynamic schedule the literature presents a number of more elaborate schedules. Most of these schedules are based on a statistical analysis of the simulated annealing process, thus allowing a more theoretical estimation of the parameters. For basic simulated annealing the statistical analysis leads to a model for the cost distribution that resembles an exponential distribution at low c -values and a normal distribution at high c -values. Within this model the first two moments of the resulting distribution are given by Aarts and Korst [1]

$$\mathbb{E}_c(f) = \mathbb{E}_\infty(f) - \frac{\sigma_\infty^2(f)}{c} \left(\frac{\gamma c}{\gamma c + 1} \right) \quad (1.22)$$

and

$$\sigma_c^2(f) = \sigma_\infty^2(f) \left(\frac{\gamma c}{\gamma c + 1} \right)^2,$$

with $\gamma = (\mathbb{E}_\infty(f) - f^*)/\sigma_\infty^2(f)$. $\mathbb{E}_c(f)$ and $\sigma_c^2(f)$ can be computed by using approximate values for $\mathbb{E}_\infty(f)$ and $\sigma_\infty^2(f)$ given by the average cost value of the solutions and the corresponding standard deviation, respectively.

The analysis given above is used by several authors to derive adaptive parameter estimates. As an example we discuss the schedule proposed by Huang et al. [56], since this schedule is quoted in the literature as the most efficient one among those whose implementation require only a modicum of sophistication. For instance, the schedule of Lam and Delosme [77] is conjectured to be even more efficient but its intricacy generally hinders practical use.

Initial value of the control parameter. From (1.22) it follows directly that $\mathbb{E}_c(f) \approx \mathbb{E}_\infty(f)$ for $c \gg \sigma_\infty^2(f)$. Hence c_0 may be chosen as $c_0 = K\sigma_\infty^2(f)$, where K is a constant typically ranging from 5 to 10.

Lowering the control parameter value. Here the concept of quasi-equilibrium is quantified by requiring that the average cost difference for two consecutive Markov chains is small, i.e., $\mathbb{E}_{c_{k+1}}(f) - \mathbb{E}_{c_k}(f) = -\epsilon$ for some small positive number ϵ . Next, by using

$$\frac{\partial}{\partial \ln c} \mathbb{E}_c(f) = \frac{\sigma_c^2(f)}{c}, \quad (1.23)$$

and replacing the left hand side of (1.23) by the differential quotient, we obtain

$$\frac{\mathbb{E}_{c_{k+1}}(f) - \mathbb{E}_{c_k}(f)}{\ln c_{k+1} - \ln c_k} = \frac{\sigma_{c_k}^2(f)}{c_k}.$$

This results in a decrement rule given by

$$c_{k+1} = c_k \exp \left(-\frac{\epsilon c_k}{\sigma_{c_k}^2(f)} \right), \quad (1.24)$$

where, for practical purposes, $\sigma_{c_k}(f)$ is approximated by the measured deviation. In their original paper, Huang et al. [56] replace ϵ by $\lambda \sigma_{c_k}$, $\lambda < 1$, which gives only a slight modification of (1.24).

Final value of the control parameter. Execution is terminated if at the end of a Markov chain the condition is set that $f'_{\max} - f'_{\min} = \Delta f'_{\max}$, where f'_{\max} and f'_{\min} denote the maximum and minimum cost value, respectively, and $\Delta f'_{\max}$ the maximum cost difference of the solutions accepted during the generation of that chain. If the condition holds, c is set to 0, and the execution is concluded with a simple local search to ensure local optimality of the final solution.

Markov chain length. Statistical analysis leads to the observation that in equilibrium the fraction of solutions generated with cost values within a certain range ϵ from the expected cost reaches a stationary value κ . Assuming a normal distribution of the cost values, Huang et al. [56] show that $\kappa = -\text{erf}(\epsilon/\sigma_c(f))$, where $\text{erf}(x)$ is the well-known *error function*. The Markov chain length is determined by the number of trials L_k for which $L_k^* = p\kappa$, where p is a parameter depending on the size of the problem instance, and L_k^* is defined as the number of accepted solutions with a cost value within the interval $(\mathbb{E}_c - \epsilon, \mathbb{E}_c + \epsilon)$. An additional bound on L_k is introduced to avoid extremely long Markov chains.

We end this section by discussing some results that recently appeared in the literature. Park and Kim [94] propose a systematic procedure for setting the parameters in a cooling schedule, using a simplex method for nonlinear programming. Results of using this procedure are compared with results of parameter values that are selected through extensive experiments, for three combinatorial optimization problems, showing a comparable performance.

Nourani and Anderson [92] compare different proposed cooling schedules in order to find the cooling schedule that has the least total entropy production during the annealing process for given initial and final states and fixed number of iterations. The cooling schedules considered are constant thermodynamic speed, exponential, logarithmic, and linear cooling schedules. Experiments show that the constant thermodynamic speed schedule, introduced by Andersen [11], performs best.

An interesting alternative to using the temperature as control parameter, is to use the acceptance probability to control the annealing process, as proposed by Poupaert and Deville [97]. The acceptance probability has two natural

bounds in the annealing process: it is close to 1 at the start and close to 0 at the end. The authors propose an exponential decrease of the acceptance probability during the annealing process. For each intended acceptance probability, a corresponding temperature is estimated. Experiments show a performance that is comparable to the the schedule proposed by Huang et al. [56].

1.7 ISSUES FROM PRACTICE

During its twenty years of existence, simulated annealing has been applied to a large variety of problems ranging from practical real-life problems to theoretical test problems. Two appealing examples of real-life applications are the scheduling of the Australian state cricket season by Willis and Terrill [121] and the design of keyboards for typewriters by Light and Anderson [80]. VLSI design, atomic and molecular physics, and picture processing are the three problem areas in which simulated annealing is probably most frequently applied. The set of theoretical test problems includes about all the well-known problems in discrete mathematics and operations research such as coding, graph coloring, graph partitioning, sequencing and scheduling problems; see Aarts and Lenstra [3]. General overviews of applications of simulated annealing are given by Aarts and Korst [1], Collins et al. [29], Dowsland [34], Van Laarhoven and Aarts [74], and Vidal [117]. Overviews of applications in operations research are given by Egglese [36] and Koulamas, Antony and Jaen [70]. Studies emphasizing performance issues for theoretical test problems are given by several authors. Probably one of the most elaborate studies is presented by Johnson et al. [61, 62], who report on an extensive numerical study for several combinatorial optimization problems including graph partitioning, graph coloring and number partitioning problems. This work provides many practical findings that in our opinion reflect the general experience of annealing practitioners. Perhaps the most striking element is the performance ambivalence that was observed. For the graph partitioning problem, simulated annealing seems to outperform all existing approximation algorithms, whereas for the number partitioning problem the performance is hopelessly poor. Although this bad performance for the number partitioning problem can be understood from analytical arguments, there seems no way to adapt the algorithm in order to improve it.

Furthermore, the literature presents results of studies in which the performance of simulated annealing is compared with that of other local search algorithms. Recent results for the job shop scheduling problem are presented by Vaessens et al. [115] and for the traveling salesman problem by Johnson and McGeoch [63]. With some restraint one may conclude from these studies that simulated annealing, if large running times are allowed, can outperform many other algorithms with respect to effectiveness. Such conclusion however should be handled with care because they strongly depend on the way the investigations have been carried out and on the quality measures that are used. This can be illustrated by the following discussion on the performance of simulated annealing for quadratic assignment. Pardalos et al. [93] report that simulated

annealing can find acceptable solutions within fewer iterations than tabu search. Batitti and Tecchiolli [17] question this conclusion and argue that it no longer holds for difficult problem instances if high quality solutions are required.

Broadly speaking, after twenty years of practical experience, it is widely accepted that simulated annealing can find good solutions for a wide variety of problems, but often at the cost of substantial running times. As a result of this, the true merits of the algorithm become obvious in industrial problem settings, where running times are of little or no concern. As an example we mention design problems, since in those cases one is primarily interested in finding high-quality solutions, whereas design time often only plays a minor role. A well-known successful simulated annealing area in this respect is VLSI design (Kravitz and Rutenbar [71]; Sechen and Sangiovanni-Vincentelli [103]; Shahrok and Mazumder [105]; Wong et al. [122]).

1.8 VARIANTS OF SIMULATED ANNEALING

The literature presents many variants of simulated annealing that are intended to improve its performance. Many of these approaches are aimed at reducing the potentially burdensome running times required by simulated annealing to converge to good solutions. Below, we mention a few examples.

Greene and Supowit [48] introduced the *rejectionless method* as an example of a deterministic simulated annealing approach based on an improved generation mechanism. They propose to generate new solutions with a probability proportional to the effect of a transition on the cost function. In this way, a subsequent solution is directly chosen from the neighborhood of a given solution, i.e., no rejection of solutions takes place. This method leads to shorter Markov chains for a number of problems. However, the efficient use of the method depends strongly on some additional conditions on the neighborhood function, which, unfortunately, cannot be met by many combinatorial optimization problems. Fox [42, 43] further elaborates on this issue. He introduces the concept of *self loop elimination* and shows that this not only speeds up simulated annealing, but also causes the algorithm to be more efficient than multi-start iterative improvement with random restarts, thus contradicting an assertion made by Ferreira and Žerovnik [40], who stated the opposite. Recent papers on deterministic annealing are Jagota et al. [60] and Tsuchiya et al. [112].

Dueck and Scheuer [35] introduced the *threshold accepting* algorithm, where the probabilistic acceptance of annealing is replaced by using a deterministic threshold. In that case, a transition is accepted so long as it does not increase the cost by more than the threshold value, where the value of the threshold decreases during the execution of the algorithm. The convergence of the threshold accepting algorithm is addressed by Althöfer and Koschnick [7].

Herault [53] considers an improved variant called *rescaled simulated annealing*, which is particularly adapted to combinatorial optimization problem where the available running times are limited. The approach is based on rescaling the values of the cost function before the Metropolis criterion is applied. Herault

proposes to replace the cost value $f(i)$ of a solution i by the expression

$$f'(i) = (\sqrt{f(i)} - \sqrt{f_0})^2, \quad (1.25)$$

where f_0 denotes a *target cost* that scales with the value of the control parameter. For target cost equal to 0, the rescaled cost equals the original cost. For large values of the target cost the minima of the rescaled cost equal the maxima of the original cost. Herault proposes to use

$$f_0 = \bar{f} \left(\frac{c}{c_0} \right)^2, \quad (1.26)$$

where c_0 is the initial value of the control parameter and \bar{f} the average value of the cost function over all possible solutions. The author claims that due to the inversion of the extreme solutions and due to the fact that the local minima of the rescaled cost function are shallower than those of the original cost function, convergence is more easy from a practical point of view. Herault proves asymptotic convergence of the rescaled simulated annealing method, and though the convergence rates are still logarithmic, the lower bound on the decrement of the control parameter is smaller than in the case of classical simulated annealing. Herault also presents numerical evidence for this assertion by showing results for the traveling salesman problem that indicate that rescaled simulated annealing requires less transitions to find the same quality results as in classical simulated annealing.

Amin [8] proposes *simulated jumping*, where alternately the control parameter is rapidly increased ('heating') and decreased ('cooling') without reaching equilibrium, where heating and cooling continuously act on different parts of the problem. Amin discusses its application to the quadratic assignment problem and the asymmetric traveling salesman problem, and claims excellent results.

1.9 GLOBAL OPTIMIZATION

Simulated annealing has primarily been applied to combinatorial optimization problems. However, during the past decade simulated annealing has also been applied to several other types of optimization problems, including global optimization, multi-criteria optimization, and discrete stochastic optimization. Of these three, we review global optimization in more detail below. We already mentioned some convergence results for discrete stochastic optimization problems at the end of Section 1.3. Examples of how simulated annealing can be applied to multi-criteria optimization problems are given by Köktener Karasakal and Köksalan [68], and Tuyttens et al. [113].

Global optimization refers to the problem of finding extreme points in an n -dimensional real-valued space. More specifically, given an objective function $f : \mathbb{R}^n \rightarrow \mathbb{R}$, find a point $x^* \in \mathbb{R}^n$ for which $f(x^*)$ is optimal. Several authors have addressed the application of simulated annealing to this class of optimization problems both from a theoretical and a practical point of view. Similar as in the application of simulated annealing to combinatorial optimization problems, the

algorithm continually generates a point $y \in \mathbb{R}^n$ as a neighbor from the current point $x \in \mathbb{R}^n$, and accepts y as the new current point with a probability depending on the difference in objective function values between y and x . The analysis is again based on the combined use of a generation and an acceptance probability. In *classical simulated annealing* (CSA) these probabilities are given by

$$g(c_k, \Delta x) = g_0 \exp\left(-\frac{(\Delta x)^2}{c_k}\right) \quad (1.27)$$

and

$$a(c_k, \Delta f) = \min\left\{1, \exp\left(-\frac{\Delta f}{c_k}\right)\right\}, \quad (1.28)$$

respectively, where $\Delta x = y - x$, $\Delta f = f(y) - f(x)$, and g_0 a normalization constant. Geman and Geman [45] where the first to show asymptotic convergence of CSA under the necessary and sufficient condition that the value of the control parameter is decreased logarithmically as a function of the iteration number k , i.e.,

$$c_k = \frac{c_0}{\log(k+1)}, \quad (1.29)$$

with $k > 0$; see also Theorem 2. Szu and Hartley [110] proposed *fast simulated annealing* (FSA) in which they replace the Gaussian distribution of (1.27) by a Cauchy distribution of the form

$$g(c_k, \Delta x) = g_0 \frac{c_k}{[c_k^2 + (\Delta x)^2]^{(n+1)/2}}. \quad (1.30)$$

They prove asymptotic convergence using a cooling rate that is inversely proportional to the iteration number, hence indicating that FSA converges much faster than CSA. This approach has been further refined by Ingber [57], who proposes a technique called *very fast simulated re-annealing* permitting an exponential cooling rate. The application of these approaches is limited to the optimization of continuous real-valued functions, which prohibits their use in the many existing combinatorial optimization problems. Ingber [58] also compared a number of these simulated annealing approaches for their practical use.

More recently, Andricioaei and Straub [12] and Tsallis and Stariolo [111] introduced a new approach which they called *generalized simulated annealing* (GSA) -not to be confused with generalized simulated annealing of Section 1.4. Their approach assumes expressions for the generation and acceptance probabilities given by

$$g(c_k, \Delta x) = g_0 \frac{(c_k)^{-3n/(2-a)}}{[1 + a(\Delta x)^2/(c_k^2)^{2/(2-a)}]^{1/a+(n-1)/2}} \quad (1.31)$$

and

$$a(c_k, \Delta f) = \min\left\{1, [1 + b\Delta f/c_k]^{-1/b}\right\}, \quad (1.32)$$

where $a > o$ and $b > 0$ are two real-valued parameters, and

$$c_k = c_0 \frac{2^a - 1}{(1 + k)^a - 1}. \quad (1.33)$$

For $a = 0$ and $b \downarrow 0$ GSA recovers CSA, and for $a = 1$ and $b \downarrow 0$ GSA recovers FSA. Moreover, for $a > 1$ the convergence rate of GSA is faster than that of CSA and FSA.

Xiang and Gong [123] have studied the efficiency of GSA through a comparative study with CSA and FSA for several objective functions. They conclude that for all objective functions they considered, GSA outperforms CSA and FSA, and that the performance gap increases with increasing complexity of the objective function. Nishimori and Inoue [89] studied the convergence of GSA for a general class of generation probabilities and they proved asymptotic convergence to optimality under mild conditions. Yang [124] and Locatelli [83] present similar convergence studies in which they relate both the generation probabilities and the decrement functions of the control parameter to the iteration number.

Several authors consider the application of simulated annealing to constrained global optimization, i.e., the optimization of real-valued functions over a feasible region Q . In this case the selection of a neighboring point becomes more intricate because of the presence of the boundaries of Q , and the problem of *jамming*, which occurs when the algorithm reaches points close to one of the boundaries of the feasible region. To cope with this issue, Romeijn and Smith [99] introduce a class of algorithms which they called hide-and-seek that apply a so-called hit-and-run generator to select neighboring points of the form

$$x_{k+1} = x_k + \lambda d_k, \quad (1.34)$$

where $d_k \in \mathbb{R}^n$ determines a search direction and $\lambda \in \mathbb{R}$ a step size. They introduce efficient means to determine values for λ and d_k , and prove asymptotic convergence of this class of algorithms. Romeijn et al. [100] further elaborate on this concept by introducing a modified hit-and-run generator. This generator uses reflection which basically is aimed at directing the search away from boundary points. They also show that this method can be effectively applied to a class of mixed-integer global optimization problems in the field of structural optimization. They proved convergence in probability to globally optimal points.

Wah and Wang [118] analyze the application of simulated annealing to constrained global optimization problems within a penalty framework using Lagrange multipliers. They derived necessary and sufficient conditions for asymptotic convergence to globally optimal points, by coupling the ascent in the Lagrangian function space with the ascent in the Lagrangian multiplier space. They also present some numerical results of their approach for a number of objective functions and constraints of various types, indicating good performance, i.e., fast convergence to good final points.

Finally, we mention that several authors analyzed the asymptotic convergence of simulated annealing for global optimization within the framework of

Langevin type Markov diffusion processes using stochastic differential equations. Examples are Geman and Huang [46], Gelfand and Mitter [44], and Yin [128]. These studies provide a thorough basis for the mathematical modeling of continuous-time simulated annealing processes. In general terms the rate of convergence in the continuous-time case is comparable to that of the discrete-time case.

1.10 NEURAL NETWORKS

The link between simulated annealing and neural networks is twofold. Some neural network models can be considered as massively parallel implementations of simulated annealing. For this subject we refer to Section 1.11. In addition, several neural network models use a simulated annealing approach to govern self-organization. This follows naturally from the strong resemblance between neural networks and many-particle systems in statistical physics; see for instance Herz et al. [54]. We elaborate on this latter subject by considering a class of stochastic two-state neural networks that includes Boltzmann machines (Aarts and Korst [1]; Ackley et al. [4]). A stochastic two-state neural network consists of a set \mathcal{U} of two-state neurons that are interconnected by weighted symmetric connections. A *neuron* u can be in one of two *states*, either it is firing, corresponding to state ‘1’, or it is not firing, corresponding to state ‘-1’. A *configuration* k is a $|\mathcal{U}|$ -dimensional vector that describes the global state of the network. The state of an individual neuron u is given by the component $s_u(k)$. Let $\mathcal{R} = \{-1, 1\}^{|\mathcal{U}|}$ be the set of all possible configurations. With the connection between a neuron u and a neuron v a *weight* $w_{u,v} \in \mathbb{R}$ is associated. By definition, we assume $w_{u,u} = 0$ for each $u \in \mathcal{U}$ and $w_{u,v} = w_{v,u}$ for each pair $u, v \in \mathcal{U}$. Each neuron u has a *threshold* $b_u \in \mathbb{R}$.

Self organization is achieved by allowing neurons to change their states, from ‘-1’ to ‘1’ or vice versa. Let the network be in configuration k , then a *state change* of neuron u results in a configuration l , with $s_u(l) = -s_u(k)$ and $s_v(l) = s_v(k)$ for all neurons $v \neq u$. A possible state change of a neuron u is related to its net input $z_u(k)$, defined as the sum of its weighted inputs minus its threshold, i.e.,

$$z_u(k) = \sum_{v \in \mathcal{U}} w_{u,v} s_v(k) - b_u.$$

The state change rule is given by

$$\begin{aligned} &\text{select randomly a neuron } u; \\ &r := \text{random number in } [0, 1); \\ &\text{if } r < f_c(2z_u(k)) \text{ then } s_u(k) := 1 \text{ else } s_u(k) := -1, \end{aligned} \tag{1.35}$$

where $f_c(y) = 1/(1 + e^{-y/c})$ is the so-called *logistic activation function*. Hence, the selected neuron goes to state ‘1’ with probability $f_c(2z_u(k))$ and to state ‘-1’ with probability $1 - f_c(2z_u(k))$. The parameter c determines the steepness of the logistic function f_c and plays the same role as the control parameter in simulated annealing.

Next we introduce the *energy function* $E : \mathcal{R} \rightarrow \mathbb{IR}$, defined by

$$E(k) = -\frac{1}{2} \sum_{u,v \in \mathcal{U}} w_{u,v} s_u(k) s_v(k) + \sum_{u \in \mathcal{U}} b_u s_u(k).$$

The energy is a global measure indicating to what extent the neurons in the network have reached a consensus about their individual states relative to the weights and thresholds. Let the network be in configuration k , and suppose a state change of a neuron u results in a configuration l , with $s_u(l) = -s_u(k)$ and $s_v(l) = s_v(k)$ for all neurons $v \neq u$. The energy difference corresponding to a state change of u in configuration k equals

$$E(l) - E(k) = 2s_u(k)z_u(k). \quad (1.36)$$

The probability of a state change of neuron u equals

$$\begin{aligned} \mathbb{P}\{\text{accept a state change of neuron } u \mid \text{current state is } k\} = \\ \frac{1}{1 + \exp(\beta(E(l) - E(k)))}. \end{aligned} \quad (1.37)$$

The convergence analysis of Boltzmann machines with the acceptance probability of (1.37) is similar to that of simulated annealing and can be summarized as follows (Aarts and Korst [1]).

- (i) The probability $q_\beta(k)$ of obtaining a configuration k after a sufficiently large number of trials carried out at a fixed value of β is given by

$$q_\beta(k) = \frac{\exp(-\beta E_k/c)}{Z(\beta)}, \quad (1.38)$$

where $Z(\beta) = \sum_{l \in \mathcal{R}} \exp(-\beta E_l/c)$ is a normalization constant.

- (ii) For $\beta \rightarrow \infty$, the Boltzmann machine finds a configuration with minimum energy with probability one.

Ackley et al. [4] used the self-organization described above to formulate a learning algorithm for Boltzmann machines in which the weights are iteratively adjusted to minimize an information theoretic measure that quantifies the difference between the network performance in the *clamped phase* and the *free running phase*. They distinguish between *visible neurons* and *hidden neurons*, and assume that the Boltzmann machine is supposed to learn a given behavior on its visible neurons. In the clamped phase the visible neurons are fixed to a required behavior, and only the hidden neurons can evolve. In the free running phase all neurons can evolve. The difference in behavior between these two phases is used to adapt the weights and thresholds.

Amit et al. [9, 10] studied the relation of stochastic two-state neural networks with statistical many-particle using an analogy of two-state neural networks with *spin glasses*. Spin glasses are physical systems consisting of an ensemble

of spin particles having spin ‘up’ or ‘down’, i.e., $S_i = \pm 1$ that are placed in a magnetic field. The *Hamiltonian* of a spin glass is given by

$$H = -\frac{1}{2} \sum_{i,j} J_{ij} S_i S_j + h_0 \sum_i S_i,$$

where the J_{ij} denote the coupling strengths of the spin-spin interaction between two particles i and j , and h_0 denotes the strength of an external magnetic field. The analogy with two-state neural networks is evident. The spins play the role of the states of the individual neurons and the Hamiltonian is the equivalent of the energy.

The statistical analysis based on the physics analogy has lead to a variety of neural network variant of simulated annealing which are aimed at finding deterministic update rules for the states of the neurons. Using (1.38) the average activity $\langle s_u \rangle_c$ of neuron u of a Boltzmann machine at control parameter value c equals

$$\langle s_u \rangle_c = \frac{\sum_{k \in \mathcal{R}} s_u(k) \exp(-E_k/c)}{\sum_{l \in \mathcal{R}} \exp(-E_l/c)},$$

where $\langle \cdot \rangle_c$ denotes the average over all configurations in equilibrium at a given c -value. This expression can be rewritten as

$$\langle s_u \rangle_c = \frac{\partial F}{\partial b_u},$$

where F denotes the *free energy* given by

$$F = -c^{-1} \log \sum_{k \in \mathcal{R}} \exp(-E(k)/c) = -c \log Z(c).$$

Peterson and Anderson [95] use the statistical averages to determine a deterministic simulated annealing approach based on the concept of mean fields. In *mean field annealing* statistical equilibrium is expressed from the point of view of the *mean field* experienced by a neuron. This means that we approximate the actual net input $z_u(k)$ of a neuron by the mean net input, i.e., the mean field, given by

$$\langle z_u \rangle_c = \sum_{v \in \mathcal{U}} w_{u,v} \langle s_v \rangle_c - b_u.$$

The state change rule (1.34) yields that a neuron with this net input has probability $f(\langle 2z_u \rangle_c)$ of being in state ‘1’ and probability $1 - f(\langle 2z_u \rangle_c)$ of being in state ‘-1’. Using the logistic activation function we find that the mean activation of the neuron satisfies

$$\langle s_u \rangle_c = \tanh \left(c^{-1} \left(\sum_{v \in \mathcal{U}} w_{u,v} \langle s_v \rangle_c - b_u \right) \right). \quad (1.39)$$

The expressions of (1.39) constitute a system of $|\mathcal{U}|$ deterministic non-linear equations with $|\mathcal{U}|$ unknown variables. The solvability of the equations depends

on the weights and the thresholds. For some specific values the solution is simple. For instance, in the case where $w_{u,v} = w_0/|\mathcal{U}|$ for all $u \neq v \in \mathcal{U}$ and $b_u = 0$ for all $u \in \mathcal{U}$, (1.39) reduces to

$$\langle s_u \rangle_c = \tanh(c^{-1}w_0\langle s_u \rangle_c)$$

for all $u \in \mathcal{U}$.

Several authors have considered continuous-state and time variants of the deterministic simulated annealing model presented above. They all follow the same line of argumentation in which the two-state variable $s_u(k) \in \{0, 1\}$ is replaced by a continuous state variable $x_u(k) \in [0, 1]$ reflecting the output of neuron u , and an internal state $y_u(k) \in [0, 1]$ is introduced to reflect the internal state of neuron u . The dynamics is determined by the following equations:

$$x_u(k) = \frac{1}{1 + \exp(-y_u(k)/c_k)} \quad (1.40)$$

and

$$y_u(k+1) = g(k)y_u(k) + \alpha \left(\sum_{v \in \mathcal{U}} w_{u,v} x_v(k) - b_u \right) - h_u(k)x_u(k), \quad (1.41)$$

where $g(k) \in [0, 1]$ is the *input damping factor*, $\alpha \in \mathbb{R}^+$ is the *input scaling factor* and $h_u(k) \in \mathbb{R}^+$ the *self-feedback weight*. Different choices of the function h_u result in different models. For $h_u = 0$ the classical continuous-time Hopfield network of Hopfield and Tank [55] is obtained. Using an exponential form for h_u yields the *chaotic simulated annealing* model of Chen and Aihara [26] which is further refined by Wang and Smith [119], and Kwok and Smith [73]. The refinements lead to performance improvements of the classical Hopfield models which is for instances demonstrated by Chen and Aihara [26] for the traveling salesman problem.

1.11 MISCELLANEOUS

In this last section, we briefly discuss four subjects: (*i*) choice of neighborhood (*ii*) parallel annealing, (*iii*) combined approaches, where annealing is combined with other local search techniques such as tabu search and genetic algorithms, and (*iv*) quantum annealing.

Choice of neighborhood. In contrast to the large number of papers that consider cooling schedules, the issue of selecting an appropriate neighborhood has received relatively little attention. The reason is probably that neighborhoods are usually highly problem dependent. Yao [125] gives a probabilistic analysis of the impact of the size of the neighborhood on the performance of simulated annealing. The analysis implies that a large neighborhood yields a better performance than a small neighborhood, provided that the number of transitions that are required to reach a global optimum is still large. This supports the idea of using a dynamic neighborhood size, that is large at the start of the annealing

algorithm but decreases near the end. This is experimentally confirmed for the flow shop scheduling problem by Liu [82]. The author compares different fixed neighborhood sizes and a dynamic neighborhood of which the size decreases during the execution of the annealing algorithm. Experimental results indicate that the latter outperforms fixed neighborhood sizes.

Related to the choice of neighborhood is the choice of the set solutions that is used for the annealing search. One approach that is often used in practice is to extend the set \mathcal{S} of feasible solutions with infeasible solutions. This is realized by discarding some of the feasibility constraints and including corresponding cost terms in the cost function f , which guarantee that eventually, as the control parameter c approaches 0, the algorithm ends with a feasible solution. As an example, we mention the recent comparison of different search graphs for the multi-level capacitated lot sizing problem presented by Barbarosoğlu and Özdamar [16].

Parallel annealing. Parallel simulated annealing algorithms aim at distributing the execution of the various parts of a simulated annealing algorithm over a number of communicating parallel processors. This is a promising approach to the problem of speeding up the execution of the algorithm, but it is by no means a trivial task, due to the intrinsic sequential nature of the algorithm. Over the years a large variety of approaches have been proposed, leading to both generally applicable and tailored algorithms. For overviews we refer to Aarts and Korst [1], Azencott [15], Boissin and Lutton [22], Greening [49], and Verhoeven and Aarts [116]. Recently, Müller [88] proposed a parallel annealing implementation that avoids the explicit use of a control parameter and a corresponding cooling schedule. The Boltzmann distribution of cost values is accomplished by enforcing an approximately constant total cost of the ensemble during a certain sequence of steps, and implicitly the temperature is lowered during the annealing process. Kliener [67] presents a general software library for parallel simulated annealing.

A special approach to parallel simulated annealing is provided by the use of neural network models, as discussed in Section 1.10. To this end the optimization problem at hand is cast into a 0-1 programming formulation and the values of the decision variables are associated with the states of the neurons in the network. This has led to randomized approaches such as the *Boltzmann machine* (Aarts and Korst [2]), and to deterministic approaches such as the *mean field method* (Peterson and Söderberg [96]). In addition to the speedup obtained by parallel execution, neural networks also offer a speedup through their hardware implementation. This has led to fast VLSI-implementations of simulated annealing (Lee and Sheu [78]) and even to optical implementations (Lalanne et al. [76]).

Combined approaches. Recent approaches to local search concentrate on the combined use of different local search algorithms, which is also referred to as *multi-level approaches* (Vaessens et al. [114]). In a number of these approaches, simulated annealing is used. We mention some examples below. Martin, Otto

and Felten [85] propose a successful simulated annealing algorithm for the traveling salesman problem, which uses a restricted 4-exchange neighborhood, combined with a simple local search algorithm using a 3-exchange neighborhood. Eiben et al. [37] present a stochastic search procedure that combines elements of population genetics with those of simulated annealing. They prove that their stochastic approach exhibits convergence properties similar to those of simulated annealing. For a similar approach that combines annealing and genetic algorithms Yao [126] gives experimental results for the traveling salesman problem. Lin et al. [81] introduce a genetic approach to simulated annealing using population based transitions, genetic-operator based quasi-equilibrium control, and Metropolis-criterion-type selection operations in genetic algorithms' jargon. They find empirically that their approach works quite well for the zero-one knapsack, set partitioning, and traveling salesman problems. For recent papers combining simulated annealing and genetic algorithms we refer to Boettcher and Percus [21], Kurber et al. [72], Li and Jiang [79].

Clearly, the issue of combined approaches opens many possibilities for the design of new variants of local search algorithms. However, one should be careful not to propose these variants as new algorithmic concepts. Research on local search has been fascinating over the past ten years, but it has also suffered a great deal from an extensive confusion, caused by the introduction of many new fancy-named ‘concepts’, which, after their demystification, turned out to be only coarse or well-known heuristic rules.

Hart [52] investigated the performance difference between simulated annealing and evolutionary algorithms from a theoretical point of view. The main result of this study is that, under mild conditions on the mutation and cross over operators, evolutionary algorithms have a larger probability of success than simulated annealing.

Quantum annealing. Recently, Castagnoli et al. [24] suggested quantum annealing as an interesting alternative to replace the sequential computation in quantum computing. Castagnoli [23] also investigated the relationship between quantum annealing and particle physics to obtain alternatives to the local relaxation process in quantum Boolean networks. These investigations are promising and certainly interesting, but they need further substantiation to provide a solid basis for the quantum computing variant of simulated annealing.

References

- [1] E.H.L Aarts and J.H.M. Korst. *Simulated Annealing and Boltzmann Machines*. Wiley, 1989.
- [2] E.H.L. Aarts and J.H.M. Korst. Boltzmann Machines as a Model for Massively Parallel Annealing. *Algorithmica*, 6:437–465, 1991.

- [3] E.H.L.Aarts and P.J.M. van Laarhoven. Statistical Cooling: A General Approach to Combinatorial Optimization Problems. *Philips Journal of Research*, 40:193–226, 1985.
- [4] D.H. Ackley, G.E. Hinton, and T.J. Sejnowski. A Learning Algorithm for Boltzmann Machines. *Cognitive Science*, 9:147–169, 1985.
- [5] T.M. Alkhamis , M.A. Ahmed, and V.K. Tuan. Simulated Annealing for Discrete Optimization with Estimation. *European Journal of Operational Research*, 116:530–544, 1999.
- [6] M.H. Alrefaei and S. Andradottir. A Simulated Annealing Algorithm with Constant Temperature for Discrete Stochastic Optimization. *Management Science*, 45:748–764, 1999.
- [7] I. Althöfer and K.U. Koschnick. On the Convergence of “Threshold Accepting”. *Applied Mathematics and Optimization*, 24:183–195, 1991.
- [8] S. Amin. Simulated Jumping. *Annals of Operations Research*, 86:23–38, 1999.
- [9] D.J. Amit, H. Gutfreund, and H. Sompolinsky. Spin-Glass Models of Neural Networks. *Physical Review A*, 32:1007–1018, 1985.
- [10] D.J. Amit, H. Gutfreund, and H. Sompolinsky. Statistical Mechanics of Neural Networks near Saturation. *Annals of Physics*, 173:30–67, 1987.
- [11] B. Andersen. Finite-Time Thermodynamics and Simulated Annealing. In: *Entropy and Entropy Generation*, J.S. Shiner, editor, pages 111–127, Kluwer, 1996.
- [12] I. Andricioaei and J.E. Straub. Generalized Simulated Annealing Algorithms using Tsallis Statistics: Application to Conformational Optimization of a Tetrapeptide. *Physical Review E*, 53:3055–3058, 1996.
- [13] S. Anily and A. Federgruen. Ergodicity in Parametric Nonstationary Markov Chains: An Application to Simulated Annealing Methods. *Operations Research*, 35:867–874, 1987.
- [14] S. Anily and A. Federgruen. Simulated Annealing Methods with General Acceptance Probabilities. *Journal of Applied Probability*, 24:657–667, 1987.
- [15] R. Azencott (editor). *Simulated Annealing: Parallelization Techniques*. Wiley, 1992.
- [16] G. Barbarosoglu and L. Özdamar. Analysis of Solution Space-Dependent Performance of Simulated Annealing: The Case of the Multi-Level Capacitated Lot Sizing Problem. *Computers and Operations Research*, 27:895–903, 2000.

- [17] R. Battitti and G. Tecchiolli. Simulated Annealing and Tabu Search in the Long Run: A Comparison on QAP Tasks. *Computers and Mathematics with Applications*, 28:1–8, 1994.
- [18] E. Bernstein and U. Vazirani. Quantum Complexity Theory. In: *Proceedings of the 25th ACM Symposium on Theory of Computation*, pages 11–20, San Diego, 1993.
- [19] K. Binder. *Monte Carlo Methods in Statistical Physics*. Springer-Verlag, 1978.
- [20] K.D. Boese and A.B. Kahng. Best-so-Far vs. Where-you-Are: Implications for Optimal Finite-Time Annealing. *Systems and Control Letters*, 22:71–78, 1994.
- [21] S. Boettcher and A.G. Percus. Combining Local Search with Co-Evolution in a Remarkably Simple Way. In: *Proceedings of the 2000 Congress on Evolutionary Computation*, pages 1578–1584, 2000.
- [22] N. Boisson and J.-L. Lutton. A Parallel Simulated Annealing Algorithm. *Parallel Computing*, 19:859–872, 1993.
- [23] G. Castagnoli. Merging Quantum Annealing Computation and Particle Statistics: A Prospect in the Search of Efficient Solutions to Intractable Problems. *International Journal of Theoretical Physics*, 37:457–462, 1998.
- [24] G. Castagnoli, A. Ekert, and C. Macchiavello. Quantum Computation: From Sequential Approach to Simulated Annealing. *International Journal of Theoretical Physics*, 37:463–469, 1998.
- [25] V. Černý. Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.
- [26] L. Chen and K. Aihara. Chaotic Simulated Annealing by a Neural Network Model with Transient Chaos. *Neural Networks*, 8:915–930, 1995.
- [27] M. Christoph and K.H. Hoffmann. Scaling Behaviour of Optimal Simulated Annealing Schedules. *Journal of Physics A*, 26:3267–3277, 1993.
- [28] H. Cohn and M. Fielding. Simulated Annealing: Searching for an Optimal Temperature Schedule. *SIAM Journal on Optimization*, 9:779–802, 1999.
- [29] N.E. Collins, R.W. Eglese, and B.L. Golden. Simulated Annealing — An Annotated Bibliography. *American Journal of Mathematical and Management Sciences*, 8:209–307, 1988.
- [30] D.P. Connors, and P.R. Kumar. Simulated Annealing and Balance of Recurrence Order in Time-Inhomogeneous Markov Chains. In: *Proceedings of the 26th IEEE Conference on Decision and Control*, pages 2261–2263, 1987.

- [31] J.R. Cruz and C.C.Y. Dorea. Simple Conditions for the Convergence of Simulated Annealing Type Algorithms. *Journal on Applied Probability*, 35:885–892, 1998.
- [32] P. Del Moral and L. Miclo. On the Convergence and Applications of Generalized Simulated Annealing. *SIAM Journal on Control and Optimization*, 37:1222–1250, 1999.
- [33] D. Deutch. Quantum Theory, the Church-Turing Principle and the Universal Quantum Computer. In: *Proceedings of the Royal Society of London A*, 400:97–117, 1985.
- [34] K.A. Dowsland. Simulated Annealing. In: *Modern Heuristic Techniques for Combinatorial Problems*, C.R. Reeves, editor, pages 20–69, Blackwell, 1993.
- [35] G. Dueck and T. Scheuer. Threshold Accepting: A General Purpose Optimization Algorithm. *Journal of Computational Physics*, 90:161–175, 1990.
- [36] R.W. Eglese. Simulated Annealing: A Tool for Operational Research. *European Journal of Operational Research*, 46:271–281, 1990.
- [37] A.E. Eiben, E.H.L. Aarts, and K.M. van Hee. Global Convergence of Genetic Algorithms. *Springer Lecture Notes in Computer Science*, 496: 4–13, 1991.
- [38] A. Ekert and R. Josza. Quantum Computation and Shor’s Factoring Algorithm. *Reviews of Modern Physics*, 68:733–753, 1996.
- [39] W. Feller. *An Introduction to Probability Theory and Its Applications*. vol. 1, Wiley, 1950.
- [40] A.G. Ferreira and J. Žerovnik. Bounding the Probability of Success on Stochastic Methods for Global Optimization. *Computers and Mathematics with Applications*, 25:1–8, 1993.
- [41] M. Fielding. Simulated Annealing with an Optimal Fixed Temperature. *SIAM Journal on Optimization*, 11:289–307, 2000.
- [42] B.L. Fox. Integrating and Accelerating Tabu Search, Simulated Annealing, and Genetic Algorithms. In: *Tabu Search*, F. Glover, E. Taillard, M. Laguna, and D. de Werra, editors, *Baltzer, Annals of Operations Research*, 41:47–67, 1993.
- [43] B.L. Fox. Random Restart versus Simulated Annealing. *Computers and Mathematics with Applications*, 27:33–35, 1994.
- [44] S.B. Gelfand and S.K. Mitter. Recursive Stochastic Algorithms for Global Optimization in \mathbb{R}^n . *SIAM Journal on Control and Optimization*, 29:999–1018, 1991.

- [45] S. Geman and D. Geman. Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:721–741, 1984.
- [46] S. Geman and C.R. Huang. Diffusions for Global Optimization. *SIAM Journal on Control and Optimization*, 24:1031–1043, 1986.
- [47] B. Gidas. Nonstationary Markov Chains and Convergence of the Annealing Algorithm, *Journal of Statistical Physics*, 39:73–131, 1985.
- [48] J.W. Greene and K.J. Supowit. Simulated Annealing without Rejected Moves. *IEEE Transactions on Computer-Aided Design*, 5:221–228, 1986.
- [49] D.R. Greening. Parallel Simulated Annealing Techniques. *Physica D*, 42:293–306, 1990.
- [50] B. Hajek. Cooling Schedules for Optimal Annealing. *Mathematics of Operations Research*, 13:311–329, 1988.
- [51] B. Hajek and G. Sasaki. Simulated Annealing: To Cool it or Not. *Systems Control Letters*, 12:443–447, 1989.
- [52] W.E. Hart. A Theoretical Comparison of Evolutionary Algorithms and Simulated Annealing. In: *Proceedings of the 5th Annual Conference on Evolutionary Programming*, pages 147–154, San Diego, 1996.
- [53] L. Herault. Rescaled Simulated Annealing — Accelerating Convergence of Simulated Annealing by Rescaling the State Energies. *Journal of Heuristics*, 6:215–252, 2000.
- [54] J.A. Herz, A. Krogh, and P.G. Palmer. *Introduction to the Theory of Neural Computation*. Addison Wesley, 1991.
- [55] J. Hopfield and D. Tank. Neural Computation of Decisions in Optimization Problems. *Biological Cybernetics*, 52:141–152, 1985.
- [56] M.D. Huang, F. Romeo, and A. Sangiovanni-Vincentelli. An Efficient General Cooling Schedule for Simulated Annealing. In: *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 381–384, 1986.
- [57] L. Ingber. Very Fast Simulated Re-Annealing. *Mathematical and Computer Modelling*, 12:967–973, 1989.
- [58] L. Ingber. Simulated Annealing: Practice versus Theory. *Journal of Mathematical Computer Modelling*, 18:29–57, 1993.
- [59] D. Isaacson and R. Madsen. *Markov Chains*. Wiley, 1976.

- [60] A. Jagota, M. Pelillo and A. Rangarajan. A New Deterministic Annealing Algorithm for Maximum Clique. In: *Proceedings of the IEEE International Joint Conference on Neural Networks*, vol. VI, pages 505–508, 2000.
- [61] D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon. Optimization by Simulated Annealing: An Experimental Evaluation, Part I Graph Partitioning. *Operations Research*, 37:865–892, 1989.
- [62] D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon. Optimization by Simulated Annealing: An Experimental Evaluation, Part II Graph Colouring and Number Partitioning. *Operations Research*, 39:378–406, 1991.
- [63] D.S. Johnson and L.A. McGeoch. The Traveling Salesman Problem: A Case Study. In: *Local Search in Combinatorial Optimization*, E. Aarts and J.K. Lenstra, editors, Wiley, 1997.
- [64] W. Kern. On the Depth of Combinatorial Optimization Problems. Technical Report 86.33., Universität zu Köln, Köln, 1986.
- [65] G. Kesidis and E. Wong. Optimal Acceptance Probability for Simulated Annealing. *Stochastics and Stochastics Reports*, 29:221–226, 1990.
- [66] S. Kirkpatrick, C.D. Gelatt Jr., and M.P. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671–680, 1983.
- [67] G. Kliewer. A General Software Library for Parallel Simulated Annealing. In: *Proceedings of the XVIII EURO Winter Institute on Metaheuristics in Combinatorial Optimisation*, Lac Noir, Switzerland, 2000.
- [68] E. Köktener Karasakal and M. Köksalan. A Simulated Annealing Approach to Bicriteria Scheduling Problems on a Single Machine. *Journal of Heuristics*, 6:311–327, 2000.
- [69] M. Kolonko. Some Results on Simulated Annealing Applied to the Job Shop Scheduling Problem. *European Journal of Operational Research*, 113:123–136, 1999.
- [70] C. Koulamas, S.R. Antony, and R. Jaen. A Survey of Simulated Annealing Applications to Operations Research Problems. *Omega*, 22:41–56, 1994.
- [71] S.A. Kravitz and R. Rutenbar. Placement by Simulated Annealing on a Multiprocessor. *IEEE Transactions on Computer-Aided Design*, 6:534–549, 1987.
- [72] K. Kurbel, B. Schneider, and K. Singh. Solving Optimization Problems by Parallel Recombinative Simulated Annealing on a Parallel Computer — An Application to Standard Cell Placement in VLSI Design. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 28:454–461, 1998.

- [73] T. Kwok and K.A. Smith. A Unified Framework for Chaotic Neural-Network Approaches to Combinatorial Optimization. *IEEE Transactions on Neural Networks*, 10:978–981, 1999.
- [74] P.J.M. van Laarhoven and E.H.L. Aarts. *Simulated Annealing: Theory and Applications*, Reidel, 1987.
- [75] P.J.M. van Laarhoven, E.H.L. Aarts, and J.K. Lenstra. Job Shop Scheduling by Simulated Annealing. *Operations Research*, 40:185–201, 1992.
- [76] P. Lalanne, J.C. Rodier, E. Belhaire, and P.F. Garda. Optoelectronic Devices for Boltzmann Machines and Simulated Annealing. *Optical Engineering*, 32:1904–1914, 1993.
- [77] J. Lam and J.-M. Delosme. Logic Minimization using Simulated Annealing, In: *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 348–351, 1986.
- [78] B.W. Lee and B.J. Sheu. Hardware Annealing in Electronic Neural Networks, *IEEE Transactions on Circuits and Systems*, 38:134–141, 1991.
- [79] B. Li and W. Jiang. A Novel Stochastic Optimization Algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 30:193–198, 2000.
- [80] L.W. Light and P. Anderson. Designing Better Keyboards via Simulated Annealing. *AI Expert*, 9:20–27, 1993.
- [81] F.-T. Lin, C.-Y. Kao, and C.-C. Hsu. Applying the Genetic Approach to Simulated Annealing in Solving some NP-Hard Problems. *IEEE Transaction on Systems, Man, and Cybernetics*, 23:1752–1767, 1994.
- [82] J. Liu. The Impact of Neighbourhood Size on the Process of Simulated Annealing: Computational Experiments on the Flowshop Scheduling Problem. *Computers and Industrial Engineering*, 37:285–288, 1999.
- [83] M. Locatelli. Simulated Annealing Algorithms for Continuous Global Optimization: Convergence Conditions. *Journal of Optimization Theory and Applications*, 104:121–133, 2000.
- [84] M. Lundy and A. Mees. Convergence of an Annealing Algorithm. *Mathematical Programming*, 34:111–124, 1986.
- [85] O. Martin, S.W. Otto, and E.W. Felten. Large Step Markov Chains for the Traveling Salesman Problem. *Complex Systems*, 5:299–326, 1991.
- [86] M. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of State Calculations by Fast Computing Machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

- [87] D. Mitra, F. Romeo, and A.L. Sangiovanni-Vincentelli. Convergence and Finite-Time Behavior of Simulated Annealing. *Advances in Applied Probability*, 18:747–771, 1986.
- [88] D. Müller. Simulated Annealing without a Cooling Schedule. *International Journal of Computer Mathematics*, 66:9–20, 1998.
- [89] H. Nishimori and J.-I. Inoue. Convergence of Simulated Annealing using Generalized Transition Probability. *Journal of Physics, A*, 31:5661–5672, 1998.
- [90] A. Nolte and R. Schrader. Simulated Annealing and its Problems to Color Graphs. *Lecture Notes in Computer Science*, 1136:138–152, 1996.
- [91] A. Nolte and R. Schrader. Coloring in Sublinear Time. *Lecture Notes in Computer Science*, 1284:388–401, 1997.
- [92] Y. Nourani and B. Andresen. A Comparison of Simulated Annealing Cooling Strategies. *Journal of Physics A*, 31:8373–8385, 1998.
- [93] P.M. Pardalos, K.A. Murty, and T.P. Harrison. A Computational Comparison of Local Search Heuristics for Solving Quadratic Assignment Problems. *Informatica*, 4:172–187, 1993.
- [94] M.-W. Park and Y.-D. Kim. A Systematic Procedure for Setting Parameters in Simulated Annealing Algorithms. *Computers and Operations Research*, 25:207–217, 1998.
- [95] C. Peterson and J.R. Anderson. A Mean Field Theory Algorithm for Neural Networks. *Complex Systems*, 1:995–1019, 1987.
- [96] C. Peterson and B. Söderberg. A New Method for Mapping Optimization Problems onto Neural Networks. *International Journal of Neural Systems*, 1:3–22, 1989.
- [97] E. Poupaert and Y. Deville. Simulated Annealing with Estimated Temperature. *AI Communications*, 13:19–26, 2000.
- [98] S. Rajasekaran and J.H. Reif. Nested Annealing: A Provable Improvement to Simulated Annealing. *Theoretical Computer Science*, 99:157–176, 1992.
- [99] H.E. Romeijn and R.L. Smith. Simulated Annealing for Global Constrained Optimization. *Journal of Global Optimization*, 5:101–126, 1994.
- [100] H.E. Romeijn, Z.B. Zabinsky, D.L. Graesser, and S. Neogi. New Reflection Generator for Simulated Annealing in Mixed-Integer/Continuous Global Optimization. *Journal of Optimization Theory and Applications*, 101:403–427, 1999.
- [101] F. Romeo and A. Sangiovanni-Vincentelli. A Theoretical Framework for Simulated Annealing. *Algorithmica*, 6:302–345, 1991.

- [102] P.C. Schuur. *Classification of Acceptance Criteria for the Simulated Annealing Algorithm*. Memorandum COSOR 89-29, Eindhoven University of Technology, Eindhoven, 1989.
- [103] C. Sechen and A.L. Sangiovanni-Vincentelli. The TimberWolf Placement and Routing Package. *IEEE Journal on Solid State Circuits*, 30:510–522, 1985.
- [104] E. Seneta. *Non-Negative Matrices and Markov Chains*. Springer Verlag, 1981.
- [105] K. Shahookar and P. Mazumder. VLSI Cell Placement Techniques. *Computing Surveys*, 23:143–220, 1991.
- [106] G.B. Sorkin. Efficient Simulated Annealing on Fractal Energy Landscapes. *Algorithmica*, 6:367–418, 1991.
- [107] J. Stander and B.W. Silverman. Temperature Schedules for Simulated Annealing. *Statistics and Computing*, 4:21–32, 1994.
- [108] K. Steinhöfel, A. Albrecht, and C.K. Wong. On Various Cooling Schedules for Simulated Annealing Applied to the Job Shop Problem. *Lecture Notes in Computer Science*, 1518:260–279, 1998.
- [109] P.N. Strenski and S. Kirkpatrick. Analysis of Finite Length Annealing Schedules. *Algorithmica*, 6:346–366, 1991.
- [110] H. Szu and R. Hartley. Fast Simulated Annealing. *Physics Letters A*, 122:157–162, 1987.
- [111] C. Tsallis and D.A. Stariolo. Generalized Simulated Annealing. *Physica A*, 233:395–406, 1996.
- [112] K. Tsuchiya, T. Nishiyama, and K. Tsujita. A Deterministic Annealing Algorithm for a Combinatorial Optimization Problem by the Use of Replicator Equations. In: *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, vol. 1, pages 256–261, 1999.
- [113] D. Tuyttens, J. Teghem, Ph. Fortemps, and K. van Nunen. Performance of the MOSA Method for the Bicriteria Assignment Problem. *Journal of Heuristics* 6:295–310, 2000.
- [114] R.J.M. Vaessens, E.H.L. Aarts, and J.K. Lenstra. A Local Search Template. In: *Parallel Problem Solving from Nature 2*, R. Männer and B. Manderick, editors, pages 65–74, North-Holland, 1992.
- [115] R.J.M. Vaessens, E.H.L. Aarts, and J.K. Lenstra. Job Shop Scheduling by Local Search. *INFORMS Journal on Computing*, 8:302–317, 1996.
- [116] M.G.A. Verhoeven and E.H.L. Aarts. Parallel Local Search Techniques. *Journal of Heuristics*, 1:43–65, 1996.

- [117] R.V.V. Vidal. Applied Simulated Annealing. *Lecture Notes in Economics and Mathematical Systems*, 396:276–290, 1993.
- [118] B.W. Wah and T. Wang. Simulated Annealing with Asymptotic Convergence for Nonlinear Constrained Global Optimization. *Lecture Notes in Computer Science*, 1713:461–475, 1999.
- [119] L. Wang and K. Smith. On Chaotic Simulated Annealing. *IEEE Transactions on Neural Networks*, 9:716–718, 1998.
- [120] T.-Y. Wang and K.-B. Wu. A Parameter Set Design Procedure for the Simulated Annealing Algorithm under the Computational Time Constraint. *Computers and Operations Research*, 26:665–678, 1999.
- [121] R.J. Willis and B.J. Terrill. Scheduling the Australian State Cricket Season using Simulated Annealing. *Journal of the Operational Research Society*, 45:276–280, 1994.
- [122] D.F. Wong, H.W. Leong, and C.L. Liu. *Simulated Annealing for VLSI Design*. Kluwer, 1988.
- [123] Y. Xiang and X.G. Gong. Efficiency of Generalized Simulated Annealing. *Physical Review E*, 62:4473–4476, 2000.
- [124] R.L. Yang. Convergence of the Simulated Annealing Algorithm for Continuous Global Optimization. *Journal of Optimization Theory and Application*, 104:691–716, 2000.
- [125] X. Yao. Simulated Annealing with Extended Neighbourhood. *International Journal of Computer Mathematics*, 40:169–189, 1991.
- [126] X. Yao. Optimization by Genetic Annealing. In: *Proceedings of the 2nd Australian Conference on Neural Networks*, pages 94–97, Sydney, 1991.
- [127] X. Yao. A New Simulated Annealing Algorithm. *International Journal of Computer Mathematics*, 56:161–168, 1995.
- [128] G. Yin. Convergence of a Global Stochastic Optimization Algorithm with Partial Step Size Restarting. *Advances in Applied Probability*, 32:480–498, 2000.

2 REACTIVE TABU SEARCH WITH PATH-RELINKING FOR THE STEINER PROBLEM IN GRAPHS

Marcelo P. Bastos and Celso C. Ribeiro

Department of Computer Science
Catholic University of Rio de Janeiro
R. Marquês de São Vicente 225
Rio de Janeiro 22453-900, Brazil
`{mbastos,celso}@inf.puc-rio.br`

Abstract: Given an undirected graph with weights associated with its edges, the Steiner tree problem consists in finding a minimum weight subgraph spanning a given subset of nodes (terminals) of the original graph. We describe a reactive tabu search with path-relinking algorithm for the Steiner problem in graphs, based on the extension of a previously developed tabu search algorithm using a neighborhood defined by insertions and eliminations of Steiner nodes. Computational experiments on benchmark problems are reported, comparing the reactive tabu search with other metaheuristic implementations. The reactive tabu search algorithm outperforms other algorithms, obtaining better or comparably good solutions. We also describe a robust parallel implementation based on an independent-thread multiple-walk strategy and report improved computational results on a 32-processor cluster running under Linux.

2.1 INTRODUCTION

Let $G = (V, E)$ be a connected undirected graph, where V is the set of nodes and E denotes the set of edges. Given a non-negative weight function $w : E \rightarrow \mathbb{R}_+$ associated with its edges and a subset $X \subseteq V$ of terminal nodes, the Steiner problem $\text{SPG}(V, E, w, X)$ consists in finding a minimum weighted connected subtree of G spanning all terminal nodes in X . The solution of $\text{SPG}(V, E, w, X)$ is a Steiner minimum tree. The non-terminal nodes that end up in the Steiner minimum tree are called Steiner nodes.

The Steiner problem in graphs is a classic combinatorial optimization problem, see e.g. Hwang, Richards, and Winter [15], Maculan [20], and Winter [32]. Karp [16] showed that its decision version is NP-complete in the general case. Applications can be found in many areas, such as telecommunication network design, VLSI design, and computational biology, among others. Heuristics are reviewed e.g. by Duin and Voss [8], Hwang, Richards, and Winter [15] and Voss [31].

Constructive methods have been proposed e.g. by Choukmane [4], Kou et al. [19], and Takahashi and Matsuyama [28]. Improvement heuristics based on the insertion of Steiner nodes have been proposed by Minoux [25] and Voss [30]. Among the most efficient approximate algorithms, we find implementations of metaheuristics such as genetic algorithms (Esbensen [9] and Kapsalis, Rayward-Smith, and Smith [17]), tabu search (Gendreau, Laroche, and Sansó [10], Ribeiro and Souza [26]), GRASP (Martins, Ribeiro, and Souza [21] and Martins et al. [22, 23]), and simulated annealing (Dowsland [6]). Gendreau, Laroche, and Sansó [10] proposed a tabu search heuristic for the Steiner problem, based on a neighborhood defined by Steiner node insertions and eliminations. More recently, Ribeiro and Souza [26] proposed another, improved tabu search heuristic, based on the same type of neighborhood. Move value estimations, eliminations tests, and neighborhood reduction techniques are used to speedup the local search and lead to a very fast implementation, which produces high quality solutions.

In this paper, we present a reactive tabu search algorithm based on the same neighborhood definition and acceleration techniques described in Ribeiro and Souza [26], which improves the results reported in the latter. A greedy randomized algorithm is used for diversification, while path-relinking is used for intensification. In the next section, we summarize the neighborhood structure and the associated local search procedure. Each solution is characterized by its set of Steiner nodes, and two types of moves are defined: insertion moves and elimination moves. Special mechanisms are used for the evaluation of insertion and elimination moves, in order to speedup the local search. Section 2.3 describes the reactive tabu search procedure in details, including the tabu tenure adjustment mechanism. The construction of initial solutions and the diversification procedure are also described in this section. The path-relinking procedure and the construction of the pool of elite solutions are described in Section 2.4. Computational experiments on benchmark problems are reported in Section 2.5, together with some comparisons with other literature algorithms. A parallel implementation based on an independent-thread multiple-walk strategy is described in Section 2.6 and computational results on a 32-processor cluster running under Linux are reported. Concluding remarks are made in Section 2.7.

2.2 NEIGHBORHOOD STRUCTURE AND LOCAL SEARCH

Any set $S \subseteq V \setminus X$ of Steiner nodes can be associated with a Steiner tree $T(S)$, corresponding to a minimum spanning tree of the graph induced in G by $S \cup X$.

Let S^* be the set of Steiner nodes in the optimal solution of $\text{SPG}(V, E, w, X)$. Then, the Steiner minimum tree $T(S^*)$ is also a minimum spanning tree of the graph induced in G by the node set $S^* \cup X$. Moreover, let $w(T(S))$ be the weight of $T(S)$, given by the sum of its edge weights. Accordingly, the search for a Steiner minimum tree amounts to the search for an optimal set S^* of Steiner nodes, such that $w(T(S^*)) = \min_{S \subseteq V \setminus X} \{w(T(S))\}$.

Let S be a set of Steiner nodes and $T(S)$ be its associated Steiner tree. The neighbors of solution $T(S)$ are defined by all sets of Steiner nodes which can be obtained either by adding to S a new non terminal node (insertion moves) or by eliminating from S one of its Steiner nodes (elimination moves). For any non terminal node $s \notin S$ (resp. $s \in S$), we denote by $T(S \cup \{s\})$ (resp. $T(S \setminus \{s\})$) a minimum spanning tree of the graph induced in G by $(S \cup \{s\}) \cup X$ (resp. $(S \setminus \{s\}) \cup X$), as the result of the insertion (resp. elimination) of node s into (resp. from) S . The move value associated with the insertion (resp. elimination) of a non terminal node $s \notin S$ (resp. $s \in S$) is given by $\delta(s) = w(T(S \cup \{s\})) - w(T(S))$ (resp. $\delta(s) = w(T(S \setminus \{s\})) - w(T(S))$).

Ribeiro and Souza [26] proposed a very fast linear time scheme to compute estimations $\hat{\delta}(\cdot)$ which are upper bounds to the value of insertion moves, instead of performing exact move evaluations. The same scheme is used in our reactive tabu search approach. Given the current solution characterized by its set S of Steiner nodes, let $s' = \operatorname{argmin}_{s \in V \setminus (X \cup S)} \{\hat{\delta}(s) : s \text{ is not tabu}\}$ be the candidate node to insertion with the best estimation among all non-tabu nodes. The only move value which is exactly computed is $\delta(s') = w(T(S \cup \{s'\})) - w(T(S))$, associated with the insertion of s' into S .

The classical aspiration criterion is used: a tabu node may be selected for insertion whenever it improves the weight of the current best solution. For each non terminal tabu node $s \notin S$, the evaluation of $\delta(s) = w(T(S \cup \{s\})) - w(T(S))$ involves the computation of a minimum spanning tree of the graph induced in G by $(S \cup \{s\}) \cup X$.

In case the best insertion move improves the current solution, it is selected and made effective by the local search procedure. Elimination moves are evaluated and compared with the best insertion move only in case the latter is non-improving. This strategy strongly reduces the computation times involved in the neighborhood search, as far as the evaluation of elimination moves is the most time consuming phase of this procedure. A linear time neighborhood reduction strategy, also proposed in [26] and based on the computation of lower bounds $\underline{\delta}(s)$ to the exact move values, is used to speedup the search for improving elimination moves.

A move is considered to be forbidden whenever the current tabu tenure plus the index of the most recent iteration in which the associated node was involved in a move is greater than the current iteration counter. The local search procedure, already used by Ribeiro and Souza [26], applied to the current solution at each iteration of the reactive tabu search algorithm is summarized in Figure 2.1.

```

procedure Local_Search( $V, E, w, X, S, S^*$ )
   $S$ : Steiner nodes in the current solution
   $T(S)$ : current solution
   $S^*$ : current best set of Steiner nodes

  /* Determination of the best insertion among nontabu nodes */
  1    $s' \leftarrow \operatorname{argmin}_{s \in V \setminus (X \cup S)} \{\hat{\delta}(s) : s \text{ is not tabu}\}$ 

  /* Determination of the best insertion among tabu nodes */
  2    $s'' \leftarrow \operatorname{argmin}_{s \in V \setminus (X \cup S)} \{w(T(S \cup s)) : s \text{ is tabu}\}$ 

  /* Determination of the best node to be inserted */
  3   if  $w(T(S \cup \{s''\})) < w(T(S^*))$  and  $w(T(S \cup \{s'\})) < w(T(S'))$ 
  4   then return  $s''$ 
  5   else if  $w(T(S \cup \{s'\})) < w(T(S))$ 
  6   then return  $s'$ 
  7    $\text{best\_neighbor\_value} \leftarrow w(T(S \cup \{s'\}))$ 

  /* Evaluation of Steiner nodes eliminations */
  8   for each  $s \in S$  : lower bound  $\underline{\delta}(s) < \min\{0, \text{best\_neighbor\_value}\}$  do
  9     if  $(w(T(S \setminus \{s\})) < \text{best\_neighbor\_value}$  and  $s$  is not tabu) or
         $(w(T(S \setminus \{s\})) < w(T(S^*))$  and  $s$  is tabu)
  10    then  $s' \leftarrow s$ ;  $\text{best\_neighbor\_value} \leftarrow w(T(S \setminus \{s\}))$ 
  11  end-for

  /* Best among all eliminations and non-improving insertions */
  12 return  $s'$ 
end Local_Search

```

Figure 2.1 Pseudo-code of the local search procedure.

2.3 REACTIVE TABU SEARCH

Tabu search is a metaheuristic for combinatorial optimization problems, which makes use of dynamic memory structures to guide a subordinate heuristic even in the absence of improving moves [11, 12, 14]. A fundamental parameter in tabu search implementations is the *tabu tenure*, which determines the number of iterations along which a move is considered as forbidden. Too small tabu tenure values may not be effective to avoid cycling, while too large values may lead to the premature stop of the search at low quality solutions.

The *reactive tabu search* technique has been proposed by Battiti and Tecchioro [1]. Besides handling the short term memory, which stores forbidden move attributes, reactive tabu search also keeps track of the whole set of visited solutions throughout the search. Its most characteristic feature is the possibility of

checking for cycling and, in consequence, taking decisions which allow dynamic changes in the value of the tabu tenure.

There is a counter $rep(S)$ associated with each solution S , representing the number of times the latter was visited. A hashing table is used to speedup the access to solutions previously visited and stored, which is done in almost constant time in the average. The tabu tenure is multiplied by a factor larger than one (we have used 1.3 in our implementation) whenever the move selected by local search at some iteration leads to a solution previously visited. The tabu tenure is divided by this same factor whenever all moves turn out to be forbidden. The tabu tenure is also gradually and periodically reduced, in order to make the search less restrictive in regions less subjected to cycling (in our implementation, the tabu tenure is divided by 1.1 after each group of ten iterations).

Another advantage of keeping track of all visited solutions and using the reactive scheme, is to avoid that the search be confined within small regions of the solution space. To overcome this ineffective behavior, we enforce a diversification step whenever the same solution is visited more than a certain number of times.

Initial solutions are computed by a randomized version of the shortest path heuristic (SPH) of Takahashi and Matsuyama [28], using any randomly selected node $s_0 \in V$ as the root. At each iteration k of the shortest path heuristic, let S_k be the set of terminal nodes already spanned by the current tree SPH_k , with $SPH_0 = \{s_0\}$. Then, let $s_k \in X \setminus S_k$ be a non-spanned terminal node randomly selected from this list. Then, set SPH_{k+1} by appending to SPH_k all nodes in the shortest path from s_k to SPH_k . The updated set S_{k+1} of already spanned terminal nodes is obtained by inserting into S_k the terminal nodes in the shortest path from s_k to SPH_k . The initial solution heuristic stops as soon as all terminal nodes have been spanned.

Diversification steps are performed in two situations. First, after each sequence of $max_iter_for_diversification$ iterations without improvement in the best solution found since the last diversification step. Second, whenever the same solution is visited by the search a certain number of $max_repetitions$ times (in our implementation, whenever the counter $rep(S)$ attains the value eight for some solution S). The same procedure described above for the construction of initial solutions is used to compute a new solution to restart the search at a diversification step.

The pseudo-code of the reactive tabu search algorithm is given in Figure 2.2. Initializations are performed in lines 1 to 3. The loop from lines 4 to 44 is performed until $max_iter_without_improvement$ have been performed without improvement in the best solution found. Lines 5 to 17 implement the reactive mechanism of the reactive tabu search algorithm. The `Local_Search` procedure described in Section 2.2 is applied in line 18 to find the node \bar{s} associated with the best move from the current solution S . In case all moves are forbidden, the tabu tenure reduction mechanism is activated in lines 19-20 and a new iteration starts. The best neighbor solution \bar{S} is constructed in lines 21-24, depending on

whether the best move was a node insertion or a node elimination. In line 25, we update the iteration counter associated with the most recent iteration in which node \bar{s} was involved. The best solution found and the related iteration counter are updated in lines 26-29. The best solution found since the last diversification and the related iteration counter are updated in lines 30-33. The new solution found is evaluated for insertion in the pool of elite solutions (see Section 2.4) in line 34. In case $\text{max_iter_for_diversification}$ iterations without improvement in the best solution found since the last diversification have not yet been performed, the current solution is set to the best neighbor \bar{S} in lines 35-36. Otherwise, a diversification step is performed and a new initial

```

procedure RTS_for_SPG( $V, E, w, X$ )
   $S$ : Steiner nodes in the current solution  $T(S)$ 
   $L$ : list of non-terminal tabu nodes
   $R$ : list of already visited solutions
   $S^*$ : current best set of Steiner nodes
   $S^d$ : best set of Steiner nodes since last diversification

1 Compute an initial solution:  $S \leftarrow \text{Randomized\_SPH}(V, E, w, X)$ 
2  $S^*, S^d \leftarrow S; L, R \leftarrow \emptyset; \text{iterations} \leftarrow 1; \text{rep(.)} \leftarrow 0, \text{tabu\_tenure} \leftarrow 0$ 
3  $\text{iter\_without\_improvement}, \text{iter\_for\_diversification} \leftarrow 0$  /* stop criterion */
4 while ( $\text{iter\_without\_improvement} < \text{max\_iter\_without\_improvement}$ ) do
5   if  $S \in R$ 
6     then                                     /* solution already visited */
7        $\text{rep}(S) \leftarrow \text{rep}(S) + 1$ 
8        $\text{tabu\_tenure} \leftarrow 1.3 \times \text{tabu\_tenure}$            /* tabu tenure increase */
9       if  $\text{rep}(S) = \text{max\_repetitions}$ 
10      then                                         /* enforce a diversification move */
11         $S \leftarrow \text{Randomized\_SPH}(V, E, w, X); S^d \leftarrow S$ 
12         $\text{rep}(S) \leftarrow 0; \text{iter\_for\_diversification} \leftarrow 0$ 
13      end-if
14    else
15       $R \leftarrow R \cup \{S\}$ 
16       $\text{rep}(S) \leftarrow 1$ 
17    end-if
18     $\bar{s} \leftarrow \text{Local\_Search}(V, E, w, X, S, S^*)$           /* best neighbor */
19    if all moves are forbidden
20    then  $\text{tabu\_tenure} \leftarrow \max\{2, \text{tabu\_tenure}/1.3\}$ 
21    else if  $\bar{s} \notin S$ 
22      then  $\bar{S} \leftarrow S \cup \{\bar{s}\}$ 
23      else  $\bar{S} \leftarrow S \setminus \{\bar{s}\}$ 
24    end-if
```

Figure 2.2 Pseudo-code of the reactive tabu search procedure for the Steiner problem in graphs — Part I.

```

procedure RTS_for_SPG( $V, E, w, X$ )
(continued)
25      Update the iteration associated with the last move involving node  $\bar{s}$ 
26      if  $w(T(\bar{S})) < w(T(S^*))$  /* best solution found? */
27      then  $S^* \leftarrow \bar{S}$ ;  $iter\_without\_improvement \leftarrow 0$ 
28      else  $iter\_without\_improvement \leftarrow iter\_without\_improvement + 1$ 
29      end-if
30      if  $w(T(\bar{S})) < w(T(S^d))$  /* best solution since last diversification? */
31      then  $S^d \leftarrow \bar{S}$ ;  $iter\_for\_diversification \leftarrow 0$ 
32      else  $iter\_for\_diversification \leftarrow iter\_for\_diversification + 1$ 
33      end-if
34      Evaluate  $\bar{S}$  for insertion in the pool of elite solutions
35      if  $iter\_for\_diversification < max\_iter\_for\_diversification$ 
36      then  $S \leftarrow \bar{S}$  /* move to best neighbor */
37      else /* diversification step */
38           $S \leftarrow \text{Randomized\_SPH}(V, E, w, X)$ 
39           $S^d \leftarrow S$ ;  $iter\_for\_diversification \leftarrow 0$ 
40      end-if
41      if  $iterations = 0 \bmod(10)$  /* tabu tenure reduction */
42      then  $tabu\_tenure \leftarrow \max\{2, tabu\_tenure/1.1\}$ 
43      end-if
44  end-if
45   $iterations \leftarrow iterations + 1$ 
46 end-while
47 return  $S^*$ 
end RTS_for_SPG

```

Figure 2.2 Pseudo-code of the reactive tabu search procedure for the Steiner problem in graphs — Part II.

solution is recomputed in lines 37-40 through the application of the randomized version of the shortest path heuristic SPH. Finally, the tabu tenure reduction mechanism is applied in lines 41-42, the iteration counter is updated in line 45, and a new iteration starts.

2.4 ELITE SOLUTIONS AND PATH RELINKING

The path-relinking approach generates new solutions by exploring trajectories that connect elite solutions [13, 14]. Starting from one or more of these solutions, paths in the solution space leading towards other elite solutions are generated and explored in the search for better solutions. This is accomplished by selecting moves that introduce attributes contained in the guiding solutions. Path-relinking may then be viewed as a strategy that seeks to incorporate attributes of high quality solutions (elite solutions), by favoring these attributes in the selected moves.

To implement an intensification strategy based on path-relinking for the Steiner problem in graphs, the reactive tabu search algorithm handles a pool with a fixed number of *pool_size* elite solutions. The pool is initialized with “null” solutions, whose cost is artificially made equal to infinity. Any solution visited during the search is considered as an elite solution and replaces the current worst in the pool if it satisfies two criteria:

- (1) *Distance*: Let $n(S)$ and $n(S_1 \Delta S_2)$ denote, respectively, the number of Steiner nodes in solution S and the number of Steiner nodes in the symmetric difference of solutions S_1 and S_2 . A solution S is eligible to be inserted into the pool of elite solutions if the ratio $n(S \Delta S^e)/\min\{n(S), n(S^e)\}$ is larger than a certain threshold *distance*, for every elite solution S^e currently in the pool. The rationale for this criterion is to ensure diversification among the solutions in the pool.
- (2) *Quality*: A solution S qualifies to be inserted into the pool with respect to this criterion if its weight is smaller than that of the worst solution currently in the pool.

For each pair of elite solutions (initial and guiding) in the pool, we first compute their symmetric difference, i.e., the set of all nodes which are Steiner nodes in one of them, but not in the other. This set defines the moves which should be applied to the initial solution until the guiding solution is attained. Starting from the first, we always perform the best (most decreasing or least increasing) remaining move still in this list, until the latter is attained. Whenever the best solution found along this trajectory improves the best solution found so far and satisfies the distance criterion above described, it replaces the worst solution currently in the pool and is submitted itself to the path-relinking procedure.

Path-relinking is applied as an intensification-based post-optimization strategy to the set of elite solutions identified by algorithm `RTS_for_SPG`.

2.5 COMPUTATIONAL RESULTS

In this section, we present numerical results obtained with the application of the reactive tabu search with path-relinking algorithm described in the previous sections. The algorithm was implemented in C++, using version 2.90.29 of the g++ compiler with the optimization flag set to `-O3`. Computational experiments have been performed on a Pentium II machine with a 400 MHz clock, running under the operating system Linux 2.0.36. The random number generator of Schrage [27] is used. The list of already visited solutions is stored in a hash table with 10000 entries. Function `hash` from library `libelf` (available by anonymous ftp from <ftp://sunsite.unc.edu/pub/Linux/libs>) version 0.6.4 was used for string hashing. The computational experiments have been performed on two sets of test problems:

- OR-Library instances: This set is formed by the 20 instances from each of the series C, D, and E of problems available from the OR-Library [2]. The original graphs have been submitted to reduction procedures. First,

the special distance test and the nearest special vertex test of Duin [7] are applied until no further reductions are possible. The special distance test eliminates edges which may not belong to an optimal solution, while the nearest special vertex test contracts edges which necessarily belong to every optimal solution. Next, all non-terminal nodes with degree one in the partially reduced graph are also eliminated. Finally, non-terminal nodes with degree two and its two adjacent edges are contracted into a single edge. All above steps are repeated, until no further reductions can be identified.

- Incidence instances: Those are the instances of type *incidence problems* described by Duin and Voss [8], with 80 (instances DV-80), 160 (instances DV-160), and 320 (instances DV-320) vertices. They were generated so as to make the reduction tests ineffective, as confirmed by Koch and Martin [18]. In each case, 20 classes of problems were generated, combining different densities of terminal nodes and edges. There are five problems in each class and a total of 300 instances, organized in series of 100 test problems characterized by their number of nodes $|V| = 80, 160$, and 320. The graphs in these series have not been submitted to the reduction procedures.

All numerical results have been obtained with the following parameters: tabu tenure initially set equal to 10, tabu tenure multiplied by 1.3 whenever the local search procedure leads to a previously visited solution, tabu tenure divided by 1.3 whenever all moves are tabu and divided by 1.1 after each sequence of 10 iterations, the size of the pool of elite solutions is set at 20, and the stopping criterion is defined by 3000 iterations without improvement in the best solution found.

To illustrate the behavior of the reactive tabu search with path-relinking, we investigate different settings for the following parameters using the OR-Library instances: diversification criterion, defined by the number of iterations without improvement in the best solution found since the last diversification (*max_iter_for_diversification*), and the threshold used as the acceptance criterion for a solution to be included in the pool of elite solutions (*distance*).

We first investigate the behavior of algorithm *RTS_for_SPG* with respect to the parameter involved in the diversification criterion. For each value of *max_iter_diversification* = 5, 10, 20, 30, 40, 60, and 100, the algorithm is applied 10 times to each OR-Library instance, with different seeds for the pseudo-random number generator. Computational results over the problems in each series and over all 60 test problems are depicted in Figure 2.3: the average percentual error with respect to the optimal value, the number of optimal solutions, and the average computation time in seconds. The overall results show that the smallest average errors are attained with *max_iter_for_diversification* = 20, while the number of optimal solutions and the average computation times do not seem to be too much affected by this parameter.

We next report results illustrating the influence of the threshold *distance* used as the acceptance criterion for a solution to be included in the pool of

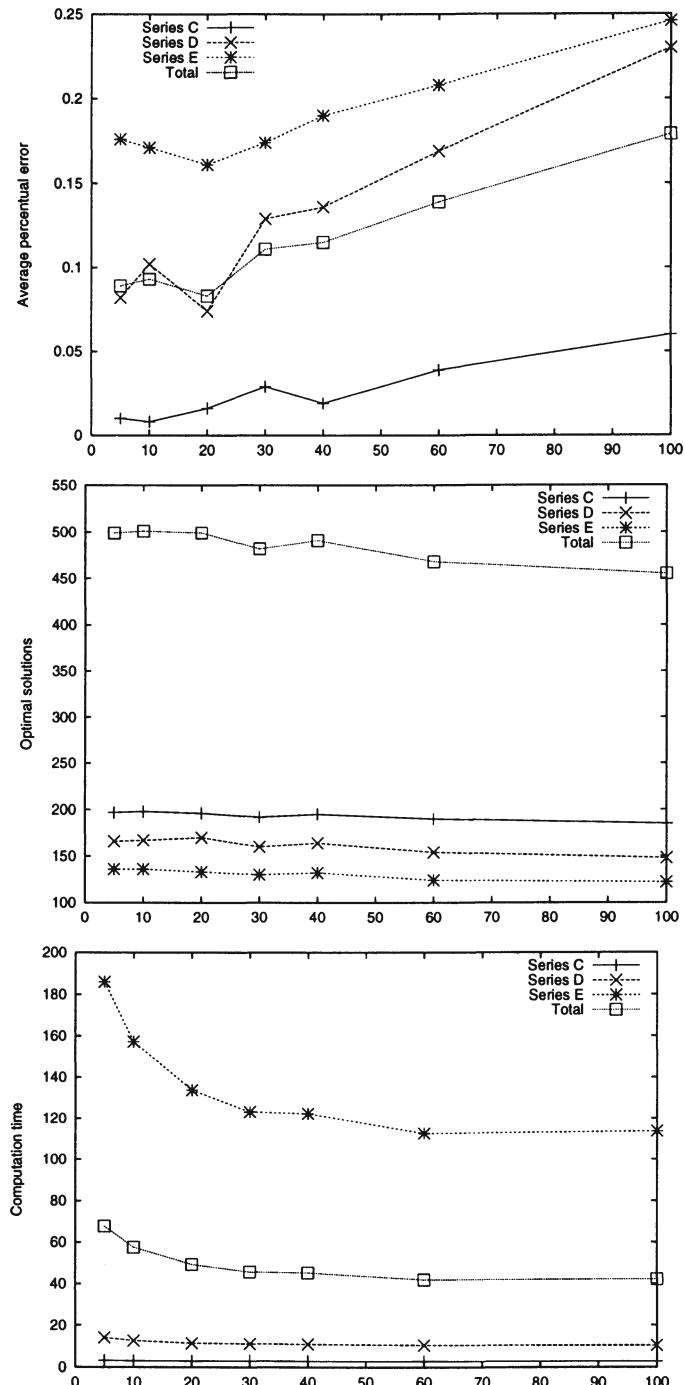


Figure 2.3 Variation of the diversification parameter *max_iter_for_diversification*.

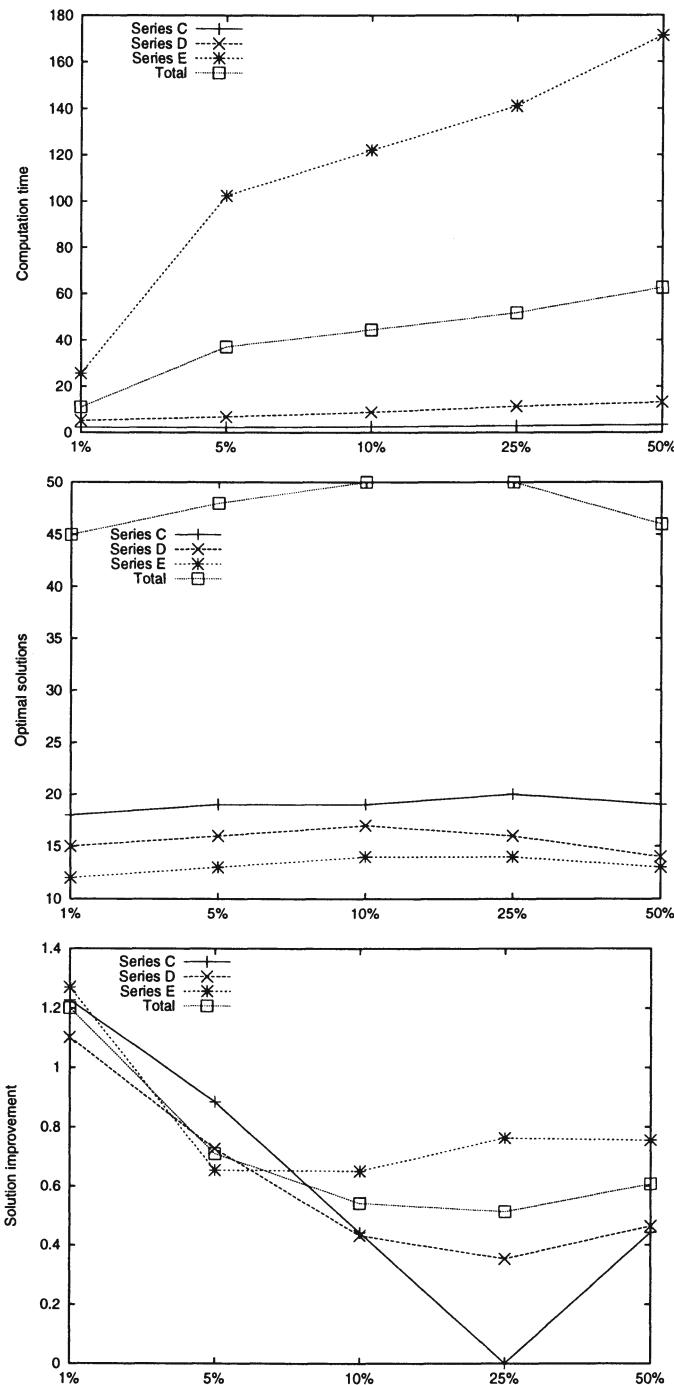


Figure 2.4 Variation of the threshold distance.

elite solutions. For each instance, we ran the reactive tabu search with path-relinking once for each value of the parameter $distance = 1\%, 5\%, 10\%, 25\%$, and 50% . Computational results over the problems in each series and over all 60 test problems are depicted in Figure 2.4: the number of optimal solutions, the average computation time in seconds, and the improvement in solution value obtained by path-relinking with respect to the solution found without it (computed over the instances for which the reactive tabu search algorithm itself has not found the optimal solution). Computation times increase with the threshold $distance$, since the length of the relinking paths increase with the diversity of the elite solutions. Best overall results in terms of the average solution improvement were obtained with $distance = 25\%$.

To further illustrate the behavior of the reactive tabu search algorithm, we display in Figure 2.5 the evolution of the *tabu_tenure* parameter along the iterations of the execution of problem E12. We notice that the mechanisms for tabu tenure increasing and reduction are often activated, keeping its value within reasonable limits most of the time.

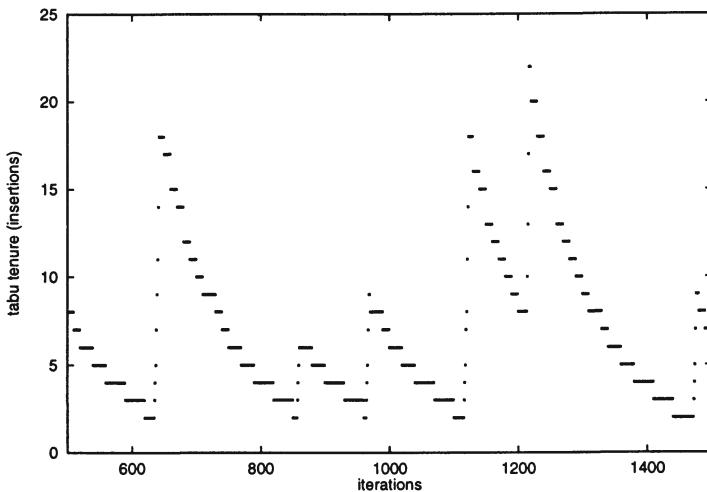


Figure 2.5 Variation of the tabu tenure along the iterations.

We summarize in Table 2.1 the results obtained by the reactive tabu search algorithm (with and without path-relinking) for the OR-Library instances with the above parameter settings. The algorithm was run ten times for each of the 20 instances in series C, D, and E, with different seeds for the pseudo-random number generator, in a total of 200 runs for each series. For each series C, D, and E, we give the average and the maximum percentual errors with respect to the optimal value, the average computation time in seconds, the number of runs for which the optimal solution was found, and the number of runs for which the percentual error was less than 1% and less than 5%. Overall results are also presented.

Reactive tabu search without path-relinking						
Series	Avg.err. (%)	Max.err. (%)	Opt.	Time	Err. < 1%	Err. < 5%
C	0.13	2.65	177	2.00	188	200
D	0.36	4.04	142	5.33	178	200
E	0.59	6.21	116	21.33	165	190
Total	0.36	6.21	435	9.55	531	590
Reactive tabu search with path-relinking						
Series	Avg.err. (%)	Max.err. (%)	Opt.	Time	Err. < 1%	Err. < 5%
C	0.01	0.88	198	2.96	200	200
D	0.10	3.45	167	12.53	197	200
E	0.17	1.77	136	157.18	186	200
Total	0.09	3.45	501	57.55	583	600

Table 2.1 Run statistics for tabu search with path-relinking for OR-Library instances.

Series	TS [26]	F-Tabu [10]	RTS	RTS-PR
C	0.26	0.02	0.13	0.01
D	0.71	0.11	0.36	0.10
E	0.82	0.31	0.59	0.17
Total	0.60	0.15	0.36	0.09

Table 2.2 Average relative errors with different tabu search strategies.

Globally, the reactive tabu search algorithm without path-relinking found the optimal solutions in 435 out of the 600 runs. Path-relinking lead to optimal solutions in 66 additional runs, although at the cost of a strong augmentation in computation times. Path-relinking also reduced the average relative error with respect to the optimal value from 0.36% to 0.09%. To further illustrate the effectiveness of the reactive tabu search algorithm on the OR-Library instances, we give in Table 2.2 the results obtained by different tabu search algorithms for the OR-Library instances. Each of these algorithms was run once for each problem. For each series, the second column (TS) in this table displays the average relative error observed with the tabu search algorithm of Ribeiro and Souza [26]. The next column (F-Tabu) displays the same information for the tabu search algorithm of Gendreau, Larochelle, and Sansó [10]. The results obtained with the tabu search algorithm without and with path-relinking are given in the columns denoted respectively with RTS and RTS-PR. Reactive tabu search with path-relinking strongly outperforms the other tabu search strategies. Computation times cannot be directly compared, since the results have been obtained in different machines.

Next, we summarize in Table 2.3 the results obtained by the reactive tabu search algorithm (with and without path-relinking) for the incidence instances, with the same parameter settings. Proven optimal solutions are available for all problems in series DV-80 and DV-160, but for only 91 out of the 100 instances in series DV-320. The algorithm was run once for each of the 100 instances in

Reactive tabu search without path-relinking				
Series	Avg. error (%)	Max. error (%)	Optima	Time
DV-80	0.02	0.66	96	5.8
DV-160	0.19	3.21	73	28.1
DV-320	≤ 0.48	≤ 5.56	≥ 56	162.6
Total	≤ 0.23	≤ 5.56	≥ 225	65.5
Reactive tabu search with path-relinking				
Series	Avg. error (%)	Max. error (%)	Optima	Time
DV-80	0.01	0.66	99	6.2
DV-160	0.12	3.21	81	29.9
DV-320	≤ 0.34	≤ 4.45	≥ 64	168.1
Total	≤ 0.16	≤ 4.45	≥ 244	68.1
Tabu search [26]				
Series	Avg. error (%)	Max. error (%)	Optima	Time
DV-80	0.08	1.7	89	3.3
DV-160	0.35	2.9	58	12.2
DV-320	≤ 0.89	≤ 6.3	≥ 39	50.5
Total	≤ 0.44	≤ 6.3	≥ 186	22.0
SVERTEX+REBUILD [8] - Version 1				
Series	Avg. error (%)	Max. error (%)	Optima	Time
DV-80	1.03	11.8	not available	0.4
DV-160	0.98	6.2	not available	2.7
DV-320	≤ 1.20	≤ 6.9	not available	14.8
Total	≤ 1.07	≤ 11.8	not available	6.0
SVERTEX+REBUILD [8] - Version 2				
Series	Avg. error (%)	Max. error (%)	Optima	Time
DV-80	1.24	11.8	not available	0.2
DV-160	1.26	5.2	not available	0.5
DV-320	≤ 1.90	≤ 7.4	not available	1.6
Total	≤ 1.47	≤ 11.8	not available	0.8

Table 2.3 Run statistics for the incidence instances.

series DV-80, DV-160, and DV-320. For each series, we give the average and the maximum percentual errors with respect to the optimal value (with respect to a lower bound, in the case of the instances in series DV-320 for which the optimal value is not known), the number of instances for which the optimal solution was found, and the average computation time in seconds. Overall results are also presented. We also report the same statistics for the tabu search algorithm of Ribeiro and Souza [26] (obtained on the same Sun UltraSPARC 1 machine) and for procedure SVERTEX+REBUILD from Duin and Voss [8] (obtained on a Pentium with a 90MHz clock). Results from the latter are presented in two versions. The first version corresponds to results extracted from Table II of [8], which have been obtained through the exact computation of the minimum distance matrix. For the second version, we reproduce the results from Table III of [8], using the approximate computation of the distance matrix. Computation times are quite smaller in the second case, due to the smaller complexity of the initialization steps. Processing times in the first case were obtained from [8],

summing up the times in column *sec1* of REBUILD with those in column *sec2* of SVERTEX+REBUILD in Table III.

The reactive tabu search with path-relinking found at least 244 optimal solutions out of the 300 instances in series DV-80, DV-160, and DV-320, with the average relative error with respect to the optimal value being not greater than 0.16%. Once again, path-relinking improved the results obtained by the reactive tabu search algorithm, but this time at a very small additional cost in terms of computation time. The reactive tabu search with path-relinking obtained much better results than both versions of SVERTEX+REBUILD, in terms of both the average and the maximum relative errors with respect to the optimal value. It also improved the solutions found by the tabu search algorithm, finding 58 additional optimal solutions and reducing the average relative error from 0.44% to 0.16%.

2.6 PARALLELIZATION

In this section we describe a parallel implementation of the reactive tabu search algorithm with pattern relinking described in Sections 2.3 and 2.4, with the goal of improving the solutions found by the latter. According with the classification discussed in [5, 29], our parallelization strategy follows an independent-thread multiple-walk strategy, in which each slave processor explores a different variant of the reactive tabu search algorithm described in Section 2.3. The master processor keeps a pool of elite solutions, which is used for path-relinking once all processors have finished their searches. We denote by p the total number of processors involved in the parallelization. The two phases of the parallel implementation are described next.

Independent-thread multiple-walk reactive tabu search. In this first phase, each of the $(p - 2)$ slave processors independently performs the reactive tabu search algorithm. Each processor uses a different parameter setting. Each slave constructs its own initial solution, using the randomized shortest-path heuristic of Takahashi and Matsuyama [28]. Whenever a slave has to perform a diversification step, it requires a new solution to the constructor processor. The latter continuously runs the randomized version of the shortest-path heuristic, keeping a list of solutions which are sent to the slaves upon request. The master processor keeps track of the total number of iterations performed by the slaves, controls the application of the stopping criterion (total number of iterations performed by all processors), and maintains a centralized pool of elite solutions. The pool of elite solutions is formed by locally optimal solutions found by the slaves, i.e. solutions whose best neighbor identified by the local search procedure described in Section 2.2 is a non-improving one. Locally optimal solutions found by the slaves are included in the pool of elite solutions only if they satisfy both criteria described in Section 2.4. We note that the slave processors asynchronously communicate only with the master and with the constructor processors, but not among themselves.

Series	OR-Library instances						Avg. elapsed time	
	Reactive tabu search phase			Path-relinking phase				
	Average err. (%)	Maximum err. (%)	Opt. sols.	Average err. (%)	Maximum err. (%)	Opt. sols.		
C	0.00	0.00	20	0.00	0.00	20	7.35	
D	0.24	3.14	18	0.04	0.45	18	14.05	
E	0.44	4.61	13	0.14	1.42	14	55.65	
Total	0.23	4.61	51	0.06	1.42	52	25.68	

Series	Incidence instances						Avg. elapsed time	
	Reactive tabu search phase			Path-relinking phase				
	Average err. (%)	Maximum err. (%)	Opt. sols.	Average err. (%)	Maximum err. (%)	Opt. sols.		
DV-80	0.00	0.14	99	0.00	0.00	100	6.85	
DV-160	0.02	0.63	90	0.01	0.33	96	10.14	
DV-320	≤ 0.30	≤ 4.64	≥ 75	≤ 0.21	≤ 3.91	≥ 82	42.51	
Total	≤ 0.11	≤ 4.64	≥ 264	≤ 0.07	≤ 3.91	≥ 278	18.83	

Table 2.4 Run statistics with the parallel implementation.

Distributed path-relinking. In this second phase, $(p - 1)$ slave processors perform the path-relinking algorithm described in Section 2.4 over all pairs of elite solutions kept in the centralized pool by the master processor. The latter distributes pairs of elite solutions to the slaves upon their request.

This parallelization strategy was also implemented in C++, using version 2.90.29 of the g++ compiler and the LAM implementation of the Message Passing Interface (MPI) standard [24] for communication. The computational experiments have been performed on a cluster of 32 Pentium II-400 processors, each having 32 Mbytes of RAM and running under the operating system Linux 2.0.36.

In our implementation, we have set the total number of tabu search iterations at 90,000 and the size of the pool of elite solutions at 100. The 30 slave processors running the sequential reactive tabu search algorithm use different combinations of the following parameters: tabu tenure increase factor equal to 1.2, 1.3, 1.6, and 1.9, maximum number of repetitions of the same solution before diversification equal to 8 and 11, number of iterations without improvement before diversification ranging from 10 to 80. The other parameter values are those set and used along the computational experiments reported in Section 2.5. Computational results obtained for OR-Library and incidence instances are reported in Table 2.4. For each series of test instances, we give the following statistics at the end of each algorithm phase: average and maximum percentual errors with respect to the optimal value, and the number of instances for each an optimal solution was found. We also inform the average elapsed time in seconds including the two phases. We also give a summary of the results obtained over all instances of each type.

The parallel implementation found optimal solutions for all OR-Library instances in series C, and for respectively 18 and 14 out of the 20 instances in each of series D and E. The maximum and the average relative errors with respect

to the optimum among all instances of this type were respectively 0.14% and 0.06%. For the incidence problems, this strategy found optimal solutions for all instances with 80 nodes, and respectively 96 and 82 optimal solutions out of each set of 100 instances with 160 and 320 nodes. The maximum and the average relative errors with respect to the optimum among all instances of this type were respectively 3.91% and 0.07%.

The comparison of the results obtained by the parallel implementation with those reported for the original sequential algorithm in Tables 2.1 and 2.3 shows that the former found improved results for all problem series and sizes. Such improvements can be better observed for the largest instances. For the incidence problems with 160 nodes, the parallel algorithm found 15 additional optimal solutions than the sequential one and reduced the average relative error from 0.12% to 0.01%. For those with 320 nodes, the parallel implementation found 18 additional optimal solutions and reduced the average error from 0.34% to 0.21%. As for the sequential algorithm, we also note that path-relinking strongly contributed to improve the solutions found by tabu search.

2.7 CONCLUSIONS

We presented in this work a reactive tabu search with path-relinking algorithm for the Steiner problem in graphs, based on the extension of a previously developed tabu search algorithm using a neighborhood defined by insertions and eliminations of Steiner nodes. Path-relinking is based on the use of a pool of elite solutions visited during the search. A randomization strategy is also used to further increase the diversification of the initial solutions constructed. We also described a parallel implementation of the reactive tabu search with path-relinking algorithm, based on an independent-thread multiple-walk strategy.

Computational experiments on benchmark problems are reported, comparing the reactive tabu search with path-relinking with other heuristics from the literature. The reactive tabu search algorithm outperformed other approximate algorithms and metaheuristics, obtaining better solutions than recent implementations of path and vertex exchange algorithms [8], tabu search [10, 26], and GRASP [22, 23]. As already observed in [3] for the prize-collecting version of the Steiner problem in graphs, we note that path-relinking used as an intensification strategy strongly contributed to improve the solutions found by tabu search for all test problem types and sizes.

The parallel implementation found still better results than the sequential algorithm, in particular for the largest and more difficult test instances. As already noticed by Cung et al. [5], the main advantage of the parallel implementation is not only due to solution improvements, but mainly to its robustness, since high quality solutions are obtained without almost any effort in parameter tuning.

References

- [1] R. Battiti and G. Tecchiolli. The Reactive Tabu Search. *ORSA Journal on Computing*, 6:126–140, 1994.
- [2] J.E. Beasley. OR-Library: Distributing Test Problems by Electronic Mail. *Journal of the Operational Research Society*, 41:1069–1072, 1990.
- [3] S.A. Canuto, M.G.C. Resende, and C.C. Ribeiro. Local Search with Perturbations for the Prize-Collecting Steiner Tree Problem. To appear in: *Networks*.
- [4] E.-A. Choukmane. Une heuristique pour le problème de l’arbre de Steiner. *RAIRO Recherche Opérationnelle*, 12:207–212, 1978.
- [5] V.-D. Cung, S.L. Martins, C.C. Ribeiro, and C. Roucairol. Parallel Strategies for the Implementation of Metaheuristics. In: *Essays and Surveys in Metaheuristics*, C.C. Ribeiro and P. Hansen, editors, Kluwer, 2001 (this volume).
- [6] K.A. Dowsland. Hill-Climbing, Simulated Annealing and the Steiner Problem in Graphs. *Engineering Optimization*, 17:91–107, 1991.
- [7] C.W. Duin. Steiner’s Problem in Graphs: Approximation, Reduction, Variation. Doctorate Dissertation, Institute of Actuarial Science and Economics, University of Amsterdam, 1994.
- [8] C.W. Duin and S. Voss. Efficient Path and Vertex Exchange in Steiner Tree Algorithms. *Networks*, 29:89–105, 1997.
- [9] H. Esbensen. Computing Near-Optimal Solutions to the Steiner Problem in a Graph Using a Genetic Algorithm. *Networks*, 26:173–185, 1995.
- [10] M. Gendreau, J.-F. Larochelle, and B. Sansò. A Tabu Search Heuristic for the Steiner Tree Problem. *Networks*, 34:163–172, 1999.
- [11] F. Glover. Tabu Search — Part I. *ORSA Journal on Computing*, 1:190–206, 1989.
- [12] F. Glover. Tabu Search — Part II. *ORSA Journal on Computing*, 2:4–32, 1990.
- [13] F. Glover. Tabu Search and Adaptive Memory Programming — Advances, Applications and Challenges. In: *Interfaces in Computer Science and Operations Research*, R.S. Barr, R.V. Helgason, and J.L. Kennington, editors, pages 1–75, Kluwer, 1996.
- [14] F. Glover and M. Laguna. *Tabu Search*, Kluwer, 1997.
- [15] F.K. Hwang, D.S. Richards, and P. Winter. *The Steiner Tree Problem*. North-Holland, 1992.

- [16] R.M. Karp. Reducibility Among Combinatorial Problems. In: *Complexity of Computer Computations*, E. Miller and J.W. Thatcher, editors, pages 85–103, Plenum Press, 1972.
- [17] A. Kapsalis, V.J. Rayward-Smith, and G.D. Smith. Solving the Graphical Steiner Tree Problem Using Genetic Algorithms. *Journal of the Operational Research Society*, 44:397–406, 1993.
- [18] T. Koch and A. Martin. Solving Steiner Tree Problems in Graphs to Optimality. *Networks*, 32:207–232, 1998.
- [19] L.T. Kou, G. Markowsky, and L. Berman. A Fast Algorithm for Steiner Trees. *Acta Informatica*, 15:141–145, 1981.
- [20] N. Maculan. The Steiner Problem in Graphs. In: *Surveys in Combinatorial Optimization*, S. Martello, G. Laporte, M. Minoux, and C.C. Ribeiro, editors, *Annals of Discrete Mathematics*, 31:185–212, 1987.
- [21] S.L. Martins, C.C. Ribeiro, and M.C. Souza. A Parallel GRASP for the Steiner Problem in Graphs. *Lecture Notes in Computer Science*, 1457:285–297, 1998.
- [22] S.L. Martins, P. Pardalos, M.G. Resende, and C.C. Ribeiro. Greedy Randomized Adaptive Search Procedures for the Steiner Problem in Graphs. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 43:133–146, 1999.
- [23] S.L. Martins, P. Pardalos, M.G. Resende, and C.C. Ribeiro. A Parallel GRASP for the Steiner Tree Problem in Graphs Using a Hybrid Local Search Strategy. *Journal of Global Optimization*, 17:267–283, 2000.
- [24] Message Passing Interface Forum. MPI: A Message Passing Interface Standard, 1995.
- [25] M. Minoux. Efficient Greedy Heuristics for Steiner Tree Problems Using Reoptimization and Supermodularity. *INFOR*, 28:221–233, 1990.
- [26] C.C. Ribeiro and M.C. Souza. Tabu Search for the Steiner Problem in Graphs. *Networks*, 36:138–146, 2000.
- [27] L. Schrage. A More Portable Fortran Random Number Generator. *ACM Transactions on Mathematical Software*, 5:132–138, 1979.
- [28] H. Takahashi and A. Matsuyama. An Approximate Solution for the Steiner Problem in Graphs. *Mathematica Japonica*, 24:573–577, 1980.
- [29] M.G.A. Verhoeven and E.H.L. Aarts. Parallel Local Search. *Journal of Heuristics*, 1:43–65, 1995.
- [30] S. Voss. *Steiner-Probleme in Graphen*. Anton Hain, 1990.

- [31] S. Voss. Steiner's Problem in Graphs: Heuristic Methods. *Discrete Applied Mathematics*, 40:45–72, 1992.
- [32] P. Winter. Steiner Problem in Networks: A Survey. *Networks*, 17:129–167, 1987.

3 A GRASP FOR JOB SHOP SCHEDULING

S. Binato¹, W.J. Hery², D.M. Loewenstern³, and M.G.C. Resende⁴

¹CEPEL, Centro de Pesquisas de Energia Elétrica
P.O. Box 68007
Rio de Janeiro 21944-970, Brazil
silvio@cepel.br

²Bell Laboratories, Lucent Technologies
Whippany, NJ 07981 USA
hery@bell-labs.com

³Department of Computer Science, Rutgers University
Piscataway, NJ 08855 USA
loewenst@paul.rutgers.edu

⁴Information Sciences Research Center, AT&T Labs Research
Florham Park, NJ 07932 USA
mgcr@research.att.com

Abstract: In this paper, we describe a greedy randomized adaptive search procedure (GRASP) for the job shop scheduling problem (JSP). We incorporate to the conventional GRASP two new concepts: an intensification strategy and POP (Proximate Optimality Principle) in the construction phase. These two concepts were first proposed by Fleurent and Glover (1999) in the context of the quadratic assignment problem. Computational experience on a large set of standard test problems indicates that GRASP is a competitive algorithm for finding approximate solutions of the job shop scheduling problem.

3.1 INTRODUCTION

In the job shop scheduling problem (JSP), a finite set of jobs is processed on a finite set of machines. Each job is characterized by a fixed order of operations,

each of which is to be processed on a specific machine for a specified duration. Each machine can process at most one job at a time and once a job initiates processing on a given machine it must complete processing on that machine uninterrupted. A schedule is an assignment of operations to time slots on the machines. The makespan is the maximum completion time of the jobs. The objective of the JSP is to find a schedule that minimizes the makespan.

Formally, the JSP can be stated as follows. Given a set \mathcal{M} of machines (denote the size of \mathcal{M} by $|\mathcal{M}|$) and a set \mathcal{J} of jobs (denote the size of \mathcal{J} by $|\mathcal{J}|$), let $\sigma_1^j \prec \sigma_2^j \prec \dots \prec \sigma_{|\mathcal{M}|}^j$ be the ordered set of $|\mathcal{M}|$ operations of job j , where $\sigma_k^j \prec \sigma_{k+1}^j$ indicates that operation σ_{k+1}^j can only start processing after the completion of operation σ_k^j . Let \mathcal{O} be the set of operations. Each operation σ_k^j is defined by two parameters: \mathcal{M}_k^j is the machine on which σ_k^j is processed and $p_k^j = p(\sigma_k^j)$ is the processing time of operation σ_k^j . Defining $t(\sigma_k^j)$ to be the starting time of the k -th operation $\sigma_k^j \in \mathcal{O}$, a disjunctive programming formulation for the JSP is as follows:

$$\text{minimize } C_{\max}$$

$$\text{subject to: } C_{\max} \geq t(\sigma_k^j) + p(\sigma_k^j), \text{ for all } \sigma_k^j \in \mathcal{O},$$

$$t(\sigma_k^j) \geq t(\sigma_l^j) + p(\sigma_l^j), \text{ for all } \sigma_l^j \prec \sigma_k^j, \quad (3.1a)$$

$$t(\sigma_k^j) \geq t(\sigma_l^i) + p(\sigma_l^i) \vee t(\sigma_l^i) \geq t(\sigma_k^j) + p(\sigma_k^j), \quad (3.1b)$$

$$\text{for all } \sigma_l^i, \sigma_k^j \in \mathcal{O} \text{ such that } \mathcal{M}_{\sigma_l^i} = \mathcal{M}_{\sigma_k^j},$$

$$t(\sigma_k^j) \geq 0, \text{ for all } \sigma_k^j \in \mathcal{O},$$

where C_{\max} is the makespan to be minimized.

A feasible solution of the JSP can be built from a permutation of \mathcal{J} on each of the machines in \mathcal{M} , observing the precedence constraints, the restriction that a machine can process only one operation at a time, and requiring that once started, processing of an operation must be uninterrupted until its completion. Once the permutation of \mathcal{J} is given, evaluate its makespan C_{\max} can be done in $O(|\mathcal{J}| \cdot |\mathcal{M}|)$, see [38]. Each set of permutations has a corresponding schedule. Thus, the objective of the JSP is to find a set of permutations with the smallest makespan.

The JSP is NP-hard [28] and has continuously challenged computational researchers. Even instances with three machines and unit processing times, as well as instances with three jobs, are NP-hard. If preemption is allowed, the JSP remains NP-hard. Exact methods [3, 8, 9, 10, 23] have been successful in solving small instances, including the notorious 10×10 instance of Fisher and Thompson [20], proposed in 1963 and only solved twenty years later. Problems of dimension 15×15 are usually considered to be beyond the reach of exact methods. For such problems there is a need for good heuristics. Surveys of heuristic methods for the JSP are given in [33, 39]. These include dispatching rules reviewed in [22], the shifting bottleneck approach [1, 3], local search [30, 31, 39], simulated annealing [40, 30], tabu search [38, 32, 31], and genetic

algorithms [11]. A comprehensive survey of job shop scheduling techniques can be found in Jain and Meeran [25]. In this paper we present a greedy randomized adaptive search procedure (GRASP) for the job shop scheduling problem.

Our interest in studying a new heuristic for a problem for which many efficient heuristics have been proposed is motivated by several observations. GRASP has been applied with success to a number of scheduling problems [4, 14, 18, 26, 12, 13, 5, 17, 29, 35, 41, 36]. A natural question is whether it can find good solutions to the job shop scheduling problem. The new heuristic is not only another tool in the toolset of practitioners, but can also serve as a building block for researchers who want to investigate heuristic refinements, such as intensification strategies or parallelization schemes. GRASP has several characteristics which make it appealing for a heuristic designer. It is a meta-heuristic and therefore is built on well defined and understood concepts. It can usually be implemented quickly, since construction and local search algorithms for the job shop scheduling problem are readily available. Parameter tuning is minimal. Finally, because of the probability distribution of its running time to find a suboptimal target solution, GRASP can be implemented in parallel with optimal speed-up.

The remainder of the paper is organized as follows. In Section 3.2, we make a brief review of the building blocks of GRASP. Section 3.3 focuses on a basic GRASP for the job shop scheduling problem, describing a construction mechanism and a local search algorithm. In Sections 3.4 and 3.5, an intensification scheme that makes use of memory mechanisms and the use of the proximal optimality principle are incorporated to the basic GRASP. Computational results are reported in Section 3.6 and concluding remarks are made in Section 3.7.

3.2 A BRIEF REVIEW OF GRASP

In this paper, we apply the concepts of GRASP (greedy randomized adaptive search procedures) to the job shop scheduling problem. GRASP [15, 16] is an iterative process, where each GRASP iteration consists of two phases: construction and local search. The construction phase builds a feasible solution, whose neighborhood is explored by local search. The best solution over all GRASP iterations is returned as the result.

In the construction phase, a feasible solution is built, one element at a time. At each construction iteration, the next element to be added is determined by ordering all elements in a candidate list with respect to a greedy function that measures the (myopic) benefit of selecting each element. This list is called the RCL (restricted candidate list). The adaptive component of the heuristic arises from the fact that the benefits associated with every element are updated at each iteration of the construction phase to reflect the changes brought on by the selection of the previous elements. The probabilistic component of a GRASP is characterized by randomly choosing one of the best candidates in the RCL, but usually not the best one. This way of making the choice

allows for different solutions to be obtained at each GRASP iteration, while not necessarily jeopardizing the adaptive greedy component.

The solutions generated by a GRASP construction phase are not guaranteed to be locally optimal with respect to simple neighborhood definitions. Hence, it is almost always beneficial to apply a local search to attempt to improve each constructed solution. A local search algorithm works in an iterative fashion by successively replacing the current solution by a better solution from its neighborhood. It terminates when there is no better solution found in the neighborhood with respect to some cost function.

Figure 3.1 illustrates a generic GRASP implementation in pseudo-code. Input for GRASP includes parameters for setting the candidate list size and the maximum number of GRASP iterations, and the seed for the random number generator. The GRASP iterations are carried out in lines 1–5. Each GRASP iteration consists of the construction phase (line 2), the local search phase (line 3) and, if necessary, the incumbent solution update (line 4).

```

procedure GRASP(ListSize,MaxIter,RandomSeed)
1   do k = 1,..., MaxIter →
2       ConstructSolution(ListSize,RandomSeed);
3       LocalSearch(BestSolutionFound);
4       UpdateSolution(BestSolutionFound);
5   od;
6   return BestSolutionFound
end GRASP;
```

Figure 3.1 A generic GRASP pseudo-code.

GRASP has been applied successfully to numerous combinatorial optimization problems, including set covering, quadratic assignment, satisfiability, vehicle routing, location problems, maximum independent set, feedback vertex set, transmission network expansion planning, and graph planarization. For an annotated bibliography of GRASP, see Festa and Resende [19].

3.3 GRASP FOR JOB SHOP SCHEDULING

In this section, we specialize GRASP for the job shop scheduling problem (JSP). We first describe a basic construction scheme and then show a local search algorithm that uses disjunctive graphs.

3.3.1 Construction phase

The GRASP construction phase builds a feasible solution, one element at a time. In the JSP, we consider a single operation to be the building block of the construction phase. That is, we build a feasible schedule by scheduling individual operations, one at a time, until all operations are scheduled.

Recall that σ_k^j denotes the k -th operation of job j and is defined by the pair (\mathcal{M}_k^j, p_k^j) , where \mathcal{M}_k^j is the machine on which operation σ_k^j is performed and p_k^j is the processing time of operation σ_k^j .

While constructing a feasible schedule, not all operations can be selected at a given stage of the construction. An operation σ_k^j can only be scheduled if all prior operations of job j have already been scheduled. Therefore, at each construction phase iteration, at most $|\mathcal{J}|$ operations are candidates to be scheduled. Let this set of candidate operations be denoted by \mathcal{O}_c and the set of already scheduled operations by \mathcal{O}_s .

More than one greedy algorithm can be proposed for the JSP. One such algorithm consists of selecting the operation σ_k^j that results in the smallest increase in the makespan of the already scheduled jobs to schedule next. Let the adaptive greedy function $h(\sigma)$ denote the makespan resulting from the addition of operation σ to the already scheduled operations, i.e.

$$h(\sigma) = \mathcal{C}_{max} \text{ for } \mathcal{O} = \{\mathcal{O}_s \cup \sigma\}.$$

The greedy choice is to next schedule operation

$$\underline{\sigma} = \operatorname{argmin}(h(\sigma) \mid \sigma \in \mathcal{O}_c).$$

Defining

$$\bar{\sigma} = \operatorname{argmax}(h(\sigma) \mid \sigma \in \mathcal{O}_c),$$

$\underline{h} = h(\underline{\sigma})$, and $\bar{h} = h(\bar{\sigma})$, the GRASP restricted candidate list (RCL) is defined as

$$\text{RCL} = \{\sigma \in \mathcal{O}_c \mid \underline{h} \leq h(\sigma) \leq \underline{h} + \alpha(\bar{h} - \underline{h})\},$$

where α is a parameter such that $0 \leq \alpha \leq 1$.

The next operation to be scheduled is chosen at random from the RCL. In a standard GRASP, the candidates in the RCL are assigned equal probabilities of being chosen. However, any probability distribution can be used to bias the selection towards some particular candidates. Bresina [7] introduced a family of such probability distributions. In Bresina's selection procedure, the candidates are ranked according to the greedy function. Let $r(\sigma)$ denote the rank of element σ , Bresina studied several bias functions, e.g. random bias, $\text{bias}(r) = 1$, for $r \in \text{RCL}$, linear bias, $\text{bias}(r) = 1/r$, for $r \in \text{RCL}$, log bias, $\text{bias}(r) = \log^{-1}(r + 1)$, for $r \in \text{RCL}$, exponential bias, $\text{bias}(r) = e^{-r}$, for $r \in \text{RCL}$, and polynomial bias of order n , $\text{bias}(r) = r^{-n}$, for $r \in \text{RCL}$. In this work we will also use Bresina's selection procedure, but restricted to elements of the RCL.

Once we evaluate **bias** values for all elements of the RCL, we can calculate the probability $\pi(\sigma)$ of selecting operation σ as

$$\pi(\sigma) = \frac{\text{bias}(r(\sigma))}{\sum_{\sigma' \in \text{RCL}} \text{bias}(r(\sigma'))}. \quad (3.2)$$

Note that the standard GRASP uses a random bias function.

A typical iteration of the GRASP construction is summarized as follows: a partial schedule (which is initially empty) is on hand, the next operation to be scheduled is selected from the RCL and is added to the partial schedule, resulting in a new partial schedule. Construction ends when the partial schedule is complete, i.e. all operations have been scheduled.

3.3.2 Local search phase

Since there is no guarantee that the schedule obtained in the construction phase is locally optimal with respect to the local neighborhood being adopted, local search may improve the constructed solution.

The local search algorithm employed in this GRASP for JSP is the two exchange local search based on the disjunctive graph model of Roy and Sussmann [37]. In this model, the disjunctive graph $G = (V, A, E)$ is defined such that

$$V = \{\mathcal{O} \cup \{0, |\mathcal{J}| \cdot |\mathcal{M}| + 1\}\}$$

is the set of nodes, where $\{0\}$ and $\{|\mathcal{J}| \cdot |\mathcal{M}| + 1\}$ are artificial source and sink nodes, respectively,

$$\begin{aligned} A = & \{(v, w) \mid v, w \in \mathcal{O}, v \prec w\} \cup \\ & \{(0, w) \mid w \in \mathcal{O}, \#v \in \mathcal{O} \ni v \prec w\} \cup \\ & \{(v, |\mathcal{J}| \cdot |\mathcal{M}| + 1) \mid v \in \mathcal{O}, \#w \in \mathcal{O} \ni w \prec v\} \end{aligned}$$

is the set of directed arcs connecting consecutive operations of the same job, and

$$E = \{(v, w) \mid \mathcal{M}_v = \mathcal{M}_w\}$$

is the set of edges that connect operations on the same machine. Vertices in the disjunctive graph model are weighted. Vertices 0 and $|\mathcal{J}| \cdot |\mathcal{M}| + 1$ have weight zero, while the weight of vertex $i \in \{1, \dots, |\mathcal{J}| \cdot |\mathcal{M}|\}$ is the processing time of the operation corresponding to vertex i . Notice that the edges of A and E correspond, respectively, to constraints (3.1a) and (3.1b) of the disjunctive programming formulation of the JSP. An example of a disjunctive graph for a 3-job, 3-machine instance is shown in Figure 3.2.

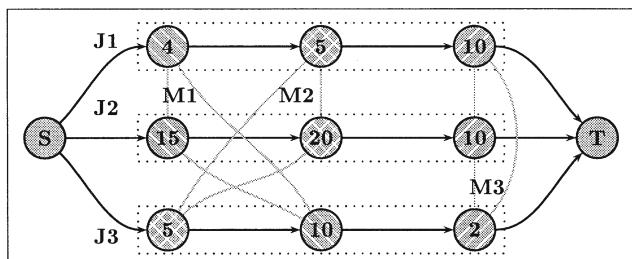


Figure 3.2 Disjunctive graph: representation of schedule.

A feasible schedule consists in giving an orientation for the edges in E . Given an orientation of E , one can compute the earliest start time of each operation by computing the longest (weighted) path from node 0 to the node corresponding to the operation. Consequently, the makespan of the schedule can be computed by finding the critical (longest) path from node 0 to node $|\mathcal{J}| \cdot |\mathcal{M}| + 1$, as illustrated in Figure 3.3. Thus, the objective of the JSP is to find an orientation of E such that the longest path in G is minimized.

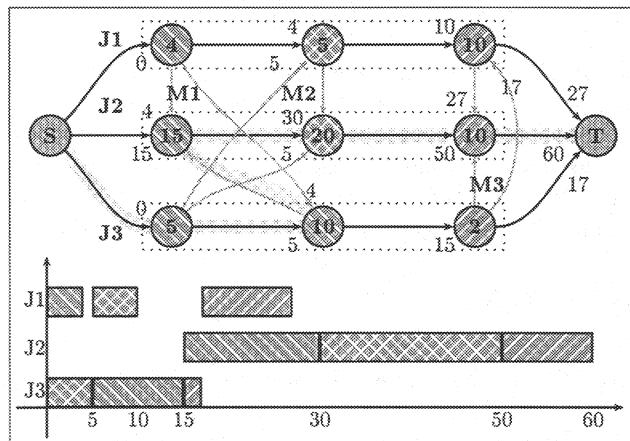


Figure 3.3 Disjunctive graph: critical path with a makespan of 60.

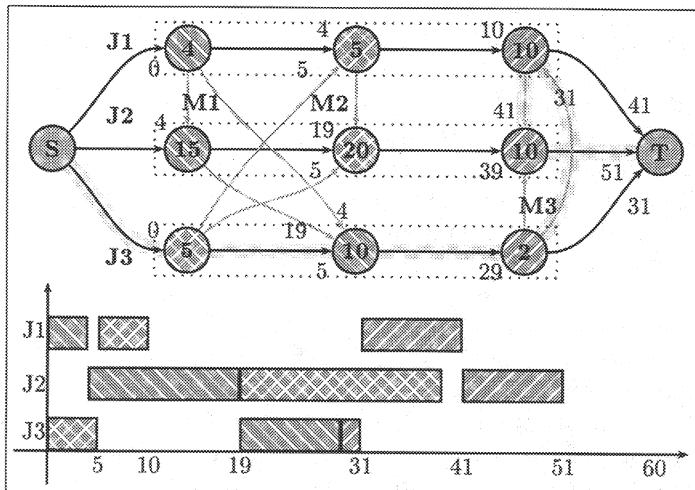


Figure 3.4 Disjunctive graph: recomputing the critical path with a makespan of 51.

Taillard [38] describes an $O(|\mathcal{J}| \cdot |\mathcal{M}|)$ algorithm to compute the longest path on G and an $O(|\mathcal{J}| \cdot |\mathcal{M}|)$ procedure to recompute the makespan when two consecutive operations in the critical path (on the same machine) are swapped.

Taillard [38] describes an $O(|\mathcal{J} \cdot \mathcal{M}|)$ algorithm to compute the longest path in G . Also, he shows that the entire neighborhood of a given schedule, where the neighborhood is defined by the swap of two consecutive operations in the critical path, can be examined, i.e. have their makespan computed, in complexity $O(|\mathcal{J} \cdot \mathcal{M}|)$ given that the longest path of G was evaluated. We use these procedures in our local search.

The local search procedure begins by identifying the critical path in the disjunctive graph corresponding to the schedule produced in the construction phase. All pairs of consecutive operations sharing the same machine in the critical path are tentatively exchanged. If the exchange improves the makespan, it is accepted. Otherwise, the exchange is undone. Once an exchange is accepted, the critical path may change and a new critical path must be identified. If no pairwise exchange of consecutive operations in the critical path improves the makespan, the current schedule is locally optimal and the local search ends. For example, in Figure 3.4, the second operation of job 3 was exchanged with the first operation of job 1, resulting in an improvement of the makespan.

3.4 INTENSIFICATION

One possible shortcoming of the standard GRASP framework is the independence of the GRASP iterations, i.e. the fact that GRASP does not *learn* from the history of solutions found in previous iterations. This is so because the standard GRASP discards information about any solution encountered that does not improve upon the incumbent. An obvious use of the information obtained from the “good” solutions is to implement a memory-based procedure to influence the construction phase by modifying the probabilities assigned to each RCL element, see eq. (3.2). Fleurent and Glover [21] introduced one such memory-based scheme. Their approach was illustrated in the context of the quadratic assignment problem (QAP), but is general and can be adapted to multistart methods, such as GRASP, for other combinatorial optimization problems. In this section, we present an intensification scheme based on the approach of Fleurent and Glover but specialized for the GRASP for JSP. Other enhancements to GRASP that make use of historical information include path relinking [27] and reactive GRASP [34].

The idea in the approach of Fleurent and Glover is to maintain a set \mathcal{E} of up to q diverse elite solutions and use this set to guide the construction phase of GRASP. A schedule found in any GRASP iteration is a candidate to be included in the elite set if its makespan is less than the largest makespan of the elite schedules. A candidate schedule is made elite if its makespan is smaller than the makespan of the best elite schedule (the elite schedule having the smallest makespan) or it is sufficiently different from all elite schedules. The elite set of schedules is initially empty and its worst makespan is arbitrarily set to ∞ . This is kept set to ∞ until the q elite schedules are included in the elite set. Once the elite set has q schedules and a new schedule is to be added to the elite set, the elite schedule having the largest makespan is removed from the elite set.

One way to measure the difference between two schedules is to compare the order of jobs processed at each machine. Another approach is to compare start times of each operation in each schedule. In the GRASP described in this paper, the latter approach is used because of its ease of implementation. A measure of non-similarity of two schedules is the number of different start times in both schedules. Let $\Delta(A, B)$ be the number of operations having different start times in schedules A and B and let δ be a parameter used to differentiate diverse schedules from similar schedules. Furthermore, let \mathcal{G} denote a GRASP schedule that is a candidate to become elite. For all elite schedules \mathcal{E}_i , $i = 1, \dots, q$, if $\Delta(\mathcal{G}, \mathcal{E}_i) \geq \delta \cdot |\mathcal{J}| \cdot |\mathcal{M}|$, schedule \mathcal{G} is made elite because it is sufficiently different from all elite schedules.

We next show how the set of elite solutions can be used to guide the search procedure in the construction phase. As in Fleurent and Glover [21], the idea is to bias the construction procedure to reinforce good characteristics of elite solutions. Our implementation of this concept differs slightly from the implementation of Fleurent and Glover. Define $t(\mathcal{S}, \sigma)$ to be the start time of operation σ in schedule \mathcal{S} and \mathcal{S}_p to be the partial schedule under construction. For $\sigma \in \text{RCL}$, the *intensity* index is defined as

$$\mathcal{I}(\sigma) = \sum_{s \in \mathcal{H}} \frac{C_{\max}(\mathcal{E}^*)}{C_{\max}(\mathcal{S})},$$

where $C_{\max}(\mathcal{E}^*)$ is the makespan of the best elite solution \mathcal{E}^* and $C_{\max}(\mathcal{S})$ is the makespan of schedule \mathcal{S} , and

$$\mathcal{H} = \{\mathcal{S} \in \mathcal{E} \mid t(\mathcal{S}, \sigma) = t(\mathcal{S}_p, \sigma), \text{ for all } \sigma \in \text{RCL}\},$$

is the set of elite schedules with operation σ having identical start time as operation σ of the partial schedule under construction. It is easy to see that the intensity of operation σ increases with the number of elite schedules having identical start times for this operation.

As discussed previously, the intensity function is used to alter the greedy heuristic function used in the construction phase. To accomplish this, define the new heuristic function, for all $\sigma \in \text{RCL}$, to be

$$h'(\sigma) = \lambda \frac{\underline{h}}{h(\sigma)} + \mathcal{I}(\sigma), \quad (3.3)$$

where λ is an adjustable parameter used to take into account the fact that $0 \leq \underline{h}/h(\sigma) \leq 1$ and $0 \leq \mathcal{I}(\sigma) \leq q$, where q is the number of elite schedules, and as defined previously, $\underline{h} = h(\underline{\sigma})$.

To give equal emphasis to $\underline{h}/h(\sigma)$ and $\mathcal{I}(\sigma)$, one would set $\lambda = q$. A strategy for setting the values of λ , suggested by Fleurent and Glover, is to give more emphasis to $\underline{h}/h(\sigma)$ in the early stages of the run and gradually shift the emphasis towards the intensity function. In this implementation, we use a scheme based on the variance of the makespan obtained for the last block of k GRASP iterations. If the variance increases, the λ parameter increases by 10.

Otherwise, if the variance decreases, the λ parameter decreases by 10. For the first block of $2k$ iterations, we take $\lambda = 100q$.

Having defined the new heuristic function, we are ready to describe the procedure used for selecting an element from the restricted candidate list. Initially, the RCL is set up in the same manner as described in Subsection 3.3.1. To select the next operation to be scheduled from the RCL, the new heuristic function is applied to all operations in the RCL and the operations are ranked according to their new heuristic function values. The probability of selecting each RCL operation is evaluated by (3.2) and an operation is sampled from the RCL according to this probability distribution.

3.5 PROXIMATE OPTIMALITY PRINCIPLE

In the construction phase of GRASP, schedules are built one operation at a time. The Proximate Optimality Principle (POP) [24], applied to this scheduling problem, states that good solutions of partial schedules with v operations are close to good solutions of partial schedules with $v + 1$ operations. An error in scheduling an operation early in the construction process may lead to other errors in scheduling operations that follow. The standard framework of GRASP corrects these erroneous steps with a local search procedure when the partial schedule is complete. However, due to the nature of local search procedures (such as 2-exchange), it may be difficult to identify and disassemble the erroneous decisions.

Fleurent and Glover [21] introduced the POP concept within the framework of a constructive multi-start method for the QAP. Periodically, they execute a local search procedure on a partial assignment (a restricted QAP problem) with the objective of reevaluating the assignments that were previously selected in the construction phase. In the context of job shop scheduling, the idea is similar, i.e. after a number of operations have been scheduled, the local search algorithm is executed on the partial schedule with the objective of reducing the makespan of the partial schedule.

To implement POP for the JSP, we use a slightly modified disjunctive graph based local search. Let S_p be the partial schedule and \mathcal{O}_p the corresponding operations of S_p . The disjunctive graph is built using only already-scheduled operations (from set \mathcal{O}_p), while the remaining operations are grouped in the sink node, as illustrated in Figure 3.5. In this way, only operations that have been already scheduled are allowed to be swapped in the local search. For example, in Figure 3.6, the second operation of job 2 was exchanged with the first operation of job 1, resulting in an improvement in the makespan of the schedule under construction. After executing POP in the partial schedule, we continue scheduling the yet unscheduled operations in the construction phase.

Because local search can be computationally demanding, POP cannot be applied at each iteration of the construction procedure. In the implementation used in this paper, a parameter `freq` is used to determine when POP is applied.

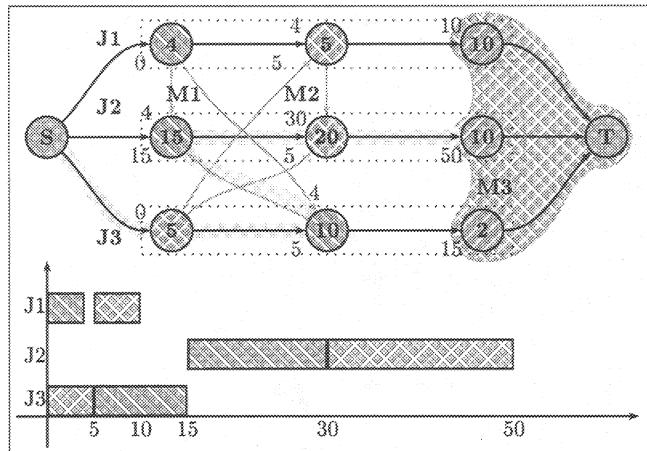


Figure 3.5 POP: Critical path of a partial schedule with a makespan of 50.

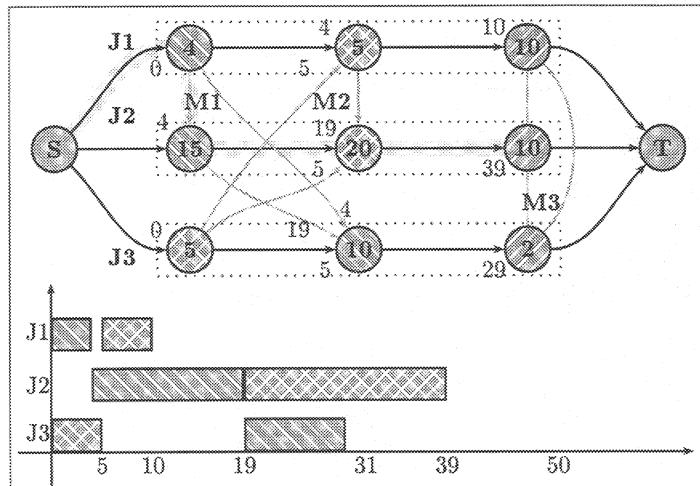


Figure 3.6 POP: Recomputing critical path and the partial schedule with a makespan of 39.

This parameter indicates the frequency of application of POP. For example, freq = 40 forces POP to be applied after 40% and 80% of the operations have been scheduled in the construction phase.

3.6 COMPUTATIONAL EXPERIMENTS

To illustrate the effectiveness and performance of an implementation of the GRASP described in this paper, we consider 66 instances from five classes of standard JSP test problems: abz, car, la, mt, and orb. Problem dimensions

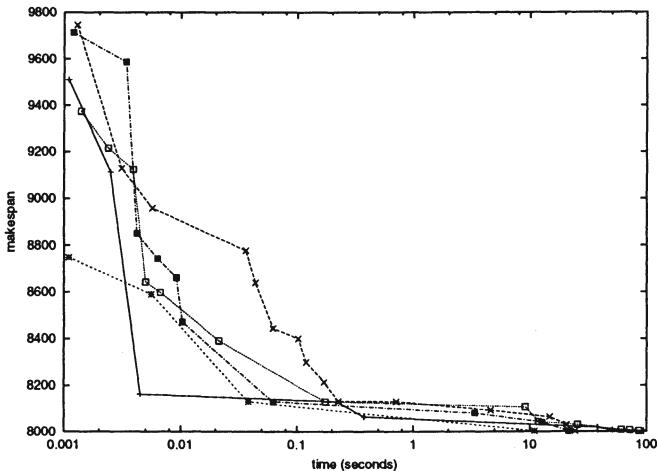


Figure 3.7 Problem car4: Incumbent makespan as a function of CPU time for five independent runs of GRASP. Runs stop when solution with a makespan of 8003 is found.

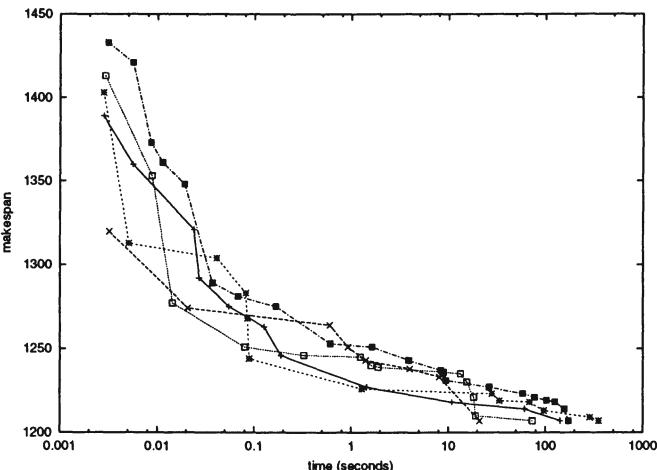


Figure 3.8 Problem la15: Incumbent makespan as a function of CPU time for five independent runs of GRASP. Runs stop when solution with a makespan of 1207 is found.

vary from 6 to 30 jobs and 4 to 20 machines. All test instances were downloaded from Beasley's OR-Library [6], <http://mscmga.ms.ic.ac.uk>.

The experiments were done on a Silicon Graphics Challenge computer (196 MHz MIPS R10000 processor). The codes were compiled with the SGI MIPSpro F77 compiler using flags `-O3 -static`. The CPU times are measured with the system routine `etime`.

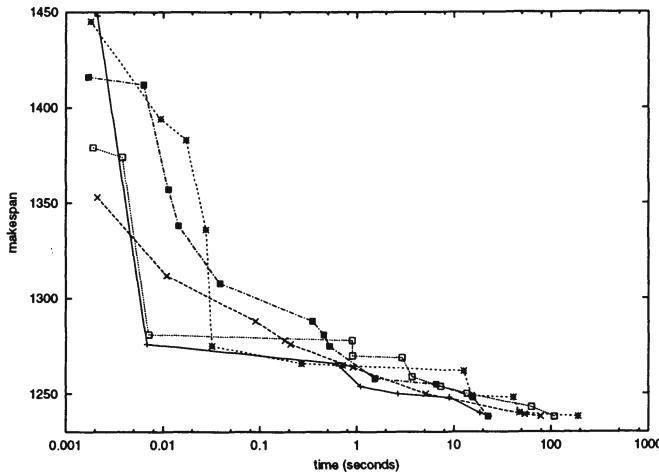


Figure 3.9 Problem abz5: Incumbent makespan as a function of CPU time for five independent runs of GRASP. Runs stop when solution with a makespan of 1238 is found.

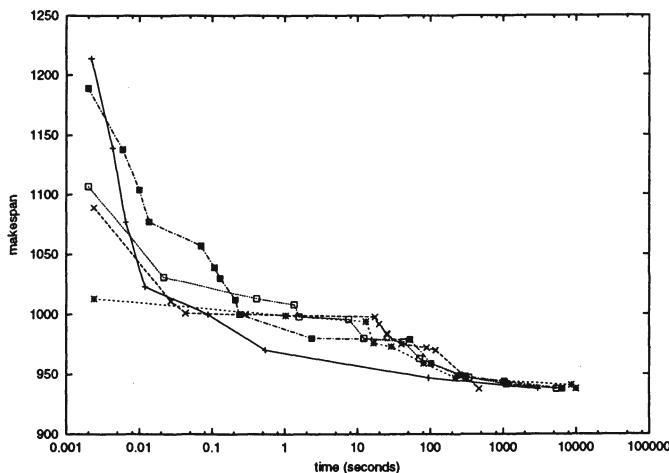


Figure 3.10 Problem mt10: Incumbent makespan as a function of CPU time for five independent runs of GRASP. Runs stop when solution with a makespan of 938 is found.

Extensive testing of the code, which we will not discuss here, determined a suitable choice of parameters. In general, we observed that the GRASP with intensification did better, i.e. improved the results, than the GRASP without intensification. An intensification scheme with an elite set of 30 solutions and a discrimination threshold value $\delta = 0.8$ was used. The variance of the makespans is computed every $k = 100$ iterations and the intensification parameter λ is adjusted as described in Section 3.4. As well, we observed that the GRASP

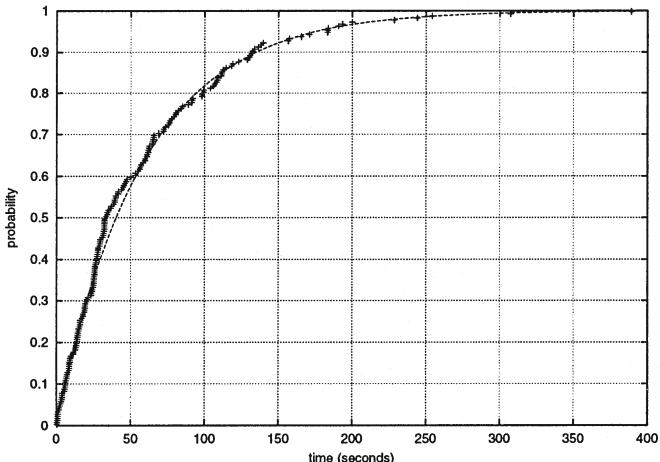


Figure 3.11 Problem 1a27: Superimposed empirical and theoretical distributions of time to sub-optimal within 10% of best known solution. Theoretical probability distribution is $F(t) = 1 - e^{-(t+0.649)/58.8688}$. Plot produced with data from 200 independent runs of GRASP.

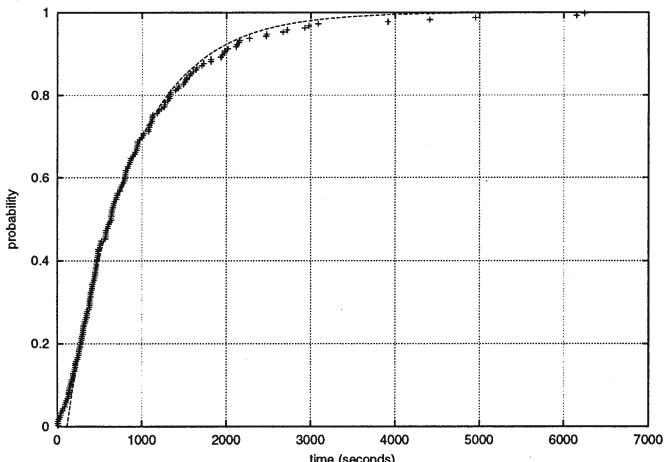


Figure 3.12 Problem orb7: Superimposed empirical and theoretical distributions of time to sub-optimal within 1% of best known solution. Theoretical probability distribution is $F(t) = 1 - e^{-(t-114.875)/733.5056}$. Plot produced with data from 200 independent runs of GRASP.

with POP local search did better than the GRASP without POP. Consequently, we opted to experiment with a GRASP having POP. POP was activated with parameter `freq = 40`. In these experiments, the RCL parameter α is selected

at random from the uniform distribution in the interval $[0, 1]$ and is fixed during each GRASP iteration. In general, we observed that the linear bias function was the best among the bias functions. Accordingly, a linear bias function was used for selecting candidates from the RCL.

Our objective with these experiments was to observe the general behavior of the algorithm, as well as learn how the algorithm performs on a standard set of test problems, i.e. how close to the best known solution (BKS) can the algorithm come.

Figures 3.7 – 3.10 show the general behavior of the algorithm on test problems **car4**, **1a15**, **abz5**, and **mt10**, respectively. These figures show the best makespan (incumbent) as a function of CPU time for five runs of GRASP (each using a distinct initial random number generator seed). The stopping criterion on these runs was solution quality. The runs terminated after the algorithm produced for the first time the best known GRASP solution. For the first two instances, these solutions are optimal, while for **abz5** the best known GRASP solution is about 0.3% off of the optimal and for **mt10** the best known GRASP solution is about 1% off of the optimal. In these figures, one can observe the wide distribution of CPU times.

As was shown by Aiex, Resende, and Ribeiro [2], many GRASP implementations have CPU time to find a given target solution that fits a two-parameter exponential distribution. This also appears to be the case for the GRASP for job shop scheduling described in this paper. Figures 3.11 and 3.12 show superimposed empirical and theoretical distributions of solution time to a target solution within 10% of the BKS for **1a27** and within 1% of the BKS for **orb7**, respectively. Each figure is generated with data from 200 independent runs of GRASP (using distinct initial random number generator seeds). The parameters of the two-parameter exponential distribution were estimated using the methodology described in [2]. Figure 3.11 shows, for example, that the probability of finding a solution within 10% of the BKS for **1a27** in less than 50 seconds on an SGI 196-MHz R10000 processor is about 60%, while with about 80% probability one expects to find such a solution in less than 100 seconds.

To learn how the algorithm performs with respect to solution quality, we ran the implementation extensively on all of the test problems. The number of GRASP iterations was often in the millions (accomplished by restarting the procedure with different initial random number generator seeds). Tables 3.1 and 3.2 report these results. Each table lists, for each test problem, its name, dimension (number of jobs and number of machines), the total number of GRASP iterations executed, the CPU time in SGI 196-MHz R10000 seconds per 1000 iterations, the value of the best known solution (BKS), the value of the best solution found by GRASP, and the relative error (in %) of the GRASP solution with respect to the BKS.

Of the 66 instances, GRASP found the BKS in 31 cases (47%). It was within 0.5% of the BKS in 38 instances (58%). In 44 cases (67%), the GRASP solution was within 1% of the BKS, and in 49 cases (74%), it was within 2% of the BKS.

problem	jobs	machines	iterations ($\times 10^6$)	time per 1000 iterations	BKS	GRASP solution	relative error (%)
abz5	10	10	20.1	0.3s	1234	1238	0.3
abz6	10	10	20.1	3.1s	943	947	0.4
abz7	15	20	20.1	17.4s	665	723	8.7
abz8	15	20	20.1	18.2s	670	729	8.8
abz9	15	20	20.1	17.1s	691	758	9.7
car1	11	5	0.1	1.4s	7038	7038	0.0
car2	13	4	0.1	1.5s	7166	7166	0.0
car3	12	5	240.1	1.5s	7312	7366	0.7
car4	14	4	0.1	1.6s	8003	8003	0.0
car5	10	6	20.1	1.3s	7702	7702	0.0
car6	8	9	10.1	1.5s	8313	8313	0.0
car7	7	7	0.1	1.0s	6558	6558	0.0
car8	7	7	0.1	1.0s	8264	8264	0.0
mt06	6	6	0.1	0.7s	55	55	0.0
mt10	10	10	90.1	2.9s	930	938	0.9
mt20	20	5	90.1	4.3s	1165	1169	0.3
orb1	10	10	40.1	2.9s	1059	1070	1.0
orb2	10	10	40.1	3.8s	888	889	0.1
orb3	10	10	40.1	3.1s	1005	1021	1.6
orb4	10	10	40.1	3.1s	1005	1031	2.6
orb5	10	10	40.1	2.8s	887	891	0.5
orb6	10	10	40.1	3.1s	1010	1013	0.3
orb7	10	10	10.1	3.2s	397	397	0.0
orb8	10	10	40.1	3.1s	899	909	1.1
orb9	10	10	40.1	3.1s	934	945	1.2
orb10	10	10	40.1	2.9s	944	953	1.0

Table 3.1 Experimental results on problem classes abz, car, mt, and orb. Table shows problem name, problem dimension (jobs and machines), total number of GRASP iterations performed, CPU time per 1000 GRASP iterations, the best known solution (BKS), the best solution found by GRASP, and the relative percentual error of the GRASP solution with respect to the BKS.

In 57 instances (86%) the GRASP solution was within 5% of the BKS, while for all cases the GRASP solution was within 10.5% of the BKS.

Tables 3.3 and 3.4 summarize these results by problem class. Table 3.3 shows, for each problem class, its name, the sum of the BKS values, the sum of the best GRASP solution values, and the relative error of the sum of the best GRASP solution values with respect to the sum of the BKS values. Table 3.4 shows, for each problem class, its name, the percentage of instances for which a GRASP solution within 0.5%, 1%, 2%, 5%, and 10% of the BKS was produced. From these tables, one can conclude that the easiest classes are car and mt, for which all GRASP solutions are within 1% of the BKS, while the most difficult are orb, la, and abz. Problem class abz was the one for which GRASP did worst, achieving an overall relative error of 4.52% with respect to the BKS. On the other two “hard” classes, the overall relative errors were much lower, 1.01% and 1.74%.

To compare the results obtained in this study with those obtained for other heuristics is a difficult task and is beyond the scope of this paper. The reader is referred to the survey of Jain and Meeran [25] where one attempt to tabulate results obtained using different methods is made.

problem	jobs	machines	iterations ($\times 10^6$)	time per 1000 iterations	BKS	GRASP solution	relative error (%)
la01	10	5	0.1	1.4s	666	666	0.0
la02	10	5	0.1	1.4s	655	655	0.0
la03	10	5	50.1	1.3s	597	604	1.2
la04	10	5	0.1	1.3s	590	590	0.0
la05	10	5	0.1	1.3s	593	593	0.0
la06	15	5	0.1	2.4s	926	926	0.0
la07	15	5	0.1	2.5s	890	890	0.0
la08	15	5	0.1	2.4s	863	863	0.0
la09	15	5	0.1	2.9s	951	951	0.0
la10	15	5	0.1	2.5s	958	958	0.0
la11	20	5	0.1	4.1s	1222	1222	0.0
la12	20	5	0.1	3.9s	1039	1039	0.0
la13	20	5	0.1	4.3s	1150	1150	0.0
la14	20	5	0.1	3.9s	1292	1292	0.0
la15	20	5	0.1	4.1s	1207	1207	0.0
la16	10	10	50.1	3.1s	945	946	0.1
la17	10	10	20.1	3.0s	784	784	0.0
la18	10	10	20.1	2.9s	848	848	0.0
la19	10	10	10.1	3.1s	842	842	0.0
la20	10	10	50.1	3.2s	902	907	0.6
la21	15	10	50.1	6.5s	1047	1091	4.2
la22	15	10	50.1	6.3s	927	960	3.6
la23	15	10	10.1	6.5s	1032	1032	0.0
la24	15	10	10.1	6.4s	935	978	4.6
la25	15	10	10.1	6.4s	977	1028	5.2
la26	20	10	10.1	10.8s	1218	1271	4.4
la27	20	10	10.1	10.9s	1235	1320	6.9
la28	20	10	10.1	10.9s	1216	1293	6.3
la29	20	10	10.1	11.1s	1157	1293	11.8
la30	20	10	10.1	10.5s	1355	1368	1.0
la31	30	10	10.1	22.9s	1784	1784	0.0
la32	30	10	10.1	23.9s	1850	1850	0.0
la33	30	10	10.1	23.9s	1719	1719	0.0
la34	30	10	10.1	23.8s	1721	1753	1.9
la35	30	10	10.1	22.0s	1888	1888	0.0
la36	15	15	11.2	10.3s	1268	1334	5.2
la37	15	15	11.2	10.3s	1397	1457	4.3
la38	15	15	11.2	10.6s	1196	1267	5.9
la39	15	15	11.2	10.3s	1233	1290	4.6
la40	15	15	11.2	11.0s	1222	1259	3.0

Table 3.2 Experimental results on problem classes 1a. Table shows problem name, problem dimension (jobs and machines), total number of GRASP iterations performed, CPU time per 1000 GRASP iterations, the best known solution (BKS), the best solution found by GRASP, and the relative percentual error of the GRASP solution with respect to the BKS.

problem	sum of BKS	sum of GRASP sol.	relative error (%)
abz	4203	4395	4.56
car	60356	60410	0.09
mt	2150	2162	0.56
orb	9028	9119	1.01
la	44297	45168	1.97

Table 3.3 Experimental results: Overall solution quality by problem class. Sum of all best known solutions (BKS) for each class is compared with sum of best GRASP solutions. Relative error is of GRASP solution with respect to BKS.

problem	BKS	percentage of GRASP solutions within				
		0.5% of BKS	1% of BKS	2% of BKS	5% of BKS	10% of BKS
abz	0.0	40.0	40.0	40.0	40.0	100.0
car	87.5	87.5	100.0	100.0	100.0	100.0
mt	33.3	66.6	100.0	100.0	100.0	100.0
orb	10.0	40.0	60.0	90.0	100.0	100.0
la	44.4	57.5	62.5	67.5	85.0	97.5

Table 3.4 Experimental results: Percentage of GRASP solutions within a tolerance of the best known solution (BKS).

3.7 CONCLUDING REMARKS

We describe a new algorithm for finding approximate solutions to the job shop scheduling problem. The algorithm is a greedy randomized adaptive search procedure (GRASP) with an intensification-enhanced construction phase which also makes use of the Proximate Optimality Principle (POP) to correct imperfections made in the construction phase. Local search is the standard two-exchange based on the disjunctive graph model. In general, the GRASP with intensification and POP was slightly better than a variant without those schemes.

The algorithm was evaluated on 66 standard test problems and was shown to produce optimal or near-optimal solutions on all instances. Though we verified that the time to target sub-optimal solution fits well a two-parameter exponential distribution for only two instances, we feel that this is the case in general. Hence, as discussed in [2], this algorithm can achieve linear speed-up with a parallel implementation.

To find better solutions on some of the notoriously difficult instances, such as abz9 and la29, where the algorithm found solutions about 10% off of the best known solution, an intensification scheme may be required to search the solution space around the elite solutions. One way in which this can be done is via path relinking, as described by Laguna and Martí [27].

References

- [1] J. Adams, E. Balas, and D. Zawack. The Shifting Bottleneck Procedure for Job Shop Scheduling. *Management Science*, 34:391–401, 1988.
- [2] R.M. Aiex, M.G.C. Resende, and C.C. Ribeiro. Probability Distribution of Solution Time in GRASP: An Experimental Investigation. To appear in: *Journal of Heuristics*.
- [3] D. Applegate and W. Cook. A Computational Study of the Job-Shop Scheduling Problem. *ORSA Journal on Computing*, 3:149–156, 1991.
- [4] J.F. Bard and T.A. Feo. Operations Sequencing in Discrete Parts Manufacturing. *Management Science*, 35:249–255, 1989.

- [5] J.F. Bard, T.A. Feo, and S. Holland. A GRASP for Scheduling Printed Wiring Board Assembly. *IIE Transactions*, 28:155–165, 1996.
- [6] J.E. Beasley. OR-Library: Distributing Test Problems by Electronic Mail. *Journal of the Operational Research Society*, 41:1069–1072, 1990.
- [7] J.L. Bresina. Heuristic-Biased Stochastic Sampling. In: *Proceedings of the AAAI-96*, pages 271–278, 1996.
- [8] P. Brucker, B. Jurisch, and B. Sievers. A Branch and Bound Algorithm for the Job-Shop Scheduling Problem. *Discrete Applied Mathematics*, 49:105–127, 1994.
- [9] J. Carlier and E. Pinson. An Algorithm for Solving the Job-Shop Problem. *Management Science*, 35:164–176, 1989.
- [10] J. Carlier and E. Pinson. A Practical Use of Jackson’s Preemptive Schedule for Solving the Job-Shop Problem. *Annals of Operations Research*, 26:269–287, 1990.
- [11] L. Davis. Job Shop Scheduling with Genetic Algorithms. In: *Proceedings of the First International Conference on Genetic Algorithms and their Applications*, pages 136–140, Morgan Kaufmann, 1985.
- [12] P. De, J.B. Ghosj, and C.E. Wells. Solving a Generalized Model for Con Due Date Assignment and Sequencing. *International Journal of Production Economics*, 34:179–185, 1994.
- [13] T.A. Feo, J. Bard, and S. Holland. Facility-Wide Planning and Scheduling of Printed Wiring Board Assembly. *Operations Research*, 43:219–230, 1995.
- [14] T.A. Feo and J.F. Bard. Flight Scheduling and Maintenance Base Planning. *Management Science*, 35:1415–1432, 1989.
- [15] T.A. Feo and M.G.C. Resende. A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem. *Operations Research Letters*, 8:67–71, 1989.
- [16] T.A. Feo and M.G.C. Resende. Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, 6:109–133, 1995.
- [17] T.A. Feo, K. Sarathy, and J. McGahan. A GRASP for Single Machine Scheduling with Sequence Dependent Setup Costs and Linear Delay Penalties. *Computers and Operations Research*, 23:881–895, 1996.
- [18] T.A. Feo, K. Venkatraman, and J.F. Bard. A GRASP for a Difficult Single Machine Scheduling Problem. *Computers and Operations Research*, 18:635–643, 1991.

- [19] P. Festa and M.G.C. Resende. GRASP: An Annotated Bibliography. In: *Essays and Surveys on Metaheuristics*, C.C. Ribeiro and P. Hansen, editors, Kluwer, 2001 (this volume).
- [20] H. Fisher and G.L. Thompson. Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules. In: *Industrial Scheduling*, J.F. Muth and G.L. Thompson, editors, pages 225–251, Prentice Hall, 1963.
- [21] C. Fleurent and F. Glover. Improved Constructive Multistart Strategies for the Quadratic Assignment Problem using Adaptive Memory. *INFORMS Journal on Computing*, 11:198–204, 1999.
- [22] S. French. *Sequencing and Scheduling: An Introduction to the Mathematics of the Job-Shop*. Horwood, 1982.
- [23] B. Giffler and G.L. Thompson. Algorithms for Solving Production Scheduling Problems. *Operations Research*, 8:487–503, 1960.
- [24] F. Glover and M. Laguna. Tabu Search. In: *Modern Heuristic Techniques for Combinatorial Problems*, C.R. Reeves, editor, pages 70–141. Blackwell, 1993.
- [25] A.S. Jain and S. Meeran. A State-of-the-Art Review of Job-Shop Scheduling Techniques. Technical report, Department of Applied Physics, Electronic and Mechanical Engineering, University of Dundee, 1998.
- [26] M. Laguna and J.L. González-Velarde. A Search Heuristic for Just-in-Time Scheduling in Parallel Machines. *Journal of Intelligent Manufacturing*, 2:253–260, 1991.
- [27] M. Laguna and R. Martí. GRASP and Path Relinking for 2-Layer Straight Line Crossing Minimization. *INFORMS Journal on Computing*, 11:44–52, 1999.
- [28] J.K. Lenstra and A.H.G. Rinnooy Kan. Computational Complexity of Discrete Optimization Problems. *Annals of Discrete Mathematics*, 4:121–140, 1979.
- [29] H. Ramalhinho Lourenço, J.P. Paixão, and R. Portugal. Metaheuristics for the Bus-Driver Scheduling Problem. Technical report, Department of Economics and Management, Universitat Pompeu Fabra, Barcelona, 1998.
- [30] H.R. Lourenço. Local Optimization and the Job-Shop Scheduling Problem. *European Journal of Operational Research*, 83:347–364, 1995.
- [31] H.R. Lourenço and M. Zwijnenburg. Combining the Large-Step Optimization with Tabu-Search: Application to the Job-Shop Scheduling Problem. In: *Meta-Heuristics: Theory and Applications*, I.H. Osman and J.P. Kelly, editors, pages 219–236, Kluwer, 1996.
- [32] E. Nowicki and C. Smutnicki. A Fast Taboo Search Algorithm for the Job Shop Problem. *Management Science*, 42:797–813, 1996.

- [33] E. Pinson. The Job Shop Scheduling Problem: A Concise Survey and Some Recent Developments. In: *Scheduling Theory and its Application*, P. Chrétienne, E.G. Coffman Jr., J.K. Lenstra, and Z. Liu, editors, pages 277–293, Wiley, 1995.
- [34] M. Prais and C.C. Ribeiro. Reactive GRASP: An application to a Matrix Decomposition Problem in TDMA Traffic Assignment. *INFORMS Journal on Computing*, 12:164–176, 2000.
- [35] R.Z. Ríos-Mercado and J.F. Bard. Heuristics for the Flow Line Problem with Setup Costs. *European Journal of Operational Research*, pages 76–98, 1998.
- [36] R.Z. Ríos-Mercado and J.F. Bard. An Enhanced TSP-Based Heuristic for Makespan Minimization in a Flow Shop with Setup Costs. *Journal of Heuristics*, 5:57–74, 1999.
- [37] B. Roy and B. Sussmann. Les problèmes d'ordonnancement avec contraintes disjonctives. SEMA Note D.S., n. 9, Paris, 1964.
- [38] E.D. Taillard. Parallel Taboo Search Techniques for the Job Shop Scheduling Problem. *ORSA Journal on Computing*, 6:108–117, 1994.
- [39] R.J.M. Vaessens, E.H.L. Aarts, and J.K. Lenstra. Job Shop Scheduling by Local Search. *INFORMS Journal on Computing*, 8:302–317, 1996.
- [40] P.J.M. Van Laarhoven, E.H.L. Aarts, and J.K. Lenstra. Job Shop Scheduling by Simulated Annealing. *Operations Research*, 40:113–125, 1992.
- [41] J. Xu and S. Chiu. Effective Heuristic Procedure for a Field Technician Scheduling Problem. Technical report, US WEST Advanced Technologies, 1998.

4 A REACTIVE GRASP FOR TRANSMISSION NETWORK EXPANSION PLANNING

Silvio Binato¹ and Gerson Couto Oliveira²

¹CEPEL, Centro de Pesquisas de Energia Elétrica
P.O. Box 68007
Rio de Janeiro 21944-970, Brazil
silvio@cepel.br

²Catholic University of Rio de Janeiro
Rua Marquês de São Vicente 225
Rio de Janeiro 22453-900 Brazil
gcouto@ele.puc-rio.br

Abstract: In this paper we describe an enhanced GRASP (Greedy Randomized Adaptive Search Procedure) applied to power transmission network expansion planning problems. GRASP is a metaheuristic that has been shown to be powerful in solving combinatorial problems. It is composed of two phases: the construction phase where a feasible solution is iteratively built, and a local search phase that seeks improvements within a given neighborhood of the solution found by the construction phase. The best solution over all GRASP iterations is chosen as the result. The enhancements analyzed in this paper are two: instead of using a random function in the construction phase, we use a linear bias function to guide the construction. We also applied a reactive procedure to self adjust GRASP parameters. Two real-world power transmission network expansion planning problems of the Brazilian power system are used to verify the performance of these improvements.

4.1 INTRODUCTION

Transmission networks transport large blocks of electric power from generating plants to loads. To reduce Ohmic losses in transmission lines, the power that is generated in low voltage is first transformed to high voltage and is then

transmitted along a meshed network to the main load centers, and there it is transformed back to low voltage and then distributed to consumers.

Nowadays, countrywide networks are interconnected, in order to allow the transfer of power across large areas, reducing total generation costs by dispatching the most efficient and economic generators, with minimal transmission limitations.

Transmission expansion planning involves the design of the most economic network reinforcements that allow the optimization of dispatch of the generators and transfer of power among different areas, avoiding transmission limitations and assuring the security and reliability of network operation. Environmental and economic constraints significantly reduce the availability of corridors for new transmission lines, specially near populated areas.

Due to the complex mathematical model of the physical laws governing the network when operating in steady state (a large system of non-linear relations, the so-called power flow equations and limits on generation and transmission equipment controls and variables [6]), the task of designing an optimal expansion plan is formidable, since binary decision variables associated to possible corridors and continuous variables related to operational state and control variables are present in the non-linear constraints that express network operating conditions. The resulting mixed-integer non-linear program is also large scale in terms of the number of binary variables, due not only to the need to design the optimal network topology but also due to the choice among different voltage levels for new transmission reinforcements. Branches using high voltage levels have greater transport capacity and also high investment costs. High voltage levels are preferable for long distance corridors in order to reduce ohmic losses. On the other hand, low voltage level branches have smaller capacity and smaller investment costs.

Investment costs in transmission network planning account for 30% to 40% of total investment in power expansion. In countries with a steady and high demand growth rates like Brazil (at least 5% per year), five billion dollars are required in power system expansion each year. Moreover, long distance transmission systems and interconnections among control areas wide apart have significant impact on total transmission investments, so small savings in expansion planning have great economic appeal.

Long term planning is mainly concerned in selecting which candidate transformers and transmission lines should be built so as to supply economically and reliably the forecasted load for a target horizon year. Usually, the planning of the generation expansion proceeds the transmission planning, so the total power supply is capable of meeting the forecasted total load.

In long term planning, it is usual to adopt a simplified version of the power flow equations and constraints, obtained by a linearization considering that all voltage magnitudes are at their nominal values, and neglecting second order terms when considering voltage angles. The resulting linearized power flow equations are known as Kirchoff's node and loop equations, that provide good approximations for power flows on high voltage meshed networks [16].

Several computational models have been proposed to cope with long term power transmission expansion planning problems. The first was due to Garver [11] and consists of an iterative procedure where the transmission facilities are selected, one at a time, using the most overloaded circuit (branch) criteria. Monticelli et al. [14] proposed the use of a “least effort” index to rank and select the most promising candidate circuits, which were built one at a time. Villasana et al. [24] improved Garver’s approach by combining the linear power flow with a transportation model, which was used to compute the overload flow. The use of Benders decomposition was introduced by Pereira et al. [17]. This was the first time an optimization framework was successfully applied to this problem.

An extensive study of this problem, using different formulations (transportation, hybrid, and linearized power flow) and alternative solution approaches (e.g. Benders’ decomposition, sensitivities, etc.), was published by Granville et al. [13, 12]. This work established a reference for the state of the art at that time. Following that, Romero and Monticelli [20, 21] used a hierarchical approach with an implicit enumeration algorithm to assemble a new successful Benders decomposition scheme. Later, Oliveira et al. [15] used a different hierarchical scheme and a B&B scheme, with a customized branching strategy, to solve the Benders master problem. Recently, Tsamasthyrou et al. [23] compared the results of Benders decomposition using two different formulations, previously proposed in [13, 12].

Several metaheuristic methods were also proposed to deal with this problem, for example: Simulating Annealing [19], Parallel Simulating Annealing [10], GRASP [1], Hybrid Tabu Search [9], Extended Genetic Algorithms [8], and a Hybrid Genetic Algorithm [2].

GRASP [7] is a metaheuristic method composed of two phases: construction and local search. In the construction phase, a feasible solution is produced and the local search phase explores its neighborhood. A generic pseudo code of GRASP is illustrated in Figure 4.1.

```

procedure GRASP(ProblemInstance, Parms)
    - BestSol  $\leftarrow \emptyset$ ;
    - for k = 1, ..., MaxIter do
        - Sol  $\leftarrow$  ConstructionPhase(Parms);
        - if Sol  $\neq$  NULL
            - Sol  $\leftarrow$  LocalSearch(Parms, Sol);
            - BestSol  $\leftarrow$  UpdateSolution(Sol);
        - enddo;
    - return BestSolution
end GRASP;

```

Figure 4.1 General description of GRASP.

The first application of GRASP to TNetEP problems was reported in [1]. The results showed that GRASP was a very promising technique for TNetEP

problems. In this paper, we enhance GRASP by applying distribution functions to bias the selection of variables of the restricted candidate list (*RCL*), see [4], and a reactive approach, see [18], to self-adjust the *RCL* parameter α used in the construction phase. The *RCL* is built at each iteration of the construction phase according to a greedy function, and is used to select the next candidate circuit to be added. The size of this list is controlled by the α parameter and different values of α may lead to different solutions for the construction phase. Low values for α can eliminate the diversification of the construction phase, while high α values make the construction phase highly random. Reactive GRASP [18] advocates to self-adjust parameters based on previously obtained results. The results of using these enhancements are illustrated with two real transmission planning studies of the Brazilian system.

This paper is organized as follows. Section 4.2 describes the power transmission network expansion planning (TNetEP) problem. The GRASP for TNetEP problems is described in Section 4.3. This section also describes the enhancements made to the classical GRASP approach. Section 4.4 illustrates the case study and presents some results. Finally, conclusions are given in Section 4.5.

4.2 THE POWER TRANSMISSION NETWORK EXPANSION MODEL

The steady-state model of a power transmission network consists of a system of non-linear equations and inequalities. The basic non-linear power flow equations are obtained from active and reactive power conservation laws on the network nodes. Denoting by \mathcal{N} the set of all nodes (cardinality of \mathcal{N} is written as $|\mathcal{N}|$), by \mathcal{E} the set of existing branches (power transmission lines, transformers, etc.), we can write the active and reactive power flow equations as

$$\begin{aligned} P^l_k - P^g_k &= \sum_{l \in \mathcal{E}_k} P_{kl}, \quad \forall k \in \mathcal{N}, \\ Q_{sh_k}(V_k) + Q^l_k - Q^g_k &= \sum_{l \in \mathcal{E}_k} Q_{kl}, \quad \forall k \in \mathcal{N}, \end{aligned} \tag{4.1}$$

where P^l_k , P^g_k , Q^l_k and Q^g_k , are, respectively, the active and reactive power demanded (superscript l) and generated (superscript g) at node k , \mathcal{E}_k is the set of branches directed connected to node k , and $Q_{sh_k}(V_k)$ is the “shunt” reactive power, which are functions of voltage magnitude (V_k) at node k . P_{kl} and Q_{kl} are the active and reactive power flows on branch kl , which is a function of the voltage magnitudes (V_k and V_l) and voltage angles (θ_k and θ_l) at nodes k and l , i.e.

$$\begin{aligned} P_{kl} &= V_k^2 g_{kl} - V_k V_l g_{kl} \cos \theta_{kl} - V_k V_l b_{kl} \sin \theta_{kl}, \quad \forall kl \in \mathcal{E}, \\ Q_{kl} &= -V_k^2 (b_{kl} + b_{kl}^{sh}) + V_k V_l b_{kl} \cos \theta_{kl} - V_k V_l g_{kl} \sin \theta_{kl}, \quad \forall kl \in \mathcal{E}, \end{aligned} \tag{4.2}$$

where $\theta_{kl} = \theta_k - \theta_l$ is the difference of voltage angles between terminal nodes k and l of branch kl . g_{kl} and b_{kl} are, respectively, the series conductance and susceptance of branch kl , which are computed from its resistance (r_{kl}) and reactance (x_{kl}) physical parameters, as

$$g_{kl} = \frac{r_{kl}}{r_{kl}^2 + x_{kl}^2} \text{ and } b_{kl} = \frac{-x_{kl}}{r_{kl}^2 + x_{kl}^2}, \quad \forall kl \in \mathcal{E}.$$

Finally, b_{kl}^{sh} is another physical branch parameter, the shunt susceptance, associated with the steady-state model of power-flow networks.

The set of inequalities basically include operational limits on the network equipments, for example: voltage limits,

$$\underline{V}_k \leq V_k \leq \overline{V}_k,$$

and active and reactive power generation limits,

$$\begin{aligned} \underline{P^g}_k &\leq P^g_k \leq \overline{P^g}_k, \\ \underline{Q^g}_k &\leq Q^g_k \leq \overline{Q^g}_k. \end{aligned}$$

In long-term studies the model of the network is usually simplified using the linearized power flow model, which is obtained from equations (4.2) by assuming that

$$\begin{aligned} V_k &= V_l = 1.0, \\ \sin \theta_{kl} &= \theta_{kl}, \\ r_{kl} &= 0, \end{aligned}$$

which results in the following equations for the power flow in the branch kl

$$f_{kl} = \gamma_{kl} (\theta_k - \theta_l)$$

where $\gamma_{kl} = x_{kl}^{-1}$.

Considering a set \mathcal{C} of all candidate branches that can be selected to be added to the basic network, the static long-term power transmission network design problem can be formulated by a mixed integer (0 – 1) nonlinear programming problem as follows:

$$\text{minimize } z = \sum_{kl \in \mathcal{C}} c_{kl} x_{kl}, \quad (4.3a)$$

$$\text{subject to: } \sum_{l \in \mathcal{E}_k} f_{kl}^0 + \sum_{l \in \mathcal{C}_k} f_{kl}^1 + g_k = d_k, \quad k \in \mathcal{N}, \quad (4.3b)$$

$$f_{kl}^0 - \gamma_{kl}^0 (\theta_k - \theta_l) = 0, \quad kl \in \mathcal{E}, \quad (4.3c)$$

$$f_{kl}^1 - x_{kl} \gamma_{kl}^1 (\theta_k - \theta_l) = 0, \quad kl \in \mathcal{C}, \quad (4.3d)$$

$$|f_{kl}^0| \leq \bar{f}_{kl}^0, \quad kl \in \mathcal{E}, \quad (4.3e)$$

$$|f_{kl}^1| \leq \bar{f}_{kl}^1 x_{kl}, \quad kl \in \mathcal{C}, \quad (4.3f)$$

$$0 \leq g_k \leq \bar{g}_k, \quad k \in \mathcal{N}, \quad (4.3g)$$

$$x_{kl} \in \{0, 1\}, \quad kl \in \mathcal{C}, \quad (4.3h)$$

where c_{kl} is the investment cost to build candidate branch kl . \mathcal{E}_k and \mathcal{C}_k represent the set of all existing and candidate branches directly connected with bus k . Superscripts indices ⁰ (¹) are references for existing (candidate) network variables. Using this notation, f_{kl} is the power flow in the branch kl , g_k is the active generation at node k , γ_{kl} states the branch susceptance for branch kl , θ_k

is the k -node voltage angle, \bar{f}_{kl} and \bar{g}_k state, respectively, kl -branch capacity and k -node generation limit.

The objective function of problem (4.3) corresponds to the minimization of all investment costs in building new transmission facilities. The constraints (4.3b) are the power flow balance equations for all nodes of the network, and constraints (4.3c) and (4.3d) are the linearized power flow equations for the existing and candidate network. The remaining constraints are operational limits and integrality conditions. Constraints (4.3h) represent the integrality conditions over the decision variables x . Note that if the kl candidate branch is not built, i.e. $x_{kl} = 0$, the corresponding branch flow over this candidate branch is required to be zero because of constraint (4.3f). Also, the second Kirchoff law (4.3d) should not be enforced for this branch. On the other hand, when $x_{kl} = 1$, i.e. the kl -th candidate branch is built, the second Kirchoff law is made valid, the branch flow is limited by \bar{f}_{kl} and constraint (4.3c) must be enforced. In this formulation the operation costs are neglected, otherwise a term $c^T g$ must be introduced in the objective function, in order to take into account the generation costs.

The main difficulty in solving this problem is due to the presence of binary variables $x_{kl}, \forall kl \in \mathcal{C}$. Another difficulty is due to the non-linear constraints (4.3c) involving integer and continuous variables. One alternative to solve such problems is to employ heuristic methods, which can provide good feasible, but not necessarily the optimal, solutions. Examples of heuristic approaches are methods that select one candidate branch to be built at a time, i.e. the vector x is iteratively constructed. It is possible that, for a given $x = \hat{x}$, the resulting linear programming problem (4.3) is infeasible. A way to deal with this is to introduce dummy variables representing the amount of load curtailment required to satisfy the power flow balance equations. Therefore, the TNetEP model becomes,

$$\text{minimize } z = \sum_{kl \in \mathcal{C}} c_{kl} x_{kl} + \sum_{k \in \mathcal{N}} p_k r_k \quad (4.4a)$$

$$\text{subject to: } \sum_{l \in \mathcal{E}_k} f_{kl}^0 + \sum_{l \in \mathcal{C}_k} f_{kl}^1 + g_k + r_k = d_k, \quad k \in \mathcal{N}, \quad (4.4b)$$

$$f_{kl}^0 - \gamma_{kl}^0 (\theta_k - \theta_l) = 0, \quad kl \in \mathcal{E}, \quad (4.4c)$$

$$f_{kl}^1 - x_{kl} \gamma_{kl}^1 (\theta_k - \theta_l) = 0, \quad kl \in \mathcal{C}, \quad (4.4d)$$

$$|f_{kl}^0| \leq \bar{f}_{kl}^0, \quad kl \in \mathcal{E}, \quad (4.4e)$$

$$|f_{kl}^1| \leq \bar{f}_{kl}^1 x_{kl}, \quad kl \in \mathcal{C}, \quad (4.4f)$$

$$0 \leq g_k \leq \bar{g}_k, \quad k \in \mathcal{N}, \quad (4.4g)$$

$$0 \leq r_k \leq \bar{g}_k, \quad k \in \mathcal{N}, \quad (4.4h)$$

$$x_{kl} \in \{0, 1\}, \quad kl \in \mathcal{C}, \quad (4.4i)$$

where r_k is a dummy generation (load curtailed) at node k , and p_k the penalty cost for loads not supplied. Note that, problem (4.4) is always feasible, for example if $r = d$ there are no power flows in the network. However, in long

term studies the penalty for the load not supplied, p , must be large enough to ensure $r = 0$, i.e. all loads must be supplied, otherwise the transmission expansion plan is classified as infeasible.

4.3 GRASP FOR TRANSMISSION NETWORK EXPANSION PLANNING PROBLEMS

In this section we specialize GRASP for the TNetEP problems and describe the implementation of the distribution bias function and the reactive approach.

4.3.1 Construction phase

The GRASP construction phase is an iterative procedure that builds a feasible network solution, choosing one candidate branch at a time to be included in the solution x . A feasible power transmission expansion plan is found when the solution of the operation problem, obtained from problem (4.4) considering a trial transmission expansion plan $x_{kl} = \hat{x}_{kl}$, $\forall kl \in \mathcal{C}$, has no load curtailment. The operation problem can be formulated as:

$$\text{minimize } \hat{z} = \sum_{k \in \mathcal{N}} p_k r_k \quad (4.5a)$$

$$\text{subject to: } \sum_{l \in \mathcal{E}_k} f_{kl}^0 + \sum_{l \in \mathcal{C}_k} f_{kl}^1 + g_k + r_k = d_k, \quad k \in \mathcal{N}, \quad \pi_k^d, \quad (4.5b)$$

$$f_{kl}^0 - \gamma_{kl}^0 (\theta_k - \theta_l) = 0, \quad kl \in \mathcal{E}, \quad \pi_{kl}^0, \quad (4.5c)$$

$$f_{kl}^1 - \hat{x}_{kl} \gamma_{kl}^1 (\theta_k - \theta_l) = 0, \quad kl \in \mathcal{C}, \quad \pi_{kl}^1, \quad (4.5d)$$

$$|f_{kl}^0| \leq \bar{f}_{kl}^0, \quad kl \in \mathcal{E}, \quad \pi_{kl}^{f^0}, \quad (4.5e)$$

$$|f_{kl}^1| \leq \bar{f}_{kl}^1 \hat{x}_{kl}, \quad kl \in \mathcal{C}, \quad \pi_{kl}^{f^1}, \quad (4.5f)$$

$$0 \leq g_k \leq \bar{g}_k, \quad k \in \mathcal{N}, \quad \pi_k^g, \quad (4.5g)$$

$$0 \leq r_k \leq \bar{g}_k, \quad k \in \mathcal{N}, \quad \pi_k^r, \quad (4.5h)$$

where π_k^d , π_{kl}^0 , π_{kl}^1 , $\pi_{kl}^{f^0}$, $\pi_{kl}^{f^1}$, π_k^g and π_k^r are the Lagrange multipliers (dual variables) of the linear program (4.5). This problem can be efficiently solved via a customized Dual Simplex code [22], which uses reduced basis and upper bound techniques.

The GRASP construction phase procedure is based on the sensitivity of the operation problem with respect to branch susceptance, i.e. $\frac{\partial \hat{z}}{\partial \gamma}$. It was shown in [5, 12] that one can estimate this sensitivity index by

$$\pi_{kl}^\gamma = (\pi_l^d - \pi_k^d)(\theta_k - \theta_l), \quad \forall kl \in \mathcal{C}. \quad (4.6)$$

Usually, the index π_{kl}^γ is negative indicating the marginal benefit of adding a new branch to the network. To take into account the investment costs, we define the greedy function h_{kl} as the feasibility sensitivity π_{kl}^γ divided by the cost of each candidate branch, i.e.

$$h_{kl} = \frac{\pi_{kl}^\gamma}{c_{kl}}, \forall kl \in \mathcal{C}. \quad (4.7)$$

The greedy choice for adding one candidate branch to the network is to select and build the candidate branch with the smallest (negative) h_{kl} value, i.e.

$$(\bar{kl}) = \arg \min_{kl \in \mathcal{C}} (h_{kl}).$$

Defining

$$(\underline{kl}) = \arg \max_{kl \in \mathcal{C}} (h_{kl})$$

and

$$h^{\max} = h_{(\bar{kl})}, h^{\min} = h_{(\underline{kl})},$$

the GRASP restricted candidate list (*RCL*) is defined as follows:

$$RCL = \{kl \in \mathcal{C} \mid h^{\min} \leq h_{kl} \leq h^{\max} + \alpha(h^{\max} - h^{\min})\}, \quad (4.8)$$

where α is a real number between 0 and 1.

In the GRASP standard approach the next candidate branch is chosen from the *RCL* at random, and a new GRASP construction phase iteration is performed, i.e. a new LP problem must be solved to update the greedy function. The GRASP construction phase stops when network feasibility is reached.

Note that when the network feasibility is reached, the construction phase solution may contain several redundant additions, e.g. candidate branches built such that they do not cause any infeasibility when removed. To eliminate them, we rank all previously made additions by decreasing cost and take out the ones whose removal do not lead to an infeasible solution. The solution obtained after removal of redundant additions is the construction phase solution, which is the starting point for the local search, described later. In case of failure of GRASP construction phase, i.e. if no feasible solution is obtained, the GRASP “outer” iteration is aborted (i.e. local search is not performed).

Figures 4.2 and 4.3 summarize, respectively, the GRASP construction phase for TNetEP and the procedure to remove the redundant branch additions.

4.3.2 Distribution bias functions

Instead of using a random distribution function to select the *RCL* candidate branch to be added, one can use any distribution function to bias the search towards the best candidate branches. The use of distribution bias function in GRASP-type algorithms was introduced by Bresina [4], who studied several distribution bias functions (linear, exponential, etc.) to evaluate the probability of selecting each *RCL* candidate.

Besides the random selection, in this work we implemented a linear bias function defined as:

$$bias(k) = \frac{1}{k}, k \in RCL.$$

```

procedure ConstructionPhase(Parms)
-    $\hat{x} \leftarrow 0;$ 
-   Solve linear program (4.5);
-   while  $\hat{z} > 0.0$  do
-       Compute  $h_{kl}, \forall kl \in \mathcal{C}$  according (4.7);
-       Build the RCL as (4.8);
-       if RCL =  $\emptyset$ 
-           return(NULL);
-       else
-            $kl \leftarrow RAND(RCL)$  ;
-            $\hat{x}_{kl} = 1;$ 
-           Solve linear program (4.5);
-       enddo;
-   RemoveRedundantBranchAdditions( $\hat{x}$ );
-   return ( $\hat{x}$ )
end ConstructionPhase.

```

Figure 4.2 The GRASP construction phase for TNetEP.

```

procedure RemoveRedundantBranchAdditions( $\hat{x}$ )
-   REMOVED = TRUE;
-    $\mathcal{X} = \{kl \mid x_{kl} = 1, \forall kl \in \mathcal{C}\};$ 
-   while REMOVED = TRUE do
-       REMOVED = FALSE;
-       for  $kl = \arg \max_{ij \in \mathcal{X}} \{c_{ij}\}$  do
-            $\hat{x}_{kl} = 0;$ 
-           Solve linear program (4.5);
-           if  $\hat{z} > 0.0$ 
-                $\hat{x}_{kl} = 1;$ 
-           else
-               REMOVED = TRUE;
-                $\mathcal{X} = \mathcal{X} \setminus \{kl\};$ 
-           enddo
-       enddo
-   return ( $\hat{x}$ )
end RemoveRedundantBranchAdditions.

```

Figure 4.3 Procedure to remove redundant branches from a feasible solution.

Let $rank(kl)$ and $bias(rank(kl))$ denote, respectively, the rank and the value of the linear bias function for the candidate branch (kl). The probability of selecting this candidate branch from *RCL* is,

$$P_{kl} = \frac{bias(rank(kl))}{\sum_{(ij) \in RCL} bias(rank(ij))}. \quad (4.9)$$

4.3.3 Reactive GRASP

GRASP-type algorithms require setting up some parameters before execution. An example is α , which implicitly controls the size of the RCL in the construction phase. A unfavorable value for α may bring a serious impact for the final solution, degrading its quality. The idea of reactive GRASP is to self-adjust this GRASP parameter during the execution, according to the quality of the solutions previously obtained. Reactive GRASP was proposed by Prais and Ribeiro for the TDMA traffic assignment problem [18].

We follow the work of Prais and Ribeiro, using a similar scheme to select α values from a discrete set, \mathcal{A} , of m predetermined acceptable values. Using different values of α in different iterations can eventually lead to construction phase solutions which would never be built if a single value of α was used. Let p_i be the probability of selecting $\alpha = \bar{\alpha}_i, i = 1, \dots, m$, where $\bar{\alpha}_i \in \mathcal{A}$ (initially we use $p_i = 1/m, i = 1, \dots, m$). In order to update the distribution p , we compute the average solution values obtained along the last block of k iterations, i.e. $avg_i, i = 1, \dots, m$ to obtain

$$q_i = \left(\frac{v^*}{avg_i} \right)^\delta, \quad i = 1, \dots, m, \quad (4.10)$$

where v^* is the best overall solution yet found. After normalizing, we get the new probability values, i.e.

$$p_i = \frac{q_i}{\sum_{j=1}^m q_j}. \quad (4.11)$$

The exponent δ is used to attenuate the updated values of the probabilities.

4.3.4 Local search phase

This phase starts from the construction phase feasible solution, and tries to find better solutions within a given neighborhood. The GRASP Local Search phase is based on branch exchanges. Before the explanation of how we implemented the local search procedure, it is interesting to notice that the number of exchanges to be analyzed increases exponentially with the number of candidate branches, so complete enumeration is computationally prohibitive, even for medium-size problems. In our implementation we limit the size of neighborhood allowing 1– or 2–exchanges as explained below.

Figure 4.4 illustrates the recursive local search procedure. The first steps consist of the initialization of the cost of the current solution ($OldCost$), the set of previously added candidate branches, the complementary set $\bar{\mathcal{X}}$ of \mathcal{X} with respect to \mathcal{C} , and the set \mathcal{H} of all combinations of elements of \mathcal{X} , that depends on the chosen neighborhood (1– or 2– exchange).

The body of the local search procedure, steps 6 – 31, consists in testing the removal of all elements of set \mathcal{H} from the current solution \hat{x} in order to explore the neighborhood of this solution. In Steps 7 – 9, an element of \mathcal{H} is selected to be removed, in step 10 the linear program (4.5) is solved for the partial solution \hat{x} , and in steps 11 – 13 a set \mathcal{W} is created, whose elements are combinations

```

procedure LocalSearch(parms,  $\hat{x}$ )
  1 -    $OldCost = \sum_{uv \in C} c_{uv} \hat{x}_{uv};$ 
  2 -    $\mathcal{X} = \{kl \mid x_{kl} = 1, \forall kl \in C\}; \bar{\mathcal{X}} = C \setminus \mathcal{X};$ 
  3 -    $\mathcal{H} = \mathcal{X};$ 
  4 -   if 2-exchange
  5 -      $\mathcal{H} = \mathcal{H} \times \mathcal{X};$ 
  6 -   while  $\mathcal{H} \neq \emptyset$  do
  7 -      $kl = RAND(\mathcal{H}); \mathcal{H} = \mathcal{H} \setminus \{kl\}; \hat{x}_{kl} = 0;$ 
  8 -     if 2-exchange
  9 -        $ij = RAND(\mathcal{H}); \mathcal{H} = \mathcal{H} \setminus \{ij\}; \hat{x}_{ij} = 0;$ 
 10 -    Solve linear program (4.5);
 11 -     $\mathcal{W} = \{pq \in \bar{\mathcal{X}} \mid \pi_{pq}^\gamma \leq 0\};$ 
 12 -    if 2-exchange
 13 -       $\mathcal{W} = \mathcal{W} \times \{rs \in \bar{\mathcal{X}} \mid \pi_{rs}^\gamma \leq 0\};$ 
 14 -    while  $\mathcal{W} \neq \emptyset$  do
 15 -      if 1-exchange
 16 -         $pq = \arg \min_{uv \in \mathcal{W}} \{h_{uv}\};$ 
 17 -         $\hat{x}_{pq} = 1; \mathcal{W} = \mathcal{W} \setminus \{pq\};$ 
 18 -      if 2-exchange
 19 -         $(pq, rs) = \arg \min_{(uv, wz) \in \mathcal{W}} \{(h_{uv}, h_{wz})\};$ 
 20 -         $\hat{x}_{pq} = 1; \hat{x}_{rs} = 1; \mathcal{W} = \mathcal{W} \setminus \{(pq, rs)\};$ 
 21 -        Solve linear program (4.5);
 22 -      if  $\hat{z} = 0.0$ 
 23 -        RemoveRedundantBranchAdditions( $\hat{x}$ );
 24 -        if  $\sum_{uv \in C} c_{uv} \hat{x}_{uv} < OldCost$ 
 25 -          LocalSearch(parms,  $\hat{x}$ )
 26 -          halt();
 27 -         $\hat{x}_{kl} = 1; \hat{x}_{pq} = 0;$ 
 28 -        if 2-exchange
 29 -           $\hat{x}_{ij} = 1; \hat{x}_{rs} = 0;$ 
 30 -        enddo;
 31 -      enddo;
end LocalSearch.

```

Figure 4.4 Local search procedure for TNetEP problems.

of candidate branches (in $\bar{\mathcal{X}}$). The objective is to evaluate new combinations of candidate branches that may be exchanged with the candidate branches previously selected from \mathcal{H} . In order to reduce the number of elements in \mathcal{W} , only branches kl whose addition is potentially beneficial in reducing network infeasibilities, i.e $\pi_{kl}^\gamma \leq 0$, are allowed.

The examination of \mathcal{W} consists in selecting its most promising element according to the greedy function, eq. (4.7), solving the resulting linear program considering the addition of branches selected from \mathcal{W} , checking if the network is feasible and, if so, removing redundant branch additions. Finally, if the cost of this new solution is “cheaper” than the former solution, saved in $OldCost$, a new local search procedure is started.

In case of failure, either due to not finding a feasible solution or the cost of the new solution is more expensive than the former cost, the previous solution \hat{x} is recovered to continue exploring its neighborhood.

The local search procedure stops when the whole neighborhood has been examined.

4.4 CASE STUDY

The GRASP for transmission network expansion planning problem was implemented using the C and Fortran programming languages, and the results reported were obtained on a SUN Ultra-2 workstation with 128 Mbytes of memory.

Two power transmission expansion case studies will be presented to illustrate our reactive-GRASP approach. The first case study corresponds to a two-high voltage level network of the reduced southern Brazilian system. It has been discussed in many references, see for example [17, 13, 12, 20, 21]. The second case study refers to the reduced southeastern Brazilian system, which has been studied in [15, 1, 3].

To assess the effect of different bias functions and the parameter δ , reactive cases with linear and random bias function and $\delta = 1$ and 5 were considered, where each case was processed 5 times using different initial seeds. Two fixed α parameter cases (0.75 and 0.5) were also considered, as shown in Table 4.1. For reactive runs, the cardinality of set \mathcal{A} is 10 and the k -block value is 50 iterations. For each of the above cases, 500 GRASP iterations were made.

Case	Bias	α
0	Linear	Reactive, $\delta = 5$
1	Random	Reactive, $\delta = 5$
2	Linear	Reactive, $\delta = 1$
3	Random	Reactive, $\delta = 1$
4	Linear	Fixed 0.75
5	Random	Fixed 0.75
6	Linear	Fixed 0.50
7	Random	Fixed 0.50

Table 4.1 Description of GRASP runs.

4.4.1 The reduced southern Brazilian network

The reduced southern Brazilian network has 46 nodes (nodes 28 and 31 have new generation units and must be connected to the network), 62 existing branches and 17 new corridors (rights of way). The number of candidate circuits is 237 ($3 \times (62 + 17)$). Figure 4.5 gives an idea of the topology of this power system and shows the existing and candidate branches (dotted lines). The resulting problem (4.4) has 237 binary variables, 437 linear variables, and 345 constraints excluding bounds on variables.

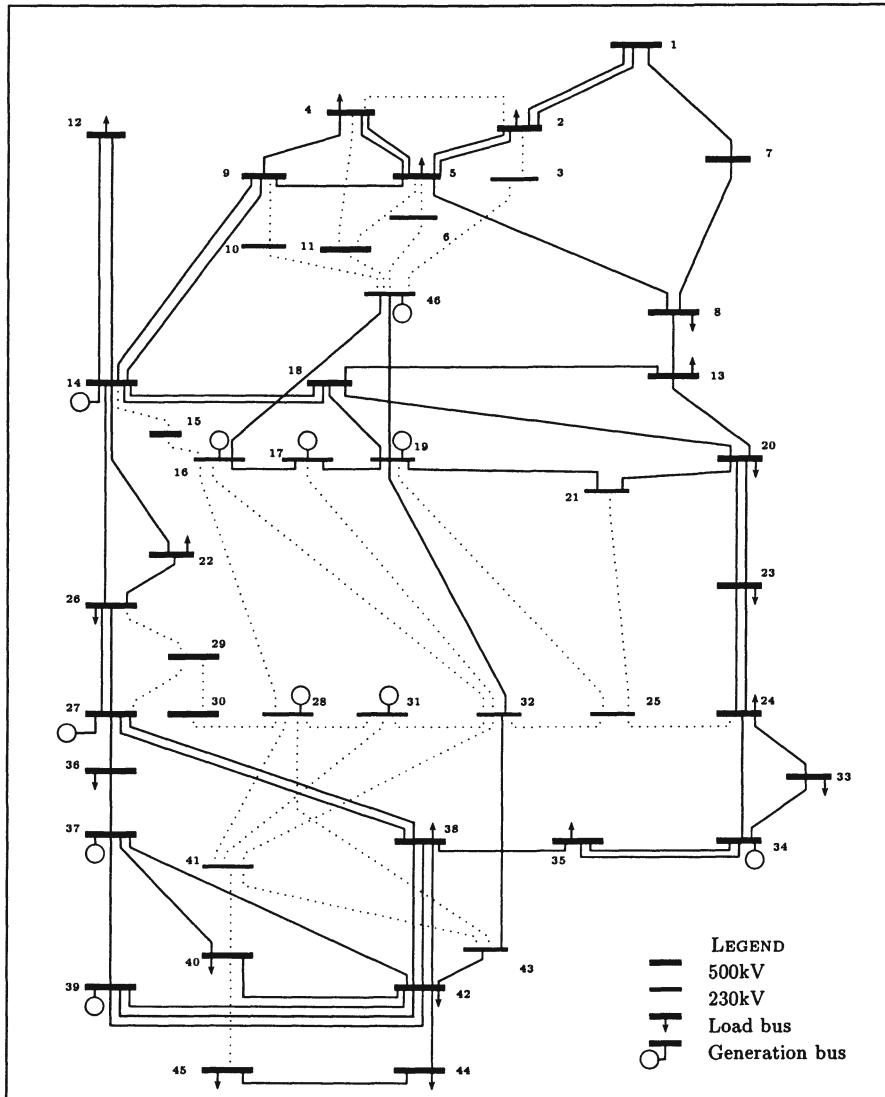


Figure 4.5 The reduced southern Brazilian system.

The optimal solution was first published by Romero and Monticelli in [20]. The investment cost of this solution is US\$154.26 millions. Table 4.2 presents the 16 added candidate circuits.

For this network, we will repeat all runs using 1- and 2-exchange neighborhoods during local search, in order to draw comparisons regarding the capability of finding the optimal solution.

From	To	# add.	From	To	# add.
26	29	3	19	25	1
42	43	2	46	6	1
24	25	2	31	32	1
29	30	2	28	30	1
5	6	2	20	21	1

Table 4.2 Best known solution for the reduced southern Brazilian system.

The CPU computing time for each run, when the neighborhood in the local search was 1-exchange, varies from 7min20s to 8min50s, and the total CPU processing time (all runs) was around 5h20min. When the neighborhood size in the local search was set to 2-exchange, the CPU computing time for each run varied from 39min20s to 50min20s. In this last case, the total computing time, to process all runs, was around 30h. The results obtained for all runs in this case study are displayed in Table 4.3 and Table 4.4.

Case	10 ⁶ US\$						
	run1	run2	run3	run4	run5	Avg.	Best
0	154.26	154.26	154.26	154.26	154.26	154.26	154.26
1	162.57	162.57	163.09	162.57	154.26	161.02	154.26
2	154.26	154.26	162.57	154.26	154.26	155.93	154.26
3	162.57	163.05	164.76	162.57	154.26	161.44	154.26
4	154.26	154.26	154.26	162.57	154.26	155.93	154.26
5	154.26	154.26	154.26	162.57	154.26	155.93	154.26
6	162.57	162.57	162.57	171.65	174.85	166.84	162.57
7	162.57	162.57	154.26	154.26	162.57	159.25	154.26

Table 4.3 Results for the reduced southern Brazilian system: 1-exchange neighborhood.

Case	10 ⁶ US\$						
	run1	run2	run3	run4	run5	Avg.	Best
0	154.26	154.26	154.26	154.26	154.26	154.26	154.26
1	162.57	154.26	154.26	154.26	154.26	155.93	154.26
2	154.26	154.26	154.26	154.26	154.26	154.26	154.26
3	162.57	154.26	154.26	162.57	154.26	157.59	154.26
4	154.26	154.26	154.26	154.26	154.26	154.26	154.26
5	154.26	154.26	154.26	154.26	154.26	154.26	154.26
6	162.57	162.57	162.57	171.65	174.85	166.84	162.57
7	162.57	162.57	154.26	154.26	162.57	159.25	154.26

Table 4.4 Results for the reduced southern Brazilian system: 2-exchange neighborhood.

Note that for both neighborhoods, the best known solution was found in all cases but one. Also, their average solution values are similar, indicating that there is no significant deterioration due to using only 1-exchange.

Now let us consider the 1-exchange results. It is interesting to point out that using α reactive approach with linear bias the optimal solution was found in all five (different initial random seed) runs.

Analyzing the average values we can see that when the α reactive approach was used the results of linear bias function are better than the results found by random bias functions. When the α reactive approach was not used, the best results were found with high values of α , and it was also found in all runs of case 0 (linear bias and reactive α approach).

Table 4.5 presents the average and best solution values for the random and linear bias functions, and using fixed ($\alpha = 0.75$) and α reactive approach ($\delta = 5$). We can see that combining linear bias functions and α reactive approach ($\delta = 5$) results in the best average performance.

Sampling function	10^6 US\$		10^6 US\$	
	$\alpha = 0.75$	α reactive	Avg.	Best
Random	155.93	154.26	161.02	154.26
Linear	155.93	154.26	154.26	154.26

Table 4.5 Results for random and linear bias, fixed and α reactive approach ($\delta = 5$) for 1-exchange.

4.4.2 The Reduced Southeastern Brazilian Network

The reduced Southeastern Brazilian network has 79 nodes and 155 existing branches. Figure 4.6 shows the network system, illustrating both existing and candidate branches (dotted lines). It must be expanded to meet the forecasted load for year 2000. Formulating this problem like (4.4), would lead to 429 (3×143) binary variables, 821 linear variables, and 663 constraints, excluding bounds on variables. This problem instance is very difficult to solve because of its size and the need to choose candidates among five different voltage levels (750kV, 500kV, 440kV, 345kV, and 230kV).

Due to the large number of candidate circuits of this case study, we did not use the 2-exchange approach in the GRASP local search phase, allowing only 1-exchange for all runs. In order to check the effect of 2-exchange in the local search, a pilot run was made using the case 0, which consumed more than 3 hours of CPU time.

Results for the five runs are shown in Table 4.7. Table 4.6 presents the best known solution found only by cases 0 and 2, with 24 added candidate circuits.

It can be seen that the best solution was only found using α -reactive and linear bias function. This solution is the same obtained for case 0 using the

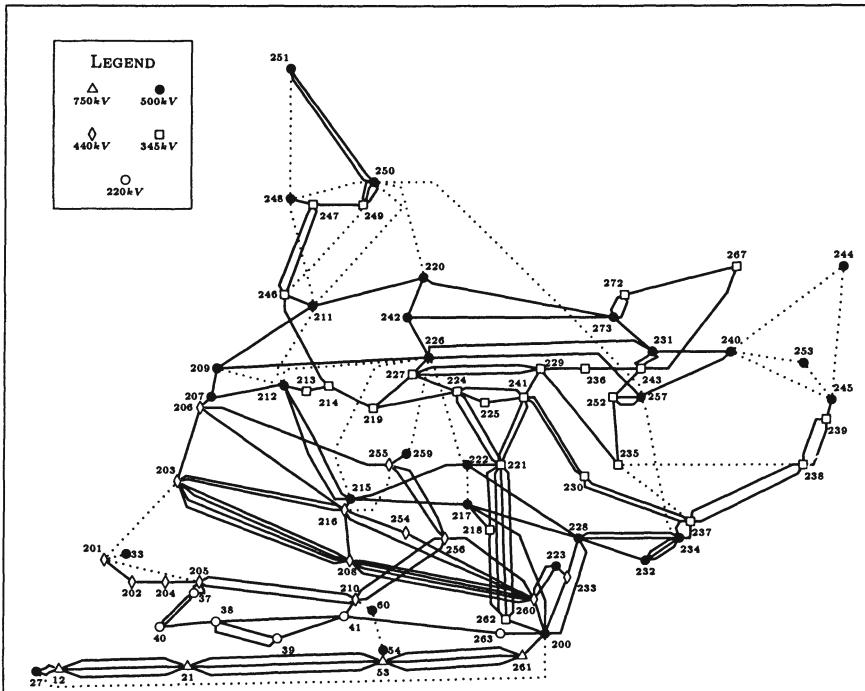


Figure 4.6 The reduced southeastern Brazilian power network.

From	To	# add.	From	To	# add.
224	227	2	210	41	2
255	259	2	220	242	2
226	242	2	220	250	1
234	237	1	221	224	1
245	253	1	245	239	1
244	245	1	226	259	1
211	246	1	226	227	1
250	251	1	207	206	1
207	209	1	249	250	1
216	215	1			

Table 4.6 Best known solution for the reduced southeastern Brazilian system.

2-exchange GRASP local search. When using different bias functions, linear bias was never worse than the random bias, and helps finding the best solution. Analyzing average solution values, we see that using the reactive GRASP approach results in a better average solution value than the traditional fixed α GRASP approach. Each run consumes from 10min18s to 11min52s, and the total processing CPU time was around 7.5h.

Case	$10^6 \text{US\$}$						
	run1	run2	run3	run4	run5	Avg.	Best
0	430.88	424.62	422.60	430.88	430.88	427.97	422.60
1	430.88	430.88	424.62	424.62	430.88	428.38	424.62
2	430.88	422.60	430.88	430.88	430.88	429.22	422.60
3	424.62	430.88	430.88	430.88	430.88	429.63	424.62
4	430.88	430.88	430.88	430.88	430.88	430.88	430.88
5	430.88	430.88	430.88	430.88	430.88	430.88	430.88
6	470.72	470.72	470.72	470.72	470.72	470.72	470.72
7	470.72	470.72	470.72	470.72	470.72	470.72	470.72

Table 4.7 Solution values for all GRASP cases for the reduced southeastern Brazilian system.

Table 4.8 presents the average and best solution values for the random and linear bias functions, and using fixed ($\alpha = 0.75$) and α reactive approach ($\delta = 5$). As in the previous case study, we can see that combining linear bias functions and α reactive approach ($\delta = 5$) results in the best average performance.

	$10^6 \text{US\$}$		$10^6 \text{US\$}$	
	$\alpha = 0.75$		α reactive	
	Avg.	Best	Avg.	Best
Random	430.88	430.88	428.38	424.62
Linear	430.88	430.88	427.97	422.60

Table 4.8 Results for random and linear bias, fixed and α reactive approach.

4.5 CONCLUSIONS

GRASP is a flexible metaheuristic that can be applied to general combinatorial optimization problems, requiring very little parameter tuning for each problem instance. Reactive GRASP allows self-tuning of the GRASP parameters along the iterative process, producing results uniformly better than those obtained by classical GRASP without any significant overhead.

In this paper, an enhanced GRASP with different sampling functions was successfully applied to the static power transmission network expansion planning problem. This non-linear mixed integer program is large-scale in real world applications, and two Brazilian expansion planning problems were analyzed and solved. For the first, and smaller, case study the best known solution was found in all of the reactive GRASP cases. In the second case study, the best solution was found in half of the reactive GRASP cases, those that use a linear bias function. For the other half of reactive GRASP cases, solution values only slightly higher than the best known solution were found.

These results, with two different power transmission network expansion planning problem instances, indicate the usefulness of these GRASP enhancements in solving real world combinatorial planning problems.

References

- [1] S. Binato, G.C. Oliveira, and J.L. Araújo. A Greedy Adaptive Search Procedure for Transmission Expansion Planning. Technical report, CEPEL, 1998. To appear in *IEEE Transactions on Power Systems*.
- [2] S. Binato and S.P. Roméro. Power Transmission Network Expansion Planning by a Hybrid Genetic Algorithm. In: *Proceedings of the IX Latin-IberoAmerican Congress on Operations Research (IX CLAIO)*, Buenos Aires, 1998.
- [3] S. Binato and S.P. Roméro. Genetic Algorithms with Genetic Engineering. In: *Extended abstracts of the Third Metaheuristic International Conference*, pages 63–68, Angra dos Reis, 1999.
- [4] J.L. Bresina. Heuristic-Biased Stochastic Sampling. In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 271–278, 1996.
- [5] C. DeChamps, J. Vankelecom, and E. Jamouille. TRANEX – An Interactive Computer Program for Transmission Expansion. *IEEE Summer Power Meeting*, paper A79 476-3, 1979.
- [6] O.I. Elgerd. *Electric Energy Systems Theory: An Introduction*. McGraw-Hill, 1971.
- [7] T.A. Feo and M.G.C. Resende. Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, 6:109–133, 1995.
- [8] R.A. Galego, A. Monticelli, and R. Romero. Transmission System Expansion Planning by Extended Genetic Algorithm. *Generation, Transmission and Distribution, IEE Proceedings*, 145:329–335, 1998.
- [9] R.A. Galego, R. Romero, and A. Monticelli. Tabu Search Algorithm for Network Synthesis. *IEEE Transactions on Power Systems*, 15:490–495, 2000.
- [10] R.A. Gallego, A.B. Alves, A. Monticelli, and R. Romero. Parallel Simulated Annealing Applied to Long Term Transmission Expansion Planning. *IEEE Transactions on Power Systems*, 12:181–187, 1997.
- [11] L.L. Garver. Transmission Network Estimation using Linear Programming. *IEEE Transactions on Power Apparatus and Systems*, PAS-89:1688–1687, 1970.

- [12] S. Granville, M.V.F. Pereira, G.B. Dantzig, B. Avi-Itzhak, M. Avriel, A. Monticelli, and L.M. V.G. Pinto. Mathematical Decomposition Techniques for Power System Expansion Planning – Analysis of the Linearized Power Flow Model using the Benders Decomposition Technique. Technical Report RP 2473-6, EPRI, 1988.
- [13] S. Granville, M.V.F. Pereira, G.B. Dantzig, B. Avi-Itzhak, M. Avriel, A. Monticelli, and L.M. V.G. Pinto. Mathematical Decomposition Techniques for Power System Expansion Planning – Decomposition Methods and Uses. Technical Report RP 2473-6, EPRI, 1988.
- [14] A. Monticelli, A. Santos Jr., M.V.F. Pereira, S.H.F. Cunha, J.G. Praça, and B. Park. Interactive Transmission Network Planning using a Least-Effort Criterion. *IEEE Transactions on Power Apparatus and Systems*, PAS-101:3919–3925, 1982.
- [15] G.C. Oliveira, A.P. Costa, and S. Binato. Large Scale Transmission Network Planning using optimization and Heuristic Techniques. *IEEE Transactions on Power Systems*, 10:1828–1834, 1995.
- [16] B.J. Parker, A. Tanabe, and M.T. Schilling. Accuracy of Linearized Power-Flow Model Simulation of the Brazilian Network. Technical Report 018/80, Eletrobrás, 1980.
- [17] M.V.F. Pereira, L.M. V.G. Pinto, S.H.F. Cunha, and G.C. Oliveira. A Decomposition Approach to Automated Generation-Transmission Expansion Planning. *IEEE Transactions on Power Apparatus and Systems*, PAS-104:3074–3083, 1985.
- [18] M. Prais and C.C. Ribeiro. Reactive GRASP: An Application to a Matrix Decomposition Problem in TDMA Traffic Assignment. *INFORMS Journal on Computing*, 12:164–176, 2000.
- [19] R. Romero, R.A. Gallego, and A. Monticelli. Transmission System Expansion Planning by Simmulated Annealing. In: *Proceedings of the 1995 Conference on Power Industry Computer Applications*, pages 278–283, Salt Lake City, 1995.
- [20] R. Romero and A. Monticelli. A Hierarchical Decomposition Approach for Transmission Expansion Planning. *IEEE Transactions on Power Systems*, 9:373–380, 1994.
- [21] R. Romero and A. Monticelli. A Zero-One Implicit Enumeration Method for Optimizing Investments in Transmission Expansion Planning. *IEEE Transactions on Power Systems*, 9:1385–1391, 1994.
- [22] B. Stott and J.L. Marinho. Linear Programming for Power System Network Security Applications. *IEEE Transactions on Power Apparatus and Systems*, PAS-99:837–848, 1979.

- [23] P. Tsamasphyrou, A. Renaud, and P. Carpentier. Transmission Network Planning: An efficient Benders Decomposition Scheme. In *Proceedings of the 13th Power System Computing Conference*, pages 487–494, 1999.
- [24] R. Villasana, L.L. Garver, and S.J. Salon. Transmission Network Planning using Linear Programming. *IEEE Transactions on Power Apparatus and Systems*, PAS-104:349–356, 1985.

5 TABU SEARCH FOR TWO-DIMENSIONAL IRREGULAR CUTTING

Jacek Błażewicz¹, Adrian Moret Salvador², and Rafał Walkowiak³

Institute of Computing Science
Poznań University of Technology
ul. Piotrowo 3A, 60-965 Poznań
Poland

¹ blazewic@put.poznan.pl

² adrianmoret@yahoo.es

³ Rafal.Walkowiak@cs.put.poznan.pl

Abstract: The two-dimensional irregular cutting problem consists of cutting figures of various shapes from a rectangular area to minimize the area required. We adapt the tabu search metaheuristic to the problem, defining a specific neighborhood and various types of moves and search strategies. A standard type of tabu search, with simple and combined evaluating functions as well as a probabilistic tabu search variants are considered. In the paper, the behavior of different probabilistic approaches and the advantage of a newly proposed random candidate list version over a simple ranking approach are demonstrated. An exact algorithm for finding a placement of the polygon has been used in order to enhance the quality of solutions. Preview of wide range of experiments performed gives an idea about the efficiency of different approaches proposed.

5.1 INTRODUCTION

The two-dimensional cutting problem is an optimization problem, where two-dimensional elements of arbitrary shapes are to be cut out of rectangular material. The objective is to minimize the amount of material used, based on a minimization of the length of the rectangular region (which has a predetermined width, e.g. a roll of material). The importance of the problem is growing due to its relation to packing, loading and partitioning. An extensive bibliography

about this problem can be found in [14]. All of these issues appear in various branches of industry and our motivation comes from studies performed for the steel industry [4].

Approximation algorithms based on a local search approach are used for solving a wide class of combinatorial problems. Such methods as simulated annealing, genetic algorithms and tabu search are called metaheuristics because they supply frameworks, that can be used to guide lower level heuristics designed especially for the problem. This paper is concerned with an application of one of these methods, i.e. tabu search [9, 10, 11, 12], to the two-dimensional irregular cutting problem.

We examine three variants of tabu search and report comparative results. The first approach is a simple short term memory version of tabu search with the usual concept of a tabu list and an aspiration function, which we apply to minimize a single criterion. The next variant introduces a new type of evaluating function which combines several criteria to be optimized and a tabu condition. In the last version a probabilistic approach is used which translates the evaluation criteria into probabilities of selection. In the paper, the behavior of different probabilistic approaches and the advantage of a newly proposed random candidate list version over a simple ranking approach are demonstrated.

The organization of the paper is as follows. In Section 5.2 the problem of cutting is formulated. Section 5.3 discusses the geometric issues and gives a special attention to a modified version of the algorithm described in [2]. In Section 5.4 the tabu search algorithm and its adaptation for the problem is presented. Section 5.5 is devoted to the detailed presentation of a classification and theoretical results concerning the probabilistic approach. Implementation problems are described in Section 5.6. The results and discussion concerning evaluation of the algorithm is given in Section 5.7. Final remarks about directions of further research complete the paper.

5.2 FORMULATION OF THE PROBLEM

The two-dimensional irregular cutting problem can be formulated as follows:

Given: a rectangular material (area) and a set of two-dimensional elements of arbitrary shapes; the shapes are described by the line-segment approximation and the material has a limited width; each element is characterized by its shape.

Minimize a length of the material used to cut out all the elements (which must be placed without overlapping).

The problem is NP-hard in the strong sense, since the simplest problem, i.e. one-dimensional cutting problem, is already strongly NP-hard [8]. A complete discussion of the complexity of the problem is given in [5].

Strong NP-hardness of the two-dimensional irregular problem implies the lack of a polynomial optimization algorithm (according to best current conjecture). Moreover, as far as we know, for the problem in question there are no algorithms with known worst case behavior bounds.

Methods that solve the problem are mainly heuristics. Approximation algorithms are often [1, 13] single run approaches without improvement stage and sometimes result in solutions of poor quality. Thus, a design of new methods constructing more satisfactory, especially in terms of quality, solutions is desirable.

In [6, 7] a special method has been proposed for solving the cutting problem. The algorithm is based on the idea of allocating figures to holes (i.e. unused areas of the allocation material). The sequential allocation of figures in order to create the feasible layout (cf. Figure 5.1) is followed by an additional improvement stage. The second stage uses a tabu search technique which has been specially adapted to create a shell for the geometric procedures, thereby ensuring the feasibility of solutions. Sequentially generated solutions depend on the previous ones according to the tabu guidance scheme. Experience gained while testing the implementation of the above method resulted in a new version of the algorithm. The method described in this paper varies from the previous one by introducing the following features:

- an optimal algorithm for feasible allocation of two polygons and
- new elements of tabu search policy.

The subsequent sections will describe these original elements of the method.

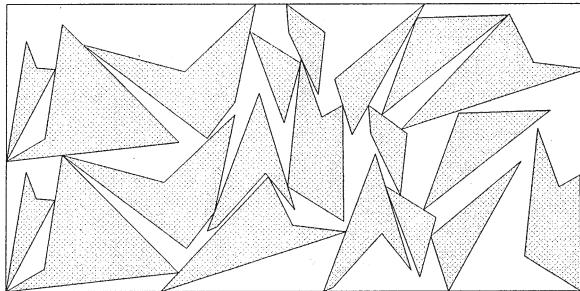


Figure 5.1 A cutting layout.

5.3 GEOMETRIC ISSUES

The allocation process consists in allocating elements to holes. The allocation of an element to occupy space within a hole is determined in the geometric phase of the algorithm, which is used to ensure the feasibility of these allocations.

The elements and the holes are polygons - “one piece” objects with no holes inside. There is no convexity restriction on the polygons. An algorithm for the mutual allocation of two polygons has been proposed in [2]. The algorithm computes an exact description of the set of all placements for a polygon (with d edges) which is free to shift and to rotate but not to intersect another polygon

(with e edges). The time complexity of the algorithm is $O(d^3e^3\log(de))$ and can be reduced to $O(d^2e^2\log(de))$ for the case when only shifts are allowed. The version of the algorithm for a fixed rotation angle has been implemented for our cutting method. While using it, the rotation step can be specified and as a result selected orientations of an element are taken into account while finding its possible allocations.

The algorithm mentioned is rather complex. Though the general version of the algorithm, where the procedure has to report all possible allocations for all continuously changed values of rotation angle, seems to be especially difficult, we encountered some difficulties even while implementing the simpler version of the algorithm (for a fixed orientation of figures). In some cases we could not use the original method. Additional erroneous allocations were reported while the dimensions (in the same direction) of both polygons were equal. To overcome the problem we introduced an improvement into the original algorithm.

The outputs of the algorithm are the sets of vectors defining different mutual allocations of polygons. A position of one polygon is fixed, while the other polygon is free to shift. Some sets of vectors represent internal or external allocations without intersections, while other sets represent allocations with intersections of elements' borders. (The internal and external allocations are understood here in the usual geometric sense.)

The figures are positioned in a co-ordinate system and vectors can be defined with their starting points in the co-ordinate origin. The end points of each set of vectors define different areas on the plane. To determine these areas the following three-step procedure is applied.

1. Defining contact-placements.

Contact-placements represent a set of end points of vectors used for allocating the elements to positions where a given pair, a vertex of one polygon and an edge of the second polygon, are in contact. These elements are defined independently for each vertex-edge pair and edge-vertex pair (the first element of a pair is taken from a fixed polygon, while the second from a shifted one).

2. Defining areas of intersection.

Parallelograms of the intersection are defined for pairs of edges taken from both figures. This structure represents the set of end points of vectors used for allocating the moved element to a position where any two edges of figures intersect. In the original version of the algorithm the parallelograms of intersection were defined independently for each pair of edges and then subtracted from all contact-placements to obtain the borders of required areas. This led to erroneous results. In some cases the method incorrectly classified as an allocation without intersection, an allocation external (one edge of a moved figure outside the area of a fixed one) for one pair of edges but internal (next edge of a moved figure inside the area of a fixed one) for the other pair of edges. Though the

intersection of a pair of edges did not arise – they were tangent at the end points – the intersection of figure borders occurred.

To solve the problem we combined the parallelograms of intersections (based on the same or the subsequent edges of a fixed figure) tangent along one edge to create a composed area of intersection containing common parts of edges. These common parts of edges represent the allocation of elements, when borders of elements intersect in the end points of element edges.

3. Subtraction of contact-placements and intersection areas.

Internal parts of areas (without borders) defined in step 2 are subtracted from all contact-placements and the set of sections of line is generated. These sections are combined (according to the original algorithm) to create borders of the requested areas.

The result of the procedure is visible in Figure 5.2. There exist three borders of non-overlapping allocation area - one border of an allocation area of elements outside each other and two borders of internal (one element inside the other) allocation of elements. The area restricted by these types of borders depict the intersection area. Notice that each point inside the area defines a vector starting in a coordinate origin, resulting in a specified type of allocation of elements.

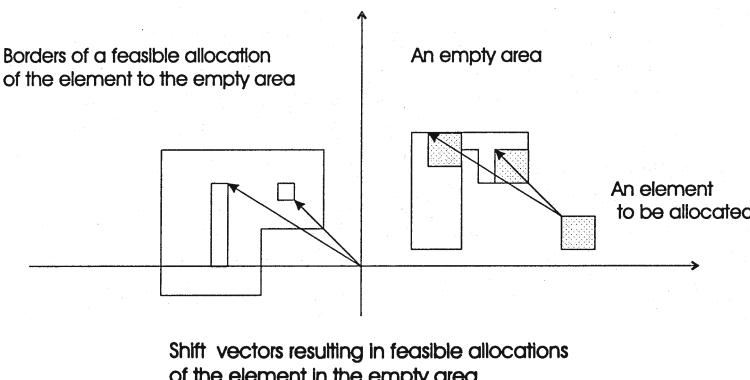


Figure 5.2 Borders of different allocation areas.

After the computation of all the possible allocations an element (figure) is placed. The position of the element is chosen according to one of the rules controlling the placement.

The allocation of the element consists in generation of a new description of the solution i.e.:

- adding allocation parameters of the element,
- updating the description of holes, and

- updating the geometric neighborhood relation for figures and holes.

The geometric neighborhood relation is used in the second (improvement) stage of the algorithm for an evaluation of solutions and a choice of elements to be removed.

The second stage of the method includes two types of operations: removing elements from a layout and their reallocation. The reallocation employs the same geometric operation as the allocation operation described above. The procedure used for removing elements is straightforward. It consists in eliminating an element from the list of allocated elements and updating solution data structures.

Finally we will define notions used in the next sections to describe features and an allocation of elements. An *external hole* is the area to the right of the rightmost allocated elements. An *internal hole* is any hole other than an external hole. An *external element* is an element that is a neighbor of the external hole (i.e., that lies on an edge in common with the hole). An *internal element* is any element other than an external element. The *waste for an element* is the sum of the area of holes surrounding the element.

5.4 TABU SEARCH

5.4.1 Neighborhood

To describe our tabu search procedure we introduce some basic elements of the method that influence the flow of the search process and the quality of the results. A basic concept in a search procedure is a neighborhood, where a solution is considered to be a neighbor of another solution if it can be reached from that solution in one step of the search, i.e., in a single “move”. Our neighborhood definition is based on the analysis of a cutting layout as a feasible solution to an instance of the cutting problem. The basic elements defining layout are figures, positions of figures, unused areas (holes) and the length of the layout. The only possibility of a change in the cutting layout is to change an element’s allocation. (We think here about the possibility of ensuring the correctness of solution’s parameters: the height of the layout used, the number of figures allocated and non-overlapping of figures.) As the elements cannot overlap each other, a move consists of two steps: removing elements and reallocating them to the unused area of the material. The moves can be described using different parameters. Some of them are given below:

- a number of figures moved,
- co-ordinates of a new allocation, and
- a characterization of the allocation area before and after a move.

The following types of moves can be defined:

- *slide* - a move inside the same hole (one figure),

- *insert* - a move allocating an element to a chosen hole (one figure),
- *exchange* - a move that reallocates two figures so that each occupies the area formerly occupied by the other, and
- *rebuild* - a move changing the allocation of a fixed number of figures (a given number of figures is removed from the solution and allocated again to new positions).

At different stages of the search process different moves can be used.

5.4.2 Evaluating function

The choice of a move (and the resulting new solution) is made using an evaluating function. The function has to compare the moves according to different measures of quality. The main parameter describing the quality of a solution is its length. As defined in the problem formulation the length of a layout is the objective function to be minimized. Different solutions may have the same values of the length of material used but their suitability for obtaining better solutions may vary significantly. Hence, to lead the search in a desirable direction, attention has to be paid to some additional features of a solution (different than the length of the material) and a move:

- maximum sum of the area of holes surrounding the figure to be moved,
- the left-most allocation of a figure,
- an area (and/or number) of holes created while allocating an element, and
- the length of a periphery and/or an area of the holes created while allocating an element.

Some or all of the above parameters may be chosen. Their values are mapped into positive numbers and combined as a weighted sum to create a final evaluation of the move.

The choice of criteria and weights to be used should be made according to the results of introductory search tests. It may happen that some of the parameters can be omitted, while others are significant.

This non-standard, including several parameters specification of the evaluating function, will probably enable a proper control of the search process, permitting additional information concerning geometrical features of solutions and a flow of the search process to be incorporated into the skeleton of the original method. The quality of the search control is important due to the high cost of each iteration, which involves extensive, geometric type, computation.

5.4.3 Tabu condition

An element of tabu search used to ensure an acyclic flow of the search is a tabu condition. This is a mechanism based on the idea of recording parameters of

solutions or moves made in previous stages of the search. This information is stored for a given period (number of iterations) and compared with the parameters of potential current moves and the solutions that would result from these moves. A tabu condition should forbid a move that reverses a move that is actually implemented or in general, should restrict the return to previously visited solutions. Maintaining a complete history of the search (such as parameters of moves and descriptions of solutions), even over a small number of iterations, would consume considerable memory and effort. Therefore, the design seeks to record critical attributes of moves and solutions rather than complete information. This may cause tabu conditions to forbid moves that lead to certain solutions not previously visited. This phenomenon can have a positive effect on the search, when it leads to avoiding solutions sufficiently similar to preceding solutions thus, making their examination unproductive. However, it can also potentially discourage a visit to solutions that are more attractive. To counter this latter possibility, aspiration criteria are used that allow sufficiently interesting solutions to be visited even if they would normally be forbidden. For example, one common aspiration criterion allows a solution better than the best previously found to be admissible to visit.

One way to implement a tabu condition is to embed it in the evaluating function. Penalties are used to discourage a tabu move, but in fact the move can be chosen if the remaining components of the evaluation are sufficiently attractive.

Information about the history of the search can also be useful for an intensification of the search in some parts of the solution space or a diversification of the search in order to visit different parts of the solution space. Adaptive short and long term memory functions are used to implement these processes, based on elements such as recency, frequency and sequential logic. A fuller description of tabu search and its practical applications may be found in [11].

5.4.4 Deterministic and probabilistic variants

In a deterministic form of tabu search a highest evaluation move (based on modifying the evaluating function to account for tabu conditions and aspiration functions) is chosen to obtain a new solution. Because of factors such as computational precision and imperfect choice of criteria, the resulting ranking of moves is an approximate one. This “level of noise” in evaluation motivates a probabilistic strategy that does not exclusively restrict attention to selecting the “best” move. Assuming that the choice criteria are not devoid of useful information, moves that are ranked more highly will be relied upon more heavily to guide the search in the right direction, but it is appropriate to assign some (smaller) probability to selecting moves whose evaluations are not as high.

The following questions arise. How can the evaluations be translated into appropriate probabilities of selection? Are the relative numerical evaluations important, or is the crucial information contained simply in the ranking of these evaluations? The answers to these questions depend on empirical analysis

according to the context. In the following approach we focus on an approach where the probability of a selection is based on the ranking of moves.

5.5 PROBABILISTIC APPROACH

The probabilistic approach used here is a variant of simple ranking probabilistic tabu search approach proposed in [11]. It proceeds in the following way:

Simple Ranking Method (SRM)

1. Evaluate the n neighbors of the current solution. Build a list whose elements are the non-tabu moves arranged by ranking of non-increasing qualities followed by the tabu moves also arranged by ranking of non-increasing qualities. Forbidden moves are not included in the list.
2. Accept the first candidate in the list as the next move with probability p_1 .
3. If not accepted, proceed down the list until a candidate is accepted. Accept candidate i with probability p_i .
4. If the list is finished without a selection, select the first (best) candidate in the list.
5. Make the move and update the tabu list.

A new way of implementing this approach is proposed. This strategy will be called *random candidate list method (RCLM)* and it is based on candidate list strategy described in [11] (pages 59-67). The candidate list is a subset of all solutions in the neighborhood considered.

The main difference between approaches is, that in our case, the candidate list is a subset of randomly chosen moves, on the contrary to a subset of evaluated solutions as it is assumed in the standard candidate list strategy.

Random Candidate List Method (RCLM)

1. Reduce the list of candidates by selecting a random subset of all the neighbors.
2. Evaluate the m candidates in this reduced list and arrange them as in the simple ranking method (SRM).
3. Select the best non-forbidden candidate from the reduced list.
4. If all candidates in the reduced list are forbidden or if the list is empty, evaluate all the neighbors that weren't included in the list and select the best one that is non-forbidden.
5. Make the move and update the tabu list.

The main aim of the application of **Random Candidate List Method (RCLM)** is to implement **Simple Ranking Method (SRM)** while decreasing the time cost of the search. It is possible to get the same probabilistic behavior of both methods. This depends on the way the reduction of the candidate list is done. This will be discussed in the following section.

A possibility of substitution of both methods comes from the fact that the probability assignment depends on the ranking of solutions on the contrary to assignments based on a value of the evaluation. The probabilistic behavior is the same but a number of evaluated solutions (and time requirements) can be significantly reduced.

Random reduction proposed here can be useful in our case. Problem considered is complex and irregular. It is not easy to elaborate a method of a “strategic” reduction of a candidate list. Such a complex procedure would be additionally burdened with computational cost and, as a consequence, it could become inefficient.

5.5.1 Probabilistic behavior of the considered approaches

In order to allow a more general analysis, we assume that in the set of neighbors there is a subset of forbidden solutions which will not be chosen regardless of their quality. This can happen due to different reasons: non-feasibility, tabu conditions, non-compatibility with the desired structure of the solutions with which the heuristic operates. Some of these facts may be impossible to check unless the solution is evaluated, and therefore this solution will be included in the list of candidates for evaluation even if it can't be finally chosen.

Let Y be the set of all neighbors (whole candidate list) and S be the set of candidates after reducing the list (the reduced candidate list). Let $n \in N$ be the cardinality of Y , $m \in N$ the cardinality of S , $k \in N$ the number of forbidden solutions in Y and let s_i be the i -th best candidate in Y , assuming its elements to be arranged in non-increasing order of qualities (tabu moves may be considered forbidden or just non-forbidden but worse than the rest of the moves). We will assume that forbidden solutions have quality equal to $-\infty$.

In order to achieve the reduction of the list of candidates in step 1 of the Random Candidate List Method (RCLM), different techniques can be used:

1. Technique 1: Fixing the number m of candidates in the reduced list.
2. Technique 2: Fixing the proportion $p = \frac{m}{n}$ of candidates in the reduced list.
3. Technique 3: Fixing the probability of including each candidate in the reduced list.
4. Technique 4: Fixing the probability distribution of studying m neighbors from the n in the neighborhood (with m taking values in $1, 2, \dots, n$).

Although these techniques might seem very similar, each of them leads to a different behavior of the RCLM in terms of probability. The probabilistic

behavior of the four proposed techniques which form a variant of the Random Candidate List Method and the behavior of the Simple Ranking Method with constant probability (SRMp) are described and compared below. We start our analysis with a more classical approach, i.e. Simple Ranking Method.

Simple Ranking Method with constant probability (SRMp)

After an evaluation, forbidden moves would be automatically rejected from the list, so we have a list with the $(n - k)$ non-forbidden candidates, i.e. $i = 1, \dots, (n - k)$.

The probabilities of selecting a candidate as the next move can be calculated easily as:

$$\begin{aligned} Pr[\text{Selecting } s(i) \text{ as next move}] &= Pr[\text{Rejecting } (s(1), \dots, s(i-1)) \\ &\quad \cap \text{Accepting } s(i)] \\ &= (1-p)^{i-1} \cdot p. \end{aligned}$$

This is true except for the best neighbor, which has an additional probability of being chosen when no others moves have been selected, that is, with an additional probability of: $(1-p)^{n-k}$.

In summary:

$$Pr[\text{Selecting } s(i) \text{ as next move}] = \begin{cases} p + (1-p)^{n-k} & \text{for } i = 1 \\ p \cdot (1-p)^{i-1} & \text{for } i = 2, \dots, (n - k) \\ 0 & \text{for } i > (n - k). \end{cases}$$

RCLM-Technique 1

Fixing the number m of candidates in the reduced list.

This is the technique used in the so-called random search if we set $m = 1$.

Assume that we arrange the n solutions by ranking of quality (best solution with rank 1, worst with rank n , respectively), taking into account that the quality of forbidden solutions is considered to be $-\infty$.

Figure 5.3 shows list Y once arranged in order of qualities. Each square represents one neighbor. Shaded squares are those which are included in the reduced list S . The forbidden candidates at the end of the list are surrounded by a bigger square.

For choosing s_i as the next move, it is necessary that it is the best in S , i.e., it is necessary that the $(m - 1)$ elements in S which are different from s_i are chosen between the $(n - i)$ elements which are worse than s_i . Thus, the number of possible cases where s_i is the best solution in the reduced list equals the number of combinations of $(m - 1)$ objects in a set of $(n - i)$, denoted by $C(n - i, m - 1)$. Note that, when $i > (n - m + 1)$, s_i will never be chosen, since in this case even the worst reduced list necessarily contains better candidates than s_i . The total number of possible lists equals $C(n, m)$. Thus:

$$\begin{aligned}
 Pr[s_i \text{ is the best in } S] &= \frac{C(n-i, m-1)}{C(n, m)} \\
 &= \begin{cases} m \cdot \frac{(n-i)!(n-m)!}{(n-i-m+1)!\cdot n!} & \text{if } i \leq (n-m+1) \\ 0 & \text{if } i > (n-m+1). \end{cases}
 \end{aligned}$$



Figure 5.3 List of solutions arranged in order of qualities.

Note also that the definitive selection will only take place if s_i is also non-forbidden, i.e. if $i > (n - k)$. Therefore:

$$Pr[\text{selecting } s_i] = 0 \text{ for } i > \min\{n - k, n - m + 1\}.$$

The best solution in the list will be additionally chosen by the method when the reduced list is composed only by forbidden moves (Case B). This can happen only in the cases where $k \geq m$. The number of lists which contain only forbidden moves equals $C(k, m)$, so:

$$Pr[S \text{ contains only forbidden solutions}] = \frac{C(k, m)}{C(n, m)} = \frac{k! \cdot (n-m)!}{(k-m)! \cdot n!} \text{ if } k \geq m.$$

Summing up these results, we can state that the distribution of probability for choosing each candidate in Y is given by:

Case A) If $k < m$

$$Pr[\text{selecting } s_i] = \begin{cases} m \cdot \frac{(n-i)!(n-m)!}{(n-i-m+1)!\cdot n!} & \text{for } i = 1, \dots, n-m+1 \\ 0 & \text{otherwise.} \end{cases}$$

Case B) $k \geq m$

$$Pr[\text{selecting } s_i] = \begin{cases} \frac{m}{n} + \frac{k! \cdot (n-m)!}{(k-m)!\cdot n!} & \text{for } i = 1 \\ m \cdot \frac{(n-i)!(n-m)!}{(n-i-m+1)!\cdot n!} & \text{for } i = 2, \dots, n-k \\ 0 & \text{otherwise.} \end{cases}$$

RCLM-Technique 2

Fixing the proportion $p = m/n$ of candidates in the reduced list.

As a result of the use of this technique, the number m of candidates studied in each iteration will be variable depending on the size of the neighborhood. Anyway, once the reduced list size is fixed, the analysis made for RCLM-Technique1 is still valid for this technique.

RCLM-Technique 3

Fixing the probability p of studying each candidate in the whole list.

The whole list is traced and each candidate is included in S with a certain probability. Let $p \in [0, 1]$ be the probability of including a candidate in the reduced list. The probability of having m candidates in the reduced list follows now a binomial distribution with parameters n and p . The mean of this distribution is equal $p \cdot n$, so the expected size of the reduced list (value of m) is still a proportion p of the whole candidate list as in the previous case studied.

However, the size of the list is now a random variable, and this leads to a different behavior of the method: the inclusion of a candidate in the reduced list is independent of the inclusion of others. Since the probability of the event that a given composition of a list has been obtained (i.e. best in the list candidate is chosen) is a product of probabilities of independent events that particular candidates are chosen or not.

A clear difference is that now any non-forbidden solution can be chosen, in opposition to the previous cases, where very bad solutions could never be chosen. When this technique is used, the following result holds:

$$\Pr[s(i) \text{ is the best in } S] = \Pr[s(i) \in S \setminus \{s(1), \dots, s(i-1)\} \subset (Y - S)] = p * (1-p)^{i-1}.$$

As in the previous technique, in the case that $s(i)$ is forbidden i.e. ($i > n-k$), it won't be chosen, and $s(1)$ will be chosen instead. The probability that S consists only in forbidden moves equals the probability that all non-forbidden moves are rejected from S , i.e.: $(1-p)^{n-k}$.

Therefore, as a result one gets finally the following distribution of probability:

$$\Pr[\text{selecting } s(i)] = \begin{cases} p + (1-p)^{n-k} & \text{for } i = 1 \\ p \cdot (1-p)^{i-1} & \text{for } i = 2, \dots, (n-k) \\ 0 & \text{for } i > (n-k). \end{cases}$$

RCLM-Technique 4

Fixing the probability distribution of studying m neighbors from the n in the neighborhood (with m taking values in $1, 2, \dots, n$).

This technique is a generalization of all the previous ones and it allows a wider range of probabilistic methods to be used as Reduction List Methods. Note that:

$$\begin{aligned} \Pr[s(i) \text{ is the best in } S] &= \Pr[s(i) \text{ is the best in } S \cap \|S\| = 1] \\ &\quad + \Pr[s(i) \text{ is the best in } S \cap \|S\| = 2] \\ &\quad + \dots + \Pr[s(i) \text{ is the best in } S \cap \|S\| = n]. \end{aligned}$$

We use the following notation: $a(i, j) = \Pr[s(i)]$ is the best in S under condition that $\|S\| = j$, $b(i) = \Pr[s(i)]$ is the best in S , and $x(j) = \Pr[\|S\| = j]$.

We have calculated the values of $a(i, j)$ when we dealt with technique 1 and we can write a diagonal system of linear equations: $\mathbf{Ax} = \mathbf{b}$

For a desired selection of \mathbf{b} (which is, in fact, a user-defined probabilistic method) there exists only one solution \mathbf{x} which may allow us to carry out the method in a reduced way. In order to carry out the reduction method, however, solution \mathbf{x} must additionally be a distribution of probability. If a solution to this system is a distribution of probability, we will call it a proper solution.

By adding all the equations we can conclude that the values of any solution \mathbf{x} to the system add up 1. Thus, the only additional restriction that \mathbf{x} must fulfill in order to be a distribution of probability is that $\mathbf{x} \in [0, 1]^n$, i.e., \mathbf{x} must be a convex combination of the vectors of the canonical base of \mathbb{R}^n .

Thus, it is easy to prove that a proper solution to the system exists if and only if \mathbf{b} is a convex combination of the column vectors in \mathbf{A} . This characterizes the probabilistic methods which can be carried out as RCLM by this technique. Note that not for all possible values of \mathbf{b} a proper solution to the system exists.

In summary we can state that SRMp and RCLM (while technique 3 is used) methods have the same probabilistic behavior.

5.5.2 Time cost of the Random Candidate List Method vs. Simple Ranking Method

We showed in the previous section that the simple ranking method with constant p (SRMp) has the same behavior as the Random Candidate List (RCLM) method if we use technique 3 (RCLM-Tech3), so the expected quality of the final solutions will be the same for both methods.

However, the RCLM will perform its iterations faster. We will analyze now the expected reduction in time cost for the evaluation of the neighborhood that the latter method (RCLM-Tech3) provides in comparison with SRMp.

The evaluation costs are generally much bigger than the rest of the operations made during an iteration: random elections, lists arranging, etc. We will not consider these small costs in our study. The expected time cost for evaluating a neighborhood is proportional to the expected number of neighbors which are evaluated. We will compare the expected number of evaluation of both methods:

The SRMp always evaluates n neighbors, so the expected value of evaluation is exactly n . We will calculate now the expected number of evaluated neighbors when using RCLM-Tech3.

Let us denote by:

A – the event: {There is at least one feasible solution in the reduced list S },

D – the random variable: {Number of neighbors which the method RCLM-Tech3 evaluates in an iteration},

$(-A)$ – the complementary event to A ,

m – the cardinality of S (now a random variable),

p – the probability of including each of the elements in Y in the reduced list S ,

$E[X]$ – the expected value of X , and

$E[X|A]$ – the expected value of X under a condition that event A occurs.

In order to calculate $E[D]$ we will need some previous results:

1. $E[X] = E[X|A] \cdot Pr[A] + E[X|-A] \cdot Pr[-A]$ for any random variable X and any event A .
2. m follows a binomial distribution with parameters n and p .
3. The distribution of $D|A$ is equal to the distribution of $m|A$.
4. $Pr[-A] = (1 - p)^{n-k}$.
5. $Pr[-A|m] = C(k, m)/C(n, m)$.
6. $m|-A$ follows a binomial distribution with parameters k and p .

The results 1 and 2 follow from basic theory of probability. From the way in which the RCLM operates sets, D is different from m only when $-A$ happens. This leads to the result 3. The equation in 4 is easy to deduct and it was stated in the previous section. Result 5 is calculated easily by basic combinatorial analysis. Result 6 is a direct consequence of results 2, 4 and 5.

Using the previous statements we can conclude:

$$\begin{aligned} E[D] &= Pr[A] * E[D|A] + Pr[-A] * E[D|-A] \\ &= Pr[A] * E[m|A] + Pr[-A] * E[D|-A] \\ &= E[m] - Pr[-A] * E[m|-A] + Pr[-A] * E[D|-A] \\ &= E[m] + Pr[-A] * (E[D|-A] - E[m|-A]). \end{aligned}$$

And now, using the expectation of the binomial distribution and the fact that given $-A$, the value of D is always n , we can conclude:

$$\begin{aligned} E[D] &= n \cdot p + (n - k \cdot p)(1 - p)^{n-k} \\ &= n \cdot \left(p + (1 - p \cdot \frac{k}{n})\right) \cdot (1 - p)^{n-k}. \end{aligned}$$

We will define the speed-up Sp of RCLM-Tech3 in relation to the classical method as:

$$Sp = \frac{\text{Expected time needed for a classical iteration}}{\text{Expected time needed for an iteration by RCLM-Technique 3}}.$$

Thus:

$$Sp = \frac{1}{p + (1 - p \cdot \frac{k}{n}) \cdot (1 - p)^{n-k}}.$$



Figure 5.4 An expected speedup as a function of p , ($n = 12, k = 6$).

Considering the expected speed-up as a function of p (cf. Figure 5.4) it can be seen that this function is continuous in $[0,1]$. Assuming additionally that $k < n$ (i.e., at least one of the neighbors is feasible) it can be proven by derivation that there exists only one point in the interval $[0,1]$ for which the expected speed-up is maximized. Notice also that for $p = 0$ and $p = 1$ the expected speed-up takes the value 1 regardless of the values of n and k .

Due to these properties of the expected speed-up, if p is set to a value different than the maximum speed-up point, there exists another value of p for which the expected time for the evaluation of the neighbors is the same. However, both methods will behave in a different way. When the value of p is smaller than the maximum speed-up point, the methods are more irregular, performing small searches in some iterations and a complete search towards the best candidate in others. The methods with p set to a bigger value will behave more smoothly and with less standard deviation from the expected time cost for an iteration.

5.5.3 General Random Candidate List Method

As it was pointed in Section 5.5.1, not every Simple Ranking Method (SRM) can be translated into RCLM by Technique 4 (the most general of the proposed techniques). We will now generalize the SRM and RCLM in the following way:

General random candidate method

1. Reduce the list of candidates by using Technique 4.
2. Evaluate the m candidates in this reduced list and arrange them as in the SRM.
3. Accept the first candidate in list S as the next move with probability $p(1)$.
4. If not accepted, proceed down the list until a feasible candidate is accepted. Accept candidate i with probability $p(i)$.
5. If the list is finished without a selection, select the first feasible candidate from the S .

6. If all solutions in S are forbidden, evaluate all Y and select the best solution in Y as next move.
7. Make the move and update the tabu list.

This general approach allows one to use RCLM in place of SRM, since it is also a generalization of the SRM. However, the extent of such a reduction depends on the method under consideration. Several choices of $p(i)$ and of the distribution of probability on m (number of candidates in the reduced list) can lead to the same probabilistic method. These methods may have a different level of expected speedup in relation with the SRM. A problem arises of how to choose $p(i)$ and the distribution on m which maximize the speedup of the method.

5.6 IMPLEMENTATION

The implementation of the method presented for the cutting problem has been written in C language. Various versions of tabu method were tested using Indy Silicon Graphic Indy workstation with 132 MHz R4600 processor and Cray T3E-900.

The initial solution may be generated in four ways: the elements may either be rotated or not, and allocated to all holes or only to an external one. Before an allocation, the elements are sorted according to their nonincreasing areas. The simplest way to allocate all the elements is to place them by a left-justified rule without rotation in a successively modified external hole. The most complex method consists of an allocation of the consecutive elements according to a left-justified rule with all available rotations to all currently existing holes.

At the improvement stage we use three types of moves: slide, insert and exchange. While selecting the allocation space, special attention is paid to *the waste for an element*, i.e. the area of holes surrounding an internal element. As a criterion for choosing an element to be removed we use the worst ratio of the element area (the smallest) and the waste (the biggest). Free space, created while an element is removed, is used to allocate one of the external elements. The element is allocated if the area and the shape of the hole allow for it. In this case the left-most allocation in the hole is chosen.

The moves mentioned above are evaluated by two types of the evaluating functions.

In the simple version of the evaluating function two parameters of a new solution are used: the length of a layout and the sum of area of all holes except for the external one. The main criterion is the length of the layout. Shorter layouts have priority over longer ones. When the layout lengths are equal, the latter parameter is used. The lower is the value of the parameter the more dense is the layout, and this solution is chosen. Finally, the tabu condition is taken into account and a solution obtained using the tabu move is rejected. However, when the solution is better than the best solution obtained so far, it is accepted (owing to the aspiration level function) in spite of the tabu condition.

The evaluating function of the second type is based on six parameters:

- a length of a layout - *Length*,
- a number of holes (unused areas) in a layout - *Holes*,
- a sum of a periphery of all holes - *Periphery*,
- a sum of ratios of an area and a periphery for small holes - *Ratio-s*,
- a sum of ratios of an area and a periphery for big holes - *Ratio-b*, and
- a tabu condition - *Tabu*.

As a small hole we define the hole which area is smaller than the area of the smallest element. The sum of an area and a periphery ratios for small holes is minimized, while for the rest of holes it is maximized. After an application of the moves, the values of all parameters are calculated. For each geometric parameter the evaluation of a move is made by using three current thresholds: the maximal, average and minimal values of a parameter. Four different evaluation values are assigned to the maximum, above the average, below the average and the minimum values of the parameter, respectively. The evaluation values are positive numbers possibly various for different parameters (see Table 5.1). The weighted sum of evaluation values of a move for all parameters gives a global evaluation value. The final ranking of moves is based on this evaluation.

Both evaluating functions are used as the two possible ways to control the search process. In the deterministic variant of the tabu search, the best evaluated moves are chosen in each iteration.

Both evaluating functions are also used in the probabilistic variant. At the first stage the SRMp was tested. The probability p (where p is a constant < 1) of choosing each move in the ranked (according to quality) sequence, conditional upon failing to choose preceding moves, is assigned to each move. For $p = 0.2$ the probability of choosing successively ranked moves is: 0.2, 0.16, 0.128, 0.1024, 0.08192, ... The value 0.1024 thus identifies the probability of choosing the forth-best move.

After receiving interesting results for SRMp subsequent tests were made using RCLM with Technique 3 in order to check experimentally time reduction of the computation.

In the construction of the tabu condition we didn't allow an element to be removed if it was moved in one of z previous iterations, where z is a fixed number adjusted experimentally. The condition is implemented in a form of two separate lists, one for each type of a figure (external and internal). The value of the parameter z depends on the number of elements of each type as well as on the number of all elements. For each figure we recorded information concerning restrictions on its movement. The figure cannot be moved in a number of iterations specified in the list. In the iteration in which the element is moved, the number of iteration z is assigned to the position of the list related to the element. After the iteration is executed the number is decreased by one.

5.7 COMPUTATIONAL RESULTS

The first part of this section is devoted to the presentation of the results obtained for different deterministic and probabilistic strategies of tabu search. In the latter part special attention is paid to the comparison of SRMP with RCLM approach.

Test data for the experiments have been prepared either manually by a division of a rectangular area (optimal solution is known for these test data) or randomly generated and adjusted to the input data format. Experiments have been made using 11 variants of figure shapes and cutting area dimensions. The number of figures varied between 10 and 28. The figures were convex or concave polygons with 3 to 7 vertices. In different data sets for all figures a lack of rotation or rotation step equal to 45 degree was applied. Waste is a parameter used for the final evaluation and the comparison of the solutions. For every solution the waste can be computed as a sum of unused area between the allocated elements as well as between the elements and the borders of rectangular area of an allocation.

To compare different variants of the method [3] three measures of quality W_i , W_i^{opt} and Q have been used. W_i is an average difference of layout lengths for solutions obtained by i -th method and the best layout length, given as a part of the best result value in %. W_i^{opt} is an average difference of layout lengths for solutions obtained by i -th method and the optimal layout length, given as a part of the optimal value in %. The values of W_i and W_i^{opt} below describe better methods:

$$W_i = \frac{1}{N} \sum_{j=1}^N \frac{l_{ij} - l_j^*}{l_j^*} 100$$

and

$$W_i^{opt} = \frac{1}{M} \sum_{j=1}^M \frac{l_{ij} - l_j^{opt}}{l_j^{opt}} 100,$$

where:

l_j^* - the shortest length of the layout for the j -th test data for all methods tested,

l_j^{opt} - the optimal length of the layout for the j -th test data,

l_{ij} - the shortest length of the layout obtained by i -th method for the j -th test data,

N - a number of test data sets, and

M - a number of test data sets with known optimal solution.

The quality parameter Q is based on the relative improvement in waste achieved from the initial solution (expressed as a percentage). The average of

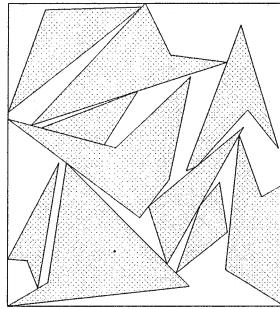


Figure 5.5 The best solution for randomly generated Test21 (width 12.0, length 11.0, waste $F_{ij} = 40, 9$) - method p5-s.

these values for the collection of problems is called the ‘quality improvement’ of the i -th method, Q_i . Higher values of

$$Q_i = \frac{1}{N} \sum_{j=1}^N \frac{I_j - F_{ij}}{I_j} 100$$

depict better methods, where:

I_j - the waste of an initial solution for the j -th test data, and

F_{ij} - the waste of the best solution found in the search done according to the i -th method for the j -th test data.

5.7.1 Comparison of deterministic vs. probabilistic approaches

Starting solutions were generated in four modes described in the previous section. Input data for the tabu search experiment consisted of the description of figures, cutting layout and starting solution. Twenty eight experiments for each input data set have been performed. In each experiment two deterministic and two probabilistic (SRMp) variants (for both, two different evaluating strategies have been used) of the method have been tested. We used three different probability assignments ($p = 0.2, 0.3$ and 0.5 , respectively) in the probabilistic approach. The exemplary solutions of problem instances are provided in the Figures 5.5 and 5.6. The end of the search was the result of compliance of one of among the following conditions:

- an execution of 40 iterations,
- a lack of improvement in subsequent 20 iterations, or
- a lack of new moves.

The evaluating function of the second type is based on parameters defined in Section 5.6. The evaluation values assigned to the parameters values are

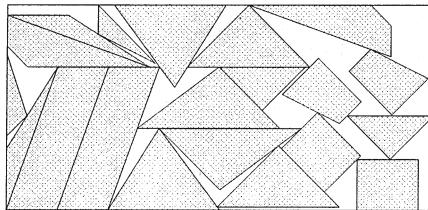


Figure 5.6 The best solution for manually generated Test7 (width 10.0, length 21.0, waste $F_{ij} = 23, 7$) - method p5-s.

given in Table 5.1. They have been defined in accordance with the natural importance of the parameters, and adjusted during introductory tests. This has to be done to assure proper level of restrictiveness of tabu condition.

<i>Evaluation values for different ranges of parameter values</i>				
Parameter	min	(min, average)	(average, max)	max
Length	9	6	3	0
Holes	3	2	1	0
Periphery	3	2	1	0
Ratio-s	3	2	1	0
Ratio-b	0	1	2	3

<i>Evaluation values for tabu condition</i>		
Number of tabu elements	Type of a move	Evaluation value
0	any	9
1	exchange move	6
1	other moves	0
2	exchange move	0

Table 5.1 An evaluation for different parameters of a solution.

The ability to improve the starting solution for deterministic and probabilistic variants of the algorithm is presented in Table 5.2. The table shows the number of

- searches with at least one improvement (column 2),
- best results for a given set of input data (column 3) - the input data set consists of the problem instance description and the description of the starting solution, and
- best results for a given instance of the problem (column 4)

that have been obtained using different variants of the method.

	- 1 -	- 2 -	- 3 -	- 4 -
Method	Number of searches with an improvement	Number of the best results obtained for different input data	Number of the best results obtained for different instances of the problem	
d-s	21	8	2	
d-c	20	7	4	
p2-s	23	1	0	
p3-s	20	2	1	
p5-s	20	6	4	
p2-c	21	1	1	
p3-c	17	3	1	
p5-c	17	3	2	
s		5	2	

Table 5.2 Results of the searches for different variants of the method.

In Table 5.2 the following notation is used for the types of tabu method:

- deterministic variants:
 - d-s** - with a simple version of an evaluating function
 - d-c** - with an evaluating function combining more parameters
- probabilistic variants with a simple version of an evaluating function:
 - p2-s** - probability equal to 0.2
 - p3-s** - probability equal to 0.3
 - p5-s** - probability equal to 0.5
- probabilistic variants with an evaluating function combining various parameters:
 - p2-c** - probability equal to 0.2
 - p3-c** - probability equal to 0.3
 - p5-c** - probability equal to 0.5.

In the row denoted by **s**, a number of times in which the starting solution was the best, has been given. None of the eight methods described could improve starting solutions for 2 out of 11 instances of the original problem in 5 out of 28 experiments (performed for different starting solutions). For randomly generated test data the optimal values of the layout length and the waste are not known. Hence, we can compare only the results obtained for different runs of the search procedure. In solutions obtained for such data the waste W has been between 6 % and 11 % of the area of the best cutting layout found.

Test data for which optimal solutions are known have optimal waste equal to 0. Hence, the waste computed for these results shows an absolute quality

of the solution. For this type of test data the waste W^{opt} has been between 44% and 51% of the area of the optimal solution. The optimal or sub-optimal solutions for test data with small values of an optimal waste are particularly hard to obtain.

Analyzing Table 5.2, one can state that the ability to improve a solution within the search process has been comparable for every method.

As far as the best results for different input data sets are concerned, two variants of the methods:

- deterministic approach with a combined evaluating function and
- probabilistic approach with a simple evaluating function with probability 0.5 of the best solution acceptance

obtained best results for 8 out of 9 problem instances for which the best starting solutions have been improved.

A comparison of the methods, using Q , W , and W^{opt} quality measures, is given in Figures 5.7 through 5.9.

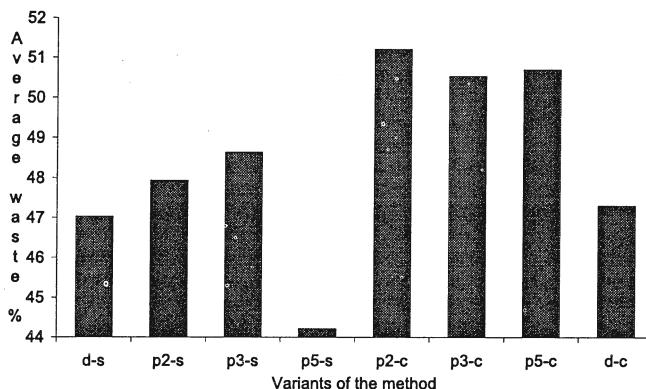


Figure 5.7 Average waste W^{opt} for different variants of the method.

There is no clear evidence that one class of variants: probabilistic, deterministic, with simple or complex evaluating function performs much better than other. Best results for different instances of the problem were obtained while using different versions of the method. Three variants of different classes: deterministic approaches with simple and complex evaluation function and probabilistic with factor 0.5 perform better than the others.

We have not observed any significant differences in the search time for different methods. The search runs presented above required from a few to some tens of minutes on Silicon Graphics Indy.

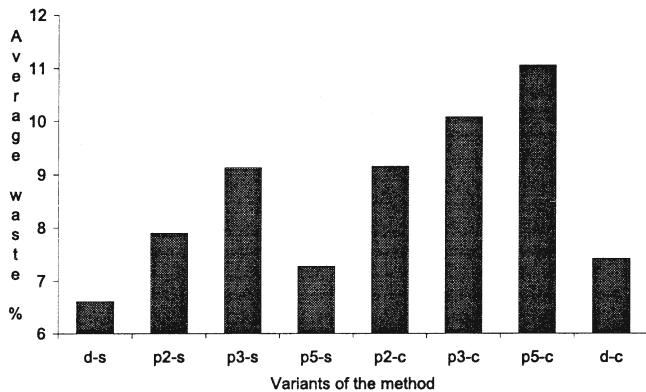


Figure 5.8 Average waste W for different variants of the method.

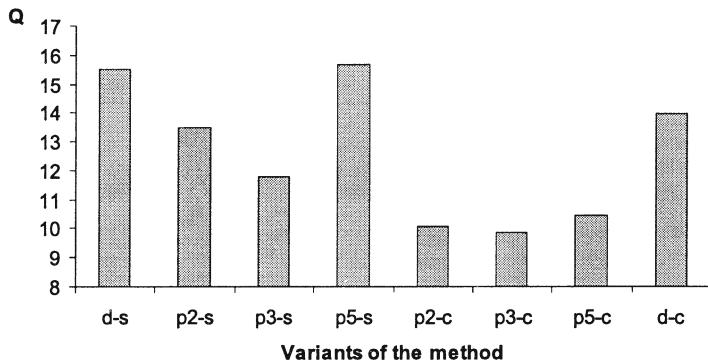


Figure 5.9 Quality Q for different variants of the method.

5.7.2 Comparison of probabilistic approaches

Interesting theoretical results considering the possible speedup of a computation (compare Section 5.5.2) while the random candidate method is used, forced a comparison of different probabilistic approaches.

A number of computational experiments were performed in order to compare the SRMp and RCLM-Tech3 approaches. All tests ran on a CRAY T3E computer. Results are presented for experiments performed for the same set of test data.

The parameter p refers to the probability parameter for both approaches. We considered ten values for this parameter, varying from 0.1 to 1 with steps of 0.1. $p = 1$ means that a deterministic tabu search strategy is being used. In all experiments a simple evaluation function was applied.

As stated above, two termination criteria were used based on a maximum number of iterations without improvement and a total maximum number of iterations. Both values were set to 20 and 40 respectively. This leads at first to different execution times for each variant of the method. However, in order to allow a fair comparison between the quality improvement of the different methods, the allowed time to solve each problem was fixed to the time needed by the method which stops first for the particular problem. Additional iterations of longer running time methods were not considered in the subsequent comparison.

First we compared (see Figure 5.10) the quality improvement (Q) of the classical probabilistic methods (SRMp) that run with 10 different values of p . In this approach the whole neighborhoods were evaluated. It can be seen that the deterministic method provides better results than this particular probabilistic method.

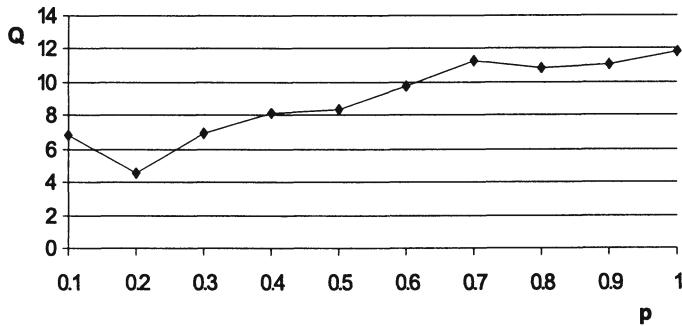


Figure 5.10 Quality for SRMp method as a function of p .

However, when the RCLM-Tech3 is applied, the time needed to evaluate the neighborhood by probabilistic methods decreases. Figure 5.11 shows the experimental average of the speedup which was achieved for the set of considered problems for different values of p :

$$\text{Speedup} = \frac{\text{Time needed for SRMp}}{\text{Time needed for RCLM-Tech3}}.$$

Note that when p decreases below 0.3, the speedup decreases. This result is related to the need of searching all the neighborhoods when no non-forbidden candidates were included in the reduced list or when the reduced list was empty. The strongest time reduction rate happens around $p = 0.3$. This value of parameter p is related to the maximum speedup point described in Section 5.5.2.

The reduction of time achieved by RCLM-Tech3 gives an advantage of this probabilistic method over the deterministic one. Figure 5.12 compares the quality improvement of the probabilistic (and deterministic, $p = 1$) methods when using the method RCLM-Tech3.

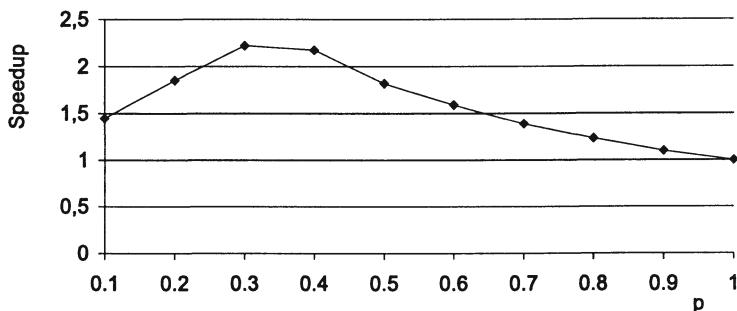


Figure 5.11 Speedup obtained for RCLM-Tech3 in relation with SRMp.

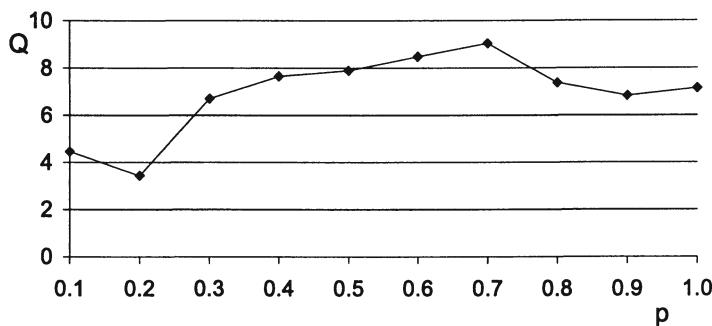


Figure 5.12 Quality for RCLM-Tech3 as a function of p .

These results show that for a wide range of values of p , the new random candidate list approach (RCLM) performs better than the deterministic method, on the contrary to the simple ranking probabilistic method (SRMp).

5.8 CONCLUSIONS

In this paper, a design and an implementation of a parallel tabu search based method for the two-dimensional irregular cutting problem has been described.

We have compared deterministic and probabilistic variants of the tabu search method which use two different forms of the evaluating function.

By comparing the results for different evaluating functions (and the same approach, i.e. either deterministic or probabilistic) one can see the advantage of the simple one-parameter based function over the composed one. This is an interesting result different from expectations. This situation may be a result of:

- difficulty in a proper choice of parameters,

- difficulty in aggregation of evaluations for different parameters (an aggregation function, assigning weights to parameters), and
- existence of conflicting criteria.

In the paper, different approaches for probabilistic tabu search have been also compared against each other and against the deterministic approach. The analysis has shown that both probabilistic approaches may have under some conditions the same behavior but different time requirements. The advantage of RCLM approach over a classical SRMp approach has been shown analytically and confirmed by experimental tests. The methods resulting from RCLM approach become more efficient than other probabilistic and deterministic variants of the tabu method.

Theoretical discussion and experimental tests made for various parameters of probability allowed for a global description of the behavior of probabilistic tabu search based methods for the two-dimensional cutting.

Acknowledgments: This work was partially supported by KBN grant number 8T11A01618.

References

- [1] A. Albano and G. Sapuppo. Optimal Allocation of Two-Dimensional Shapes using Heuristic Search Methods. *IEEE Transactions on Systems, Man and Cybernetic*, 10:242–248, 1980.
- [2] F. Avnaim and J.D. Boissonnat. Polygon Placement under Translation and Rotation. *Lecture Notes in Computer Science*, 294:322–333, 1988.
- [3] R.S. Barr, B.L. Golden, J.P. Kelly, M.G.C. Resende, and W.R. Stewart Jr. Designing and Reporting on Computational Experiments with Heuristic Methods. *Journal of Heuristics*, 1:9–32, 1995.
- [4] J. Błażewicz (editor). Computational Complexity Analysis of Algorithms for Automatic Cutting Layout Design (in Polish). Research report for OBR-PTK Tekoma, 1988.
- [5] J. Błażewicz, M. Drozdowski, B. Soniewicki, and R. Walkowiak. Two-Dimensional Cutting Problem: Basic Complexity Results and Algorithms for Irregular Shapes. *Foundations of Control Engineering*, 14:137–160, 1989.
- [6] J. Błażewicz, P. Hawryluk, and R. Walkowiak. Using a Tabu Search Approach for Solving the Two-Dimensional Irregular Cutting Problem. *Annals of Operations Research*, 41:313–325, 1993.
- [7] J. Błażewicz and R. Walkowiak. A Local Search Approach for Two-Dimensional Irregular Cutting. *OR Spektrum*, 17:93–98, 1995.

- [8] M.R. Garey, and D.S. Johnson. Strong NP-Completeness Results, Motivation, Examples and Implications. *Journal of the ACM*, 25:499–508, 1978.
- [9] F. Glover. Tabu Search - Part I. *ORSA Journal on Computing*, 1:190–206, 1989.
- [10] F. Glover. Tabu Search - Part II. *ORSA Journal on Computing*, 2:4–32, 1990.
- [11] F. Glover and M. Laguna. *Tabu Search*. Kluwer, 1997.
- [12] F. Glover, E. Taillard, and D. de Werra. A User’s Guide to Tabu Search. *Annals of Operations Research*, 41:3–28, 1993.
- [13] O. Gurel. Circular Graph of Marker Layout. Report No 320-2965, IBM Data Processing Division, New York Scientific Center, 1969.
- [14] P.E. Sweeny and E.R. Paternoster. Cutting and Packing Problems: A Categorized Application Orientated Research Bibliography. *Journal of the Operational Research Society*, 43:691–706, 1992.

6

A STUDY OF GLOBAL CONVEXITY FOR A MULTIPLE OBJECTIVE TRAVELLING SALESMAN PROBLEM

Pedro Castro Borges¹ and Michael Pilegaard Hansen²

¹FCEE - Portuguese Catholic University
R. Diogo Botelho 1327, 4100 Porto, Portugal
pmb@porto.ucp.pt

²Pilegaard Planning
Olof Palmes Gade 3, 5. tv. - 2100 Copenhagen, Denmark
mph@imm.dtu.dk

Abstract: *Global convexity, or heuristic concentration, has been mentioned in the literature as a reason for the success of many single objective metaheuristics. Global convexity is not an intrinsic characteristic of a combinatorial problem, but rather a property of the topology induced in a solution space by, for example, a neighborhood operator.* This paper presents a study of global convexity for a multiple objective travelling salesman problem using the 2-OPT operator. The analysis is based on local search and includes extensive experimentation with the weighted sums program and augmented Tchebycheff program. We study the concentration of local optima for instances of scalarizing programs, the differences observed between solutions and points generated with different weights, and the stability of the best local optima for small weight variations. The exact global optima for weighted sums programs are also generated and exhibit concentration characteristics. The results are promising in that they confirm the existence of global convexity and might be used as a basis for understanding and building new metaheuristics for multiple objective optimization problems.

6.1 INTRODUCTION: WHY DO METAHEURISTICS WORK?

The research in the field of metaheuristics has evolved much on the basis of trial and error, often motivated by competition for improving the best known solutions for given problems, and not as often by identifying the reasons for

the successes and failures. Some metaheuristics work well on some problems and are less successful in others. The quest for a *panacea*-like metaheuristic has been abandoned and it is now understood that it is not merely a question of which metaheuristic is used, but of how it is used.

The success of metaheuristics in single objective optimization ignited a rapid process of adaptation to multiple objective problems. This was reinforced by the fact that many multiple objective combinatorial problems are NP-complete even when their corresponding single criterion versions can be solved in polynomial time (Serafini [24]). Metaheuristic approaches can therefore be a pragmatic approach to the resulting computational intractability.

Besides important aspects of implementation and flexibility, the reason for the success of metaheuristics lies in the quality of their results. However, explaining the different degrees of success, and the failures, is not easy. Properties like the *ruggedness* of the cost-surface (Angel and Zissimopoulos [2]) or the existence of *heuristic concentration (global convexity, massif central)* have been proposed for explaining the differences in performance in single criterion optimization. We will concentrate our efforts on global convexity.

Global convexity is not convexity in the strict sense (Hu et al. [18]), but may be used to denote the empirical observation that the best local optima often are gathered in a very small part of the solution space, which hopefully includes the global optimum. Metaheuristics typically exploit this by concentrating the search in that part of the solution space.

Any assessment of global convexity only makes sense once a topology has been established in the solution space. A popular topology construction technique is based on neighborhood mapping, which consists in defining a set of neighboring solutions to each solution. Once such a mapping (also denoted a neighborhood function, or *neighborhood structure*) is established, a metric based on this mapping can be defined in the solution space. The distance between two solutions can be measured by e.g. the minimum number of times we need to apply the neighborhood mapping in order to obtain one of them from the other one.

In particular, when experimenting with heuristics based on local search what is tested is not how the generic search strategy performs on the problem, but how it performs on the problem given a particular neighborhood structure. It is that structure that defines the topology of the solution space, that will determine the characteristics of the cost surface, which is the cornerstone for optimization. A problem can be easy to solve with one neighborhood structure and intractable with another.

Global convexity has been studied and shown for e.g. the single objective travelling salesman problem (Boese [3]). This paper uses an approach based on the one of Boese et al. [4] but extends it to consider multiple objectives; a setting where the heuristic concentration concepts should include the effect of moving over different regions of the non-dominated frontier. The aim is to investigate global convexity over regions of the non-dominated frontier for the multiple objective travelling salesman problem by examining whether (exact or

potentially) non-dominated solutions that are close in the objective space or in a supporting weight space also are close according to a solution space metric. When and if global convexity exists, it may be exploited in metaheuristics for multiobjective combinatorial optimization.

Section 6.2 summarizes some basic concepts of multiobjective combinatorial optimization and of local search. Section 6.3 presents the multiobjective travelling salesman problem. After this, the computational results are presented. Section 6.4 examines a set of well-dispersed optimal supported non-dominated solutions in order to see whether global convexity exists for that subset of the non-dominated set. Section 6.5 essentially uses the experimental procedure of Boese et al. [4] on local optima. A fixed weight is used, and both the weighted sums program and the augmented Tchebycheff program are considered. A generalization of this investigation is presented in Section 6.6, where the entire weight-space is taken into account. Section 6.7 takes a closer look at a smaller local area in the weight-space to assess the stability of the local optima with small weight deviations. Before summarizing the conclusions, Section 6.8 suggests directions for further research and in particular, how global convexity can be exploited in (new) adaptations of well-known metaheuristics to a multiobjective environment.

6.2 BASIC CONCEPTS

6.2.1 Multiple objective combinatorial optimization (MOCO) problems

The general MOCO problem can be formulated as:

$$\min \mathbf{f}(\mathbf{x}) = \left\{ \begin{array}{l} f_1(\mathbf{x}) = z_1 \\ f_2(\mathbf{x}) = z_2 \\ \dots \\ f_J(\mathbf{x}) = z_J \end{array} \right\} \quad (6.1)$$

subject to $\mathbf{x} \in S$,

where \mathbf{x} is the decision vector, or *solution*, and S is a finite set of feasible solutions, *the solution space*. The objective function $\mathbf{f}(\mathbf{x})$ maps S into \mathbb{R}^J , where J is the number of objectives. The *image* of a solution in the decision space is a *point*, $\mathbf{z} = [z_1, z_2, \dots, z_J]$. A point, \mathbf{z} , is *attainable* if there exists a solution $\mathbf{x} \in S$ such that $\mathbf{z} = \mathbf{f}(\mathbf{x})$. The set of all attainable points is denoted Z .

The *ideal point*, \mathbf{z}^* , is defined as $\mathbf{z}^* = [\min f_1(\mathbf{x}), \min f_2(\mathbf{x}), \dots, \min f_J(\mathbf{x})]$. Normally, the ideal point is not attainable and there will therefore not exist a single optimal solution to the MOCO problem. Instead one must choose among *non-dominated* attainable points, where an attainable point, \mathbf{z} , is non-dominated if there does not exist another attainable point \mathbf{z}^+ which is better on at least one objective without being worse on others, i.e. where $z_j^+ \leq z_j$ on all objectives j .

A multicriteria decision making (MCDM) process may involve obtaining one, some, or all of the non-dominated points. For such tasks *achievement scalarizing functions* can be used for mapping points in the J -dimensional

objective space to an ordinal "scale of goodness" as defined by the scalarizing function.

This paper focuses on two families of scalarizing functions, namely the weighted sums and the augmented Tchebycheff scalarizing functions. These two families were chosen because they are used in the vast majority of MCDM literature.

The weighted sums scalarizing function can be written as follows:

$$s_{ws}(\mathbf{z}, \boldsymbol{\lambda}) = \sum_{j=1}^J \lambda_j (z_j - z_j^*), \quad (6.2)$$

where $\boldsymbol{\lambda} = [\lambda_1, \lambda_2, \dots, \lambda_J]$ is a *weight vector* with non-negative elements and \mathbf{z}^* is the ideal point.

The (weighted) augmented Tchebycheff scalarizing function can be written as:

$$s_{aTch}(\mathbf{z}, \boldsymbol{\lambda}) = \max_j \{\lambda_j (z_j - z_j^*)\} + \varepsilon s_{ws}(\mathbf{z}, \boldsymbol{\lambda}), \quad (6.3)$$

where ε is a very small (sufficiently small) positive scalar.

Achievement scalarizing functions are used to project the ideal point onto the non-dominated frontier through achievement *scalarizing programs* (Steuer [25]). Each instance of such a program uses a weight vector $\boldsymbol{\lambda}$, that defines the optimization direction towards the non-dominated frontier. The weighted sums program is as follows:

$$\begin{aligned} & \min s_{ws}(\mathbf{z}, \boldsymbol{\lambda}) \\ & \text{subject to } \mathbf{z} \in Z. \end{aligned} \quad (6.4)$$

The weighted sums program is often used in practice. But whereas all optima to (6.4) are non-dominated points, there might exist non-dominated points for which no weight vector exists so that the point minimizes (6.4). Optimal points to the weighted sums program are called *supported* non-dominated points.

The (weighted) augmented Tchebycheff program is defined as:

$$\begin{aligned} & \min s_{aTch}(\mathbf{z}, \boldsymbol{\lambda}) \\ & \text{subject to } \mathbf{z} \in Z. \end{aligned} \quad (6.5)$$

The augmented Tchebycheff program is useful because all non-dominated points can be found as optimum to (6.5) when an adequate weight vector is used just like any optimum of (6.5) will be a non-dominated point. For more on scalarizing functions, please refer to Wierzbicki [29].

Let the weight set Λ be defined as $\Lambda = \{\boldsymbol{\lambda} \in \mathbb{R}^J | \lambda_j \geq 0 \forall j \wedge \lambda_1 + \lambda_2 + \dots + \lambda_J = 1\}$. We will limit ourselves to the consideration of weight vectors $\boldsymbol{\lambda}$ in this set ($\boldsymbol{\lambda} \in \Lambda$). Furthermore, we will measure the distance between two weight vectors, $\boldsymbol{\lambda}^1$ and $\boldsymbol{\lambda}^2$, by their Manhattan distance:

$$d(\boldsymbol{\lambda}^1, \boldsymbol{\lambda}^2) = \|\boldsymbol{\lambda}^1 - \boldsymbol{\lambda}^2\|_1 = \sum_{j=1}^J |\lambda_j^1 - \lambda_j^2|. \quad (6.6)$$

Manhattan distance is also used to measure distance between points in the objective space:

$$d(\mathbf{z}^1, \mathbf{z}^2) = \|\mathbf{z}^1 - \mathbf{z}^2\|_1 = \sum_{j=1}^J |z_j^1 - z_j^2|. \quad (6.7)$$

A set $\Lambda' \subset \Lambda$ is *maximally dispersed* (with respect to the Manhattan norm) if $\min d(\lambda^1, \lambda^2) \geq \delta$ for all $\lambda^i, \lambda^k \in \Lambda', i \neq k$, and there does not exist any (larger) set $\Lambda'' \subset \Lambda$ such that $|\Lambda''| > |\Lambda'|$ and $\min d(\lambda^n, \lambda^m) \geq \delta$ for all $\lambda^n, \lambda^m \in \Lambda'', n \neq m$.

A maximally dispersed set can be constructed from any positive integer n , by including all the different weight vectors whose components are multiples of $1/n$ and which sum up to one. This set will have $\min d(\lambda^i, \lambda^k) = 2/n$, for $i \neq k$. The number of unique weight vectors is equal to:

$$\binom{J+n-1}{n} = \frac{(n+J-1)!}{n!(J-1)!} \quad (6.8)$$

6.2.2 Local search

A *neighborhood structure* is defined by an assignment of a set of neighboring solutions to each solution. The replacement of a solution with a solution from its set of neighboring solutions is a *neighborhood move*. Local search performs neighborhood moves in the pursuit of improvement and is a mechanism used by many metaheuristic search strategies, such as steepest descent, simulated annealing, tabu search, and many others.

The set of neighboring solutions to solution \mathbf{x} is denoted by $N(\mathbf{x})$. Solution \mathbf{x} is a *local optimum* of a function f if none of the neighboring solutions are better, i.e. if $f(\mathbf{y}) \geq f(\mathbf{x})$ for all $\mathbf{y} \in N(\mathbf{x})$. Solution \mathbf{x} is a *global optimum* if $f(\mathbf{z}) \geq f(\mathbf{x})$ for any feasible solution \mathbf{z} . A simple local search strategy, which locates a local optimum and which will be used in the experiments later on for the scalarizing programs, is the greedy randomized descent as outlined in Figure 6.1.

```

Generate a random starting solution, y
Repeat
    Set x equal to y
    Set M equal to N(x)
    Repeat
        Set y to a random solution from M and remove y from M
        until f(y) < f(x) or M is empty
    until f(y) ≥ f(x)
    Return locally optimal solution, x

```

Figure 6.1 Greedy randomized descent heuristic.

For a local search based heuristic to perform well in practice, its neighborhood structure must normally fulfill several criteria, many of which are problem specific. One criterion, however, that seems to be of general importance is that the neighborhood structure does not have a tendency to create deceptive cost surfaces. A deceptive cost surface is created if the global optimum is located far from the local optima. To evaluate this characteristic, and to study *heuristic concentration* a distance measure in the solutions space is required.

The distance from solution \mathbf{x}^1 to solution \mathbf{x}^2 can be measured as the minimum number of neighborhood moves needed to move from one to the other, i.e. as follows: If $\mathbf{x}^1 = \mathbf{x}^2$ then $d(\mathbf{x}^1, \mathbf{x}^2) = 0$, if $\mathbf{x}^2 \in N(\mathbf{x}^1)$ then $d(\mathbf{x}^1, \mathbf{x}^2) = 1$, else $d(\mathbf{x}^1, \mathbf{x}^2)$ is defined as one plus the length of the shortest list of solutions $(\mathbf{y}^1, \mathbf{y}^2, \dots, \mathbf{y}^k)$, such that $\mathbf{y}^1 \in N(\mathbf{x}^1)$, $\mathbf{y}^i \in N(\mathbf{y}^{i-1})$, $i = 2, \dots, k$ and $\mathbf{x}^2 \in N(\mathbf{y}^k)$. If the neighborhood structure is symmetric, i.e. if $\mathbf{y} \in N(\mathbf{x}) \Leftrightarrow \mathbf{x} \in N(\mathbf{y})$, then $d(\mathbf{x}^1, \mathbf{x}^2) = d(\mathbf{x}^2, \mathbf{x}^1)$.

6.3 THE MULTIOBJECTIVE TSP

We will consider the multiobjective travelling salesman problem (MOTSP):

$$\min \mathbf{f}(\mathbf{x}) = \left\{ \begin{array}{l} f_1(\mathbf{x}) = c_1^{x(n), x(1)} + \sum_{i=1}^{n-1} c_1^{x(i), x(i+1)} \\ f_2(\mathbf{x}) = c_2^{x(n), x(1)} + \sum_{i=1}^{n-1} c_2^{x(i), x(i+1)} \\ \dots \\ f_J(\mathbf{x}) = c_J^{x(n), x(1)} + \sum_{i=1}^{n-1} c_J^{x(i), x(i+1)} \end{array} \right\} \quad (6.9)$$

subject to $\mathbf{x} \in \wp\{1, 2, \dots, n\}$,

where n is the number of cities, \mathbf{c}_j is the cost matrix with respect to objective j , and the decision variable \mathbf{x} holds a permutation of the n cities. An often used neighborhood structure for the single objective TSP is the 2-edge exchange from the 2-OPT heuristic (Lin [21]) and consists of removing two edges from the tour and replacing them with the only two other edges that yield a feasible tour. The set of all such neighboring solutions to solution \mathbf{x} is $N(\mathbf{x})$.

The experiments optimize on the scalarizing programs (6.4) and (6.5). Notice, however, that while each weighted sums program (6.4) for the MOTSP will be a single objective TSP (Ulungu and Teghem [28]), program (6.5) will not, since the objective function is not linear.

For the experiments we chose a 100 cities MOTSP with 3 objectives, $J = 3$. The data was taken from the Krolak instances with 100 cities from TSPLIB (Reinelt [22]), namely kroA100, kroB100 and kroC100. Each cost matrix then corresponds to one objective in (6.9) so that kroA100 corresponds to the cost matrix of objective number 1, kroB100 to objective number 2, and kroC100 to objective number 3. One advantage of using these cost matrices is that the tours can be presumed to be within the same scale of range for the different objectives, making range scaling unnecessary. Also, the data sets are available and the optimal value to each problem is known, which gives us the exact ideal point \mathbf{z}^* .

As a surrogate measure for the distance between two solutions we will measure the similarity, or bonds, of two solutions by the number edges common for the two solutions. The bond distance as used by Kirkpatrick and Toulouse [20] is defined by n minus the number of edges common to both solutions. Boese [3] has developed a dynamic programming procedure to calculate the number of solutions at a given bond distance to an arbitrarily chosen solution. Using his software we computed these numbers for a 100 cities TSP, as shown in Figure 6.2.

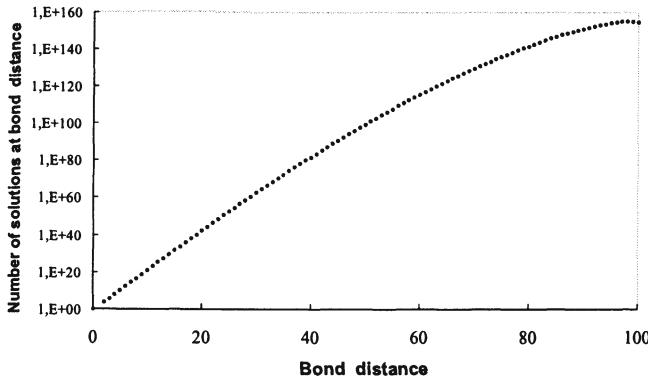


Figure 6.2 Number of solutions at different bond distances.

To provide the reader with an idea of magnitude one can mention that the mean bond distance is around 98 and the total number of solutions is around $4.66 \cdot 10^{155}$. Within “spheres” of bond distances 20, 30, 40 and 50 are located fractions of the total number of solutions of $4.61 \cdot 10^{-114}$, $8.82 \cdot 10^{-94}$, $9.89 \cdot 10^{-75}$, $4.31 \cdot 10^{-57}$, respectively. These figures give an idea of how large a proportion of the solution space actually are outside a “ball” with a given center and a given “radius”, as measured in bond distance.

6.4 A SET OF OPTIMAL (NON-DOMINATED SUPPORTED) POINTS

Let us first examine some optimal solutions to the weighted sums program (6.4). Using $n = 100$ we generated the $C_{100}^{102} = 5\,151$ maximally dispersed weight vectors, each defining a single objective TSP with 100 cities. With the software package `tsp_solve` (Hurwitz [19]) the problem instances were solved to optimality, giving 2\,150 unique points and (as it turned out) unique solutions. Unfortunately, we are not aware of algorithms for solving the augmented Tchebycheff problem to optimality within reasonable time limits. The optimal solutions to the weighted sums problems are primarily to be used in the experiments of Section 6.6, but some observed phenomena should be described here.

Each optimal solution is characterized by a generating weight vector λ , a solution \mathbf{x} in the decision space, and the point $\mathbf{z} = f(\mathbf{x})$ in objective space.

Figure 6.3 shows a contour plot of the frequencies of $d(\lambda^i, \lambda^k)$ and $d(x^i, x^k)$ considering all pairs of located optimal solutions (x^i, x^k) from all 5 151 weight vectors. It can be seen that optimal solutions belonging to close weight vectors also are located close in the solutions space. Also, the bond distances are surprisingly small, even for weights as different as 0.25 the global optima are generally within bond distances of 50. Only when the weights are completely different, corresponding to different single objective TSP instances, does the bond distances reach 100.

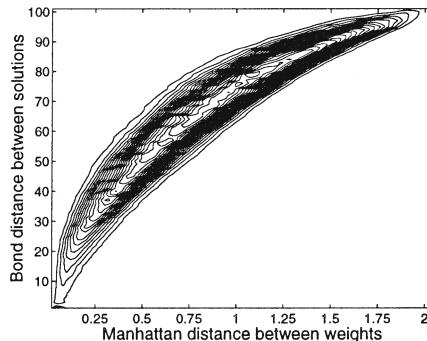


Figure 6.3 Difference in weights vs. difference in solutions for pairs of optimal solutions to the weighted sums program.

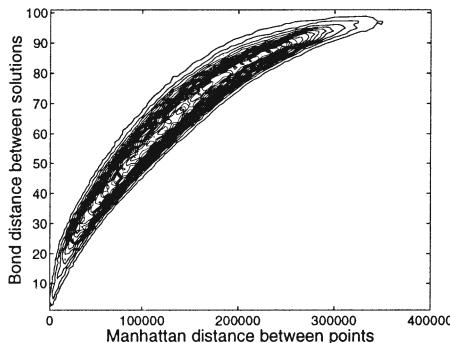


Figure 6.4 Difference in points vs. difference in solutions for pairs of optimal solutions to the weighted sums program.

In Figure 6.4 the frequencies of $d(z^i, z^k)$ and $d(x^i, x^k)$ are shown in a contour plot, considering all pairs of the 5 151 located optimal solutions (x^i, x^k) . We see how optimal solutions that are close on the non-dominated frontier of the (non-weighted) objective space also are close in the decision space. This is often important for the performance of multiobjective heuristics.

The general conclusion that can be drawn is that the solutions that are generated with similar weight vectors are also similar in decision space. It had

been observed that some MOCO problems have total *connectedness* among supported non-dominated solutions (Andersen et al. [1]). This is clearly a nice characteristic but it does, unfortunately, not hold for the non-supported non-dominated solutions.

6.5 HEURISTIC CONCENTRATION OF LOCAL OPTIMA

Finding the optimal solutions in the previous section was time-consuming and could not be done for the augmented Tchebycheff program. Often one has to use heuristics to approach problems which are larger or more difficult or for which sophisticated exact algorithms have not been developed, as was the case with the single objective TSP. To simulate such a situation and to be able to consider also the augmented Tchebycheff program, we used the greedy randomized descent heuristic with the 2-OPT neighborhood structure to locate local optima. The aim is now to investigate the distribution of the local optima in the decision space for a particular weight vector. The experiments in this Section are based on work by Boese, Kahng, and Muddu [4]. To make the explanation more clear one should remark that only local optima that were the final solutions (best solutions) of runs of greedy local search are analyzed, and not any intermediate solutions during those runs (the intermediate solutions are not locally optimal according to the 2-OPT operator).

The fixed weight vector of $(1/3, 1/3, 1/3)$ was used, and one thousand local optima were generated to both the weighted sums problem and the augmented Tchebycheff problem. The optimal solution to the weighted sums problem was also calculated.

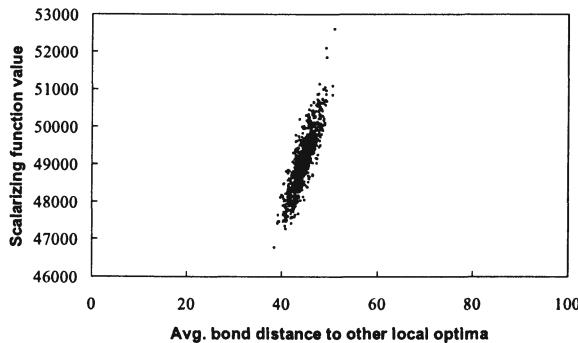


Figure 6.5 Scalarizing function values of 1000 local optima to the weighted sums program as a function of their average bond distance to the other 999 local optima.

For the weighted sums problem, Figure 6.5 shows for each local optimum the scalarizing function value as a function of the average bond distance to the other 999 local optima. Figure 6.6 shows for each local optimum the bond distance to the best of the local optima. (Note that for Figures 6.6, 6.8 and 6.9, random numbers between -0.5 and 0.5 were added to the bond distances

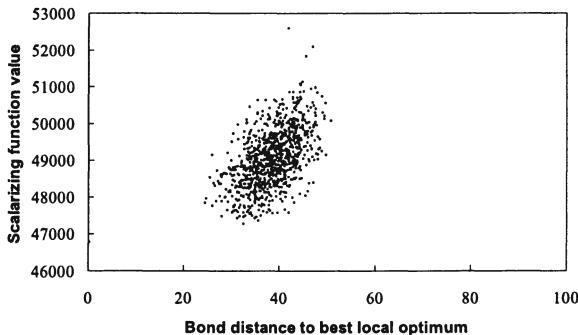


Figure 6.6 Scalarizing function values of 1000 local optima to the weighted sums program as a function of their average bond distance to the best local optima.

so the points do not describe lines at the original integer values but allow the reader to assess the intensity of the concentrations.)

Figures 6.7 and 6.8 show analogous graphs for the augmented Tchebycheff problem.

First it should be noticed that the bond distances again are rather small considering the vastness of the solution space, and that this is most evident for the weighted sums problem. This supports the idea of a *big valley* structure where the local search tends to converge to. A second observation regards centrality. One can see that local optima, which are closer to other local optima, also are better as opposed to those which are not so centrally located. In particular, the *best* local optimum has an average bond distance to other local optima of 38.3 (60.9), which is the lowest (6th lowest) value among the 1000 local optima to the weighted sums (augmented Tchebycheff) program. This means that the best solutions would be located centrally in such a valley. It looks like global convexity, or heuristic concentration, indeed exists. This is also the case with the Tchebycheff program, even if the bond distances here are significantly larger.

In global optimization it is also important, that the local optima are not located too far from the global optimum. Analyzing the differences between the global optimum for the weighted sums program and the 1000 local optima generated by local search can also be a way of controlling a potential bias of the 2-OPT operator. For the weighted sums problem, Figure 6.9 shows for each local optimum the scalarizing function value as a function of the bond distance to the *optimal* solution. The graph is similar to that of Figure 6.6 and the global optimum likewise seems to be located centrally among the local optima. Also, the best local optimum is closer to the global optimum as compared to the other local optima. The conclusion is that the best local optimum is indeed the most similar to the global optimum, which again is located centrally in respect to the local optima.

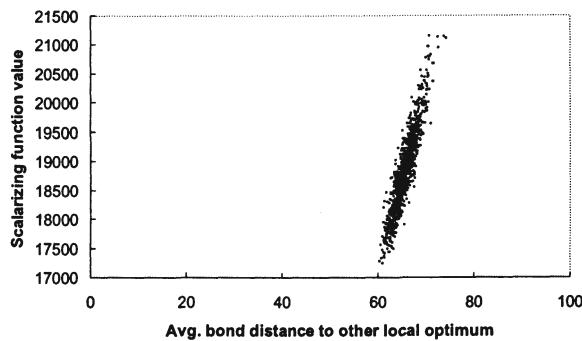


Figure 6.7 Scalarizing function values of 1000 local optima to the augmented Tchebycheff program as a function of their average bond distance to the other 999 local optima.

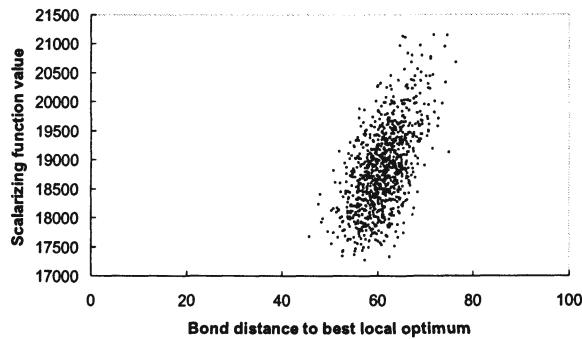


Figure 6.8 Scalarizing function values of 1000 local optima to the augmented Tchebycheff program as a function of their average bond distance to the best local optima.

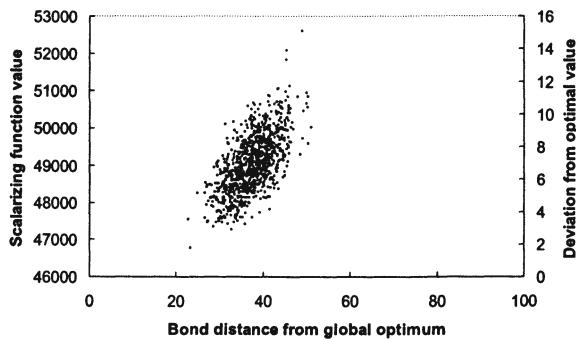


Figure 6.9 Scalarizing function values of 1000 local optima to the weighted sums program (and their deviation from the optimal value) as a function of their bond distance to the global optimum.

6.6 HEURISTIC CONCENTRATION OF LOCAL OPTIMA OVER THE ENTIRE NON-DOMINATED FRONTIER

The previous section analyzed the dispersion of local optima for a single optimization direction since only one weight vector was used. This section extends the analysis to cover the whole set of weights. One thousand local optima were generated for each of the 5 151 maximally dispersed weight vectors. This was done both for the weighted sums program and for the augmented Tchebycheff program.

Let us first take a look at whether the results of Section 6.5 are general for the whole non-dominated frontier. The convexity assumption would say that the local optima are gathered in a relatively small region of the solution space. The centrality assumption says further that the best of these local optima are located centrally with respect to the other local optima. If both assumptions are valid, we should also expect that if the local optima are gathered in smaller regions, then these should also be closer to the best local optimum. That this holds is seen in Figure 6.10 and Figure 6.11.

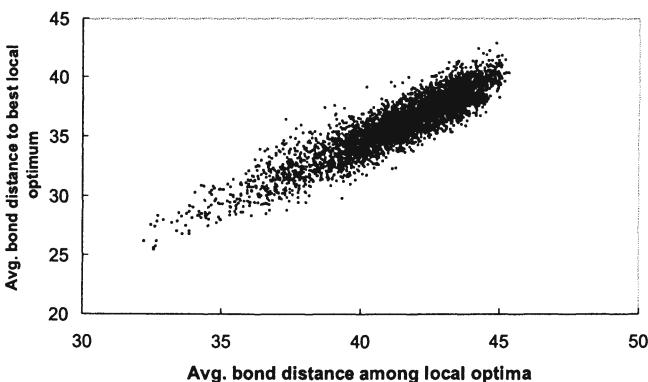


Figure 6.10 For each of the 5 151 experiments for the weighted sums program is plotted the relationship between the avg. bond distance among the local optima and avg. bond distance to the best local optimum.

From the global optimization point of view it would be nice if we could expect that if the local optima are gathered in small regions, then these are also close to the global optimum. Figure 6.12 shows that this indeed is the case for the local optima of the weighted sums programs.

Another interesting aspect is analyzing how often the best local optima for the weighted sums problem were the closest local optima to the global optimum. The proportion of best local optima which were the solutions closest to the respective global optima was 18% (of the 5 151 cases). In over 90% of the cases the best local optima were in the 10% local optima closest to the respective

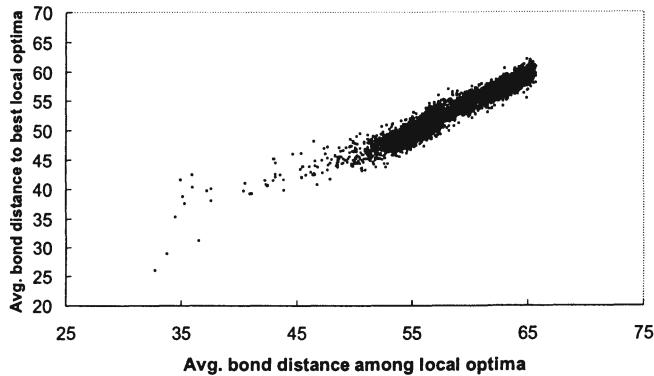


Figure 6.11 For each of the 5 151 experiments for the augmented Tchebycheff program is plotted the relationship between the avg. bond distance among the local optima and avg. bond distance to the best local optimum.

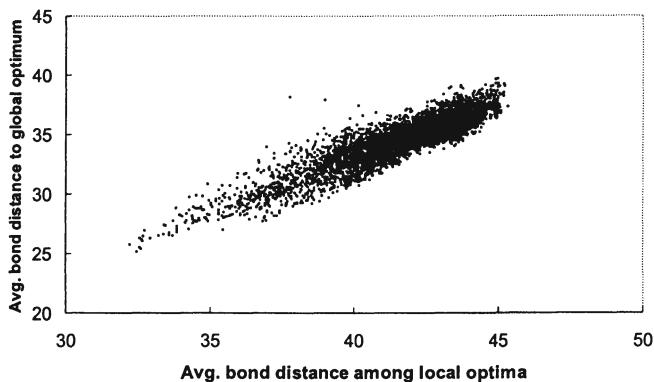


Figure 6.12 For each of the 5 151 experiments for the weighted sums program is plotted the relationship between the avg. bond distance among the local optima and avg. bond distance to the global optimum.

global optima. This indicates that in general the best of the local optima is closer to the global optimum than other local optima are.

Section 6.4 addressed the relationships between distances in weights, solutions and points for the global optima of the weighted sums program. One can also compare the same kind of contour plots for the best local optima both of the weighted sums program and of the Tchebycheff program. These are represented in Figures 6.13 to 6.16. The weighted sums program shows a similar pattern to the one observed for the optimal solutions, only the bond distances start at a higher value. The plots for the best local optima in the augmented Tchebycheff

program show, as expected, even higher bond distances. The higher spread of the Tchebycheff heuristic might be originated by the poorer performance of the heuristic itself. One can not exclude that if we had other heuristics for the Tchebycheff program then we could have results more similar to the ones found for the weighted sums program.

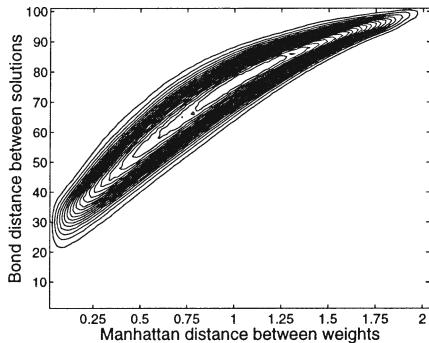


Figure 6.13 Difference in weights vs. difference in solutions for pairs of best local solutions to the weighted sums program.

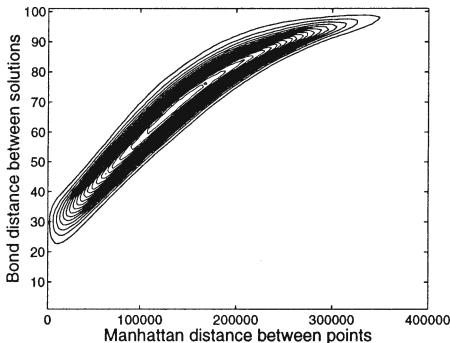


Figure 6.14 Difference in points vs. difference in solutions for pairs of best local solutions to the weighted sums program.

One can conclude that local optima are gathered in small regions of the solution space and that the best local optima are consistently central over the entire frontier. For the weighted sums program the global optima are central to those regions.

6.7 SENSITIVITY TO SMALL CHANGES IN THE WEIGHTS

Since the usage of achievement scalarizing functions where the weights are somehow modified is common in many MOCO procedures, it makes sense to examine the implications of small weight variations for the outcome of local

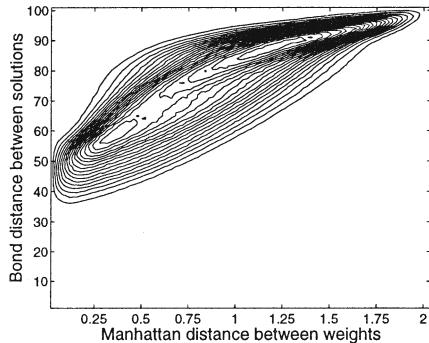


Figure 6.15 Difference in weights vs. difference in solutions for pairs of best local solutions to the augmented Tchebycheff program.

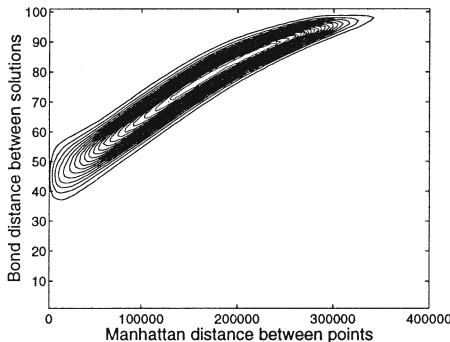


Figure 6.16 Difference in points vs. difference in solutions for pairs of best local solutions to the augmented Tchebycheff program.

search. The previous sections concluded that local optima obtained with similar weights tend to be close to each other but a more localized analysis was carried out. This approach consists in zooming in on a smaller area around a particular weight vector by generating successive sets of well dispersed weight vectors that are equidistant to that *central vector*, such that the distance increases for each set. The local optima found by using local search on the scalarizing programs with the spread weight vectors can afterwards be compared to the best local optimum (out of 1000 runs) found with the central vector.

Again we chose to zoom in on the area around the central weight $(1/3, 1/3, 1/3)$. To be able to examine what happens for different weight differences the nearby weights were generated for 100 different distances equally spread from 0.003 to 0.300. The weights at a fixed distance, or “radius”, from the central point measured with the Manhattan distance describe a hexagonal line in the weight space when projected on the 2D plane. A total of 1200 equidistant

weights vectors were generated for each fixed distance, which corresponds to having 200 equidistantly spread weights on each line segment of that hexagon.

For each weight vector one local optimum was found for both the weighted sums program and for the augmented Tchebycheff program.

The average scores obtained with the 1200 weight vectors in each equidistant set, measured with in the central scalarizing function, were compared. Figures 6.17 and 6.18 show the results. It seems clear that the scalarizing function values are rather constant for the weighted sums program when the weight vector only is changed marginally. This is not so for the augmented Tchebycheff program and it may indicate, that the weighted sums program is more robust with respect to uncertain or imprecise weights and therefore is easier to handle in practice.

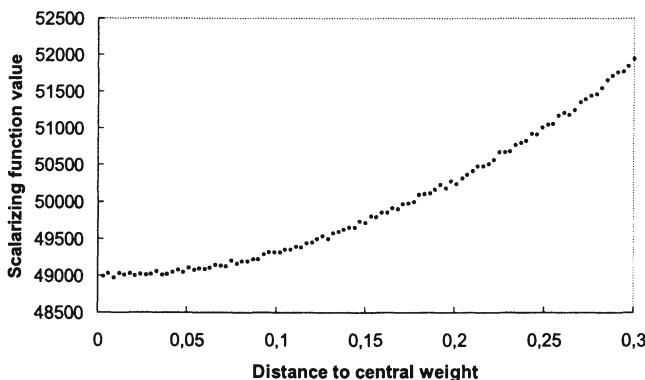


Figure 6.17 Average scalarizing function values for local optima at different distances to the central point in the weighted sums program.

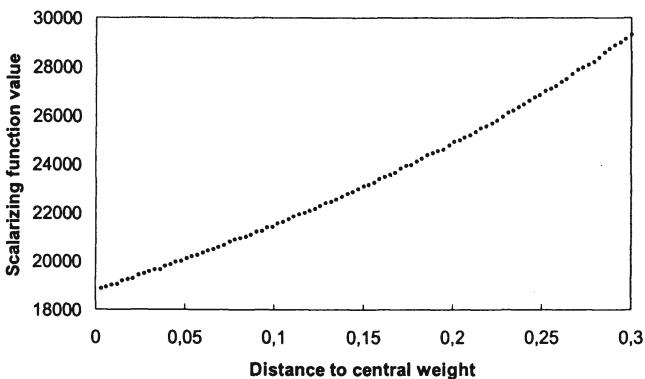


Figure 6.18 Average scalarizing function values for local optima at different distances to the central point in the augmented Tchebycheff program.

In Figures 6.19 and 6.20 the reverse is observed when the average distance in the objective space between the best local optima at the central weight (as found in Section 6.5) and the local optima with weights at small distances from the central weight is shown. But notice that the weighted sums programs start at a smaller distance and that it constantly keeps a lower level.

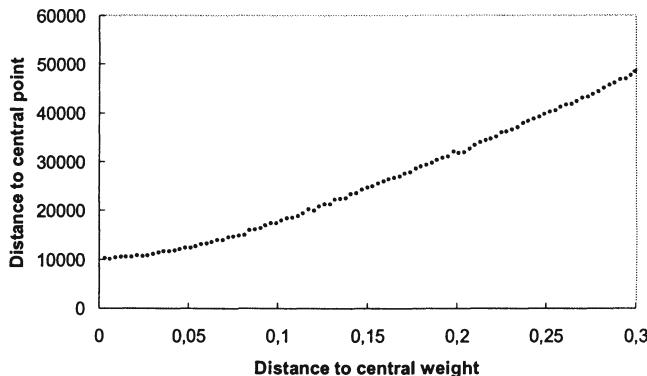


Figure 6.19 Average distance in objective space from local optima at given distances from the central weight to the best local optimum for the central weight for the weighted sums program.

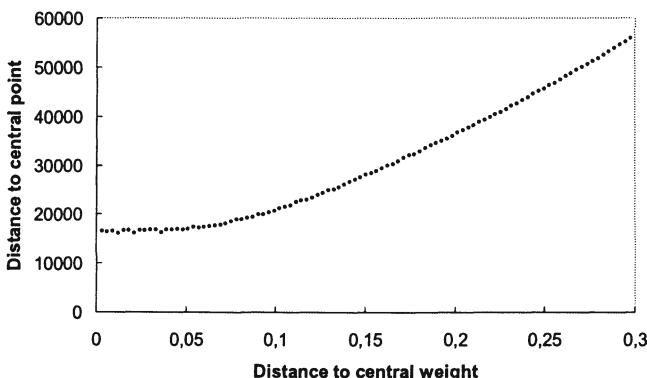


Figure 6.20 Average distance in objective space from local optima at given distances from the central weight to the best local optimum for the central weight for the augmented Tchebycheff program.

It is important to remember, however, that the weights are used in different programs so that the influence of small changes of the weights does not represent a direct comparison.

The experiments of this section gave the same general outcome when they were repeated with three other central weights, namely (0.4, 0.3, 0.3), (0.3, 0.4,

0.3) and (0.3, 0.3, 0.4), but with fewer local optima for each distance to the central weight.

6.8 IMPLICATIONS AND DIRECTIONS FOR FUTURE RESEARCH

The conclusions of this paper apply to the problem and problem instance used herein, even if the authors believe that the scope of those conclusions is more general. Naturally it will be beneficial to perform similar investigations on instances with correlated objectives since they are very common in practice. Other MOCO problems should also be considered.

Special attention should be given to MOCO problems of mixed structures, that is, MOCO problems where the objective functions belong to different problem classes. The MOCO problem of this paper is a TSP in all objectives, and the 2-OPT is recognized as a suitable neighborhood structure for this problem. If some objectives were not of the TSP structure, the 2-OPT may not be as well chosen and the global convexity not as obvious or non-existent.

The results of the paper are based on observed behavior rather than on theoretical analysis. More theoretical work would be beneficial in the field. Especially on the differences and similarities of the characteristics of scalarizing functions based on weighted sums and on the Tchebycheff function.

Most interesting would be the development of MOCO metaheuristics that can exploit global convexity whenever it exists. Instead of making direct translations of single objective metaheuristics in the sense of repeatedly optimizing scalarizing functions with different weights, one could try to use previously obtained information gained from the solution structure of obtained good solutions close (in the objective space or in the weight space) to the current one. This is perhaps most easily implemented with heuristics which consider not one, but a multitude of current solutions, as genetic algorithms (Goldberg [14]) and scatter search (Glover [13]).

Genetic algorithms combine known solutions into new solutions that are located “between” the known solutions. The term “between” is here based on the topology induced in the solution space as defined by the crossover operator. Whether the traditional building block viewpoint or the solution space topology viewpoint is more relevant depends on the purpose but it can be read from the work presented here, that the latter certainly has a usage in MOCO heuristics. Hybrid genetic algorithms as well as scatter search based heuristics adapted for MOCO problems may have a bright future and experiments with these is an obvious path for future research. Indeed, genetic algorithms have already been widely used for multiobjective problems (Horn [17]). (survey by Fonseca and Fleming [9]).

But also the traditional single solution metaheuristics like simulated annealing and tabu search (Glover [11, 12]) may be adapted for MOCO. Multiple solution adaptations for MOCO problems have been presented in Czyzak and Jaszkiewicz [7] and Hansen [15] for simulated annealing and tabu search, respectively. These methods could also be adapted to exploit global convexity.

Greedy randomized adaptive search procedures or GRASP (Feo and Resende [8]), Heuristic concentration (Rosing and ReVelle [23]) and Jump search (Tsumakitani and Evans [26]) are already prepared for exploiting global convexity, only they have not been adapted for MOCO problems. We see such an adaptation as very valuable for the field of MOCO metaheuristics. In particular, they may be used in single solution MOCO approaches such as NISE (Cohon [6]), MOSA (Ulungu et al. [27]), MOTS (Gandibleux et al. [10]) and CHESS (Borges [5]). This last approach explores the existing information by optimizing a function of the previously found points and focusing only on solution and objective spaces instead of weight space, not being bound to a particular direction at a time.

The type of analysis presented here is the genesis of what would be an ideal approach to building heuristics, for both single objective and multiple objective settings: On one hand to be able to characterize the topology of the solution space as induced by e.g. a given neighborhood operator; and on the other hand to describe the suitability of different metaheuristic approaches on different characteristics of topologies. Combining the two it should be possible, for the topology of a given problem instance, to find a suitable heuristic which exploits that topology. This paper only investigates some of the first part and much research remains to be done.

6.9 CONCLUSION

Even if generalizations cannot be claimed, this work casts new light into multiple objective optimization and might serve as a basis to build new multiple objective metaheuristics to exploit global convexity.

The main result is to show existence of global convexity for a multiobjective combinatorial optimization problem when using a common neighborhood structure. To be more precise, the experiments conducted in this paper indicate that heuristic concentration is present in a persistent manner over the entire area both of the exact (supported) and heuristically approximated non-dominated frontier for a MOTSP when using the common 2-OPT neighborhood structure. It is also illustrated how the non-dominated (or potentially non-dominated) solutions that result from the weighted sums and Tchebycheff programs vary with the possible weight vectors. This is observed in objective space, and in the solutions space, where distance is measured by a bond distance.

The association of those distances is far greater and promising than the authors had anticipated as relatively modest differences were observed in solutions for fairly different weights. We expected that the bond distances would go over 90 after only minor weight changes. Instead, we observed bond distances under 50 for the weighted sums program and under 70 for the augmented Tchebycheff program for fairly large changes in the weights. These distances correspond to very small fractions of the solutions space.

The results suggest that the augmented Tchebycheff program is harder to optimize than the weighted sums program, on the MOTSP using 2-OPT local optima. Hansen [16] found the same result in a tabu search implementation

based on 2-OPT on a MOTSP problem contained in the problems of that paper. It must be stressed, however, that the results are based on one MOTSP using the 2-OPT neighborhood structure. Other problems and other neighborhood structures could lead to other conclusions. (An illustrative example from Andersen et al. [1] shows how all supported solutions to the bi-objective spanning tree problem are connected with a minimal bond distance (of one edge) whereas this is not true for neighboring unsupported solutions.) In particular, the 2-OPT neighborhood search was used due to its previous successes in single objective TSP as for instance the weighted sums program. The augmented Tchebycheff program is different in nature and the 2-OPT might not be as effective for this program.

As a spin-off from the exhaustive computational experiments, we obtained a well-dispersed set of supported non-dominated solutions and two sets of potentially non-dominated solutions. These sets may serve as reference sets for experiments on approximation of the non-dominated frontier for this MOTSP instance.

It is our opinion, that many metaheuristics can be adapted to MOCO problems and, in cases where global convexity has a strong presence, this can and should be exploited in these adaptations. To say it in another way: metaheuristics for MOCO should be developed so that global convexity can be exploited.

References

- [1] K.A. Andersen, K. Jornsten, and M. Lind. On Bicriterion Minimal Spanning Trees: An Approximation. *Computers and Operations Research*, 23:1171–1182, 1996.
- [2] E. Angel and V. Zissimopoulos. On the Classification of NP-Complete Problems in Terms of their Correlation Coefficient. Submitted for publication.
- [3] K.D. Boese. *Models for Iterative Global Optimization*. PhD thesis, Computer Science Department, UCLA, 1996.
- [4] K.D. Boese, A.B. Kahng, and S. Muddu. A New Adaptive Multi-Start Technique for Combinatorial Global Optimizations. *Operations Research Letters*, 16:101–113, 1994.
- [5] P.C. Borges. CHESS: Changing Horizon Efficient Set Search. *Journal of Heuristics*, 6:405–418, 2000.
- [6] J.L. Cohon. *Multiobjective Programming and Planning*. Academic Press, 1978.
- [7] P. Czyzak and A. Jaszkiewicz. Pareto Simulated Annealing — A Metaheuristic Technique for Multiple-Objective Combinatorial Optimization. *Journal of Multi-criteria Decision Analysis*, 7:34–47, 1998.

- [8] T.A. Feo and M.G.C. Resende. Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, 6:109–133, 1995.
- [9] C.M. Fonseca and P.J. Fleming. An Overview of Evolutionary Algorithms in Multiobjective Optimization, Evolutionary Computation. *Evolutionary Computation*, 3:1–16, 1995.
- [10] X. Gandibleux, N. Mezdaoui, and A. Fréville. A Tabu Search Procedure to Solve Multiobjective Combinatorial Optimization Problems. *Lecture Notes in Mathematical Economical Systems*, 455:291–300, 1997.
- [11] F. Glover. Tabu Search — Part I. *ORSA Journal on Computing*, 1:190–206, 1989.
- [12] F. Glover. Tabu Search — Part II. *ORSA Journal on Computing*, 2:4–32, 1990.
- [13] F. Glover. Scatter Search and Star-Paths: Beyond the Genetic Metaphor. *OR Spektrum*, 17:125–137, 1995.
- [14] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.
- [15] M.P. Hansen. Tabu Search for Multiobjective Combinatorial Optimization: TAMOCO. *Control and Cybernetics*, 29:799–818, 2000.
- [16] M.P. Hansen. Use of Substitute Scalarizing Functions to Guide a Local Search Based Heuristic: The Case of moTSP. *Journal of Heuristics*, 6:419–431, 2000.
- [17] J. Horn. Multicriteria Decision Making. In: *Handbook of Evolutionary Computation*, T. Bäck, D.B. Fogel, and Z. Michalewicz, editors, pages 1–15, Oxford University Press, 1997.
- [18] T.C. Hu, V. Klee, and D. Larman, D. Optimization of Globally Convex Functions. *SIAM Journal on Control and Optimization*, 27:1026–1047, 1989.
- [19] C. Hurwitz. *tsp-solve Software Package*. Free software foundation. version 1.3.6, 1994.
- [20] S. Kirkpatrick and G. Toulouse. Configuration Space Analysis of Traveling Salesman Problems. *Journal de Physique*, 46:1277–1292, 1985.
- [21] S. Lin. Computer Solutions to the Traveling Salesman Problem. *Bell Systems Technical Journal*, 44:2245–2269, 1965.
- [22] G. Reinelt. TSPLIB. <http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/TSPLIB95/TSPLIB.html>, 1995.

- [23] K.E. Rosing and C. ReVelle. Heuristic Concentration: Two Stage Solution Construction. *European Journal of Operational Research*, 97:75–86, 1997.
- [24] P. Serafini. Some Considerations about Computational Complexity for Multiobjective Combinatorial Problems. *Lecture Notes in Economics and Mathematical Systems*, 294:222–232, 1987.
- [25] R.E. Steuer. *Multiple Criteria Optimization: Theory, Computations and Application*. Wiley, 1986.
- [26] S. Tsumakitani and J.R. Evans. An Empirical Study of a New Metaheuristic for the Traveling Salesman Problem. *European Journal of Operational Research*, 104:113–128, 1998.
- [27] E.L. Ulungu, J. Teghem, and P. Fortemps. Heuristics for Multi-Objective Combinatorial Optimization by Simulated Annealing. In: *Multiple Criteria Decision Making: Theory and Applications, Proceedings of the 6th International Conference on MCDM*, Q. Wei, J. Gu, G. Chen, and S. Wang, editors, pages 228–238, Beijing, 1995.
- [28] E.L. Ulungu and J. Teghem. Multi-Objective Combinatorial Optimization Problems: A Survey. *Journal of Multiple-Criteria Decision Analysis*, 3:83–104, 1994.
- [29] A.P. Wierzbicki. On the Completeness and Constructiveness of Parametric Characterizations to Vector Optimization Problems. *OR Spektrum*, 8:73–87, 1986.

7 A LOWER BOUND BASED META-HEURISTIC FOR THE VEHICLE ROUTING PROBLEM

José Brandão

Gestão e Administração Pública
EEG, Universidade do Minho
Largo do Paço, 4709 Braga Codex, Portugal
sbrandao@eeg.uminho.pt

Abstract: In this paper is described a tabu search algorithm for the vehicle routing problem, considering a fleet of K vehicles. The most interesting and innovative feature of this algorithm is that the starting solution, instead of being obtained from scratch or from some kind of well known heuristic method, as is the usual pattern, is given by a lower bound for this problem. This lower bound is a minimum cost K -tree with two K edges incident on the depot, and with two types of additional constraints representing the facts that each customer must be visited just once and that the capacity of the vehicles must not be exceeded.

7.1 INTRODUCTION

The vehicle routing problem (VRP) represents a situation where there is a set of customers geographically scattered, with known locations and demands that must be served from a depot. The VRP is concerned with the definition of a set of routes that are travelled by a fleet of K vehicles of the same size, in order to minimise the total travelling cost of these vehicles. Each route is constituted by a set of customers, whose demands do not exceed the capacity of the vehicle assigned to it, that are visited in a given a sequence, starting and finishing at the depot. The VRP with some side constraints has many applications in real life distribution of goods.

Most of the resolution methods that have been developed for the VRP are approximative ones, but there are also some exact algorithms, which can only solve small instances. Recently, Fisher [5] published an algorithm that was

able to find the optimal solution of a problem with 100 customers, and several other problems with between 25 and 71 customers. The quality of Fisher's [5] algorithm is mainly due to a very good method to determine a lower bound of a problem which, in general, is very near the optimum.

Fisher [5] formulated the VRP as a minimum cost K -tree, with degree $2K$ on the depot, with two other types of constraints: the degree two on each customer and the capacity constraints. The lower bound for the VRP is obtained by making a Lagrangian relaxation of these two types of constraints and then calculating the degree-constrained K -tree.

The tabu search algorithm for the VRP, presented in this article, has as its most distinctive feature the fact that the initial solution is given by a lower bound of high quality. In the beginning of this research, besides the lower bound produced by the minimum K -tree, was also experimented as lower bound the k -degree centre tree due to Christofides et al. [2]. This is a spanning tree with degree k on the depot plus K least cost edges, where $K \leq k \leq 2K$. So, the value of k taken is the one that gives the highest cost for the k -degree centre tree. After some tests, it was decided to abandon this line of research because the lower bounds were, in general, inferior to those generated by the K -tree. For the sake of comparison, the lower bounds produced by both methods are given in Section 7.2. The improvement of the lower bounds requires a mathematical formulation different from that given by Christofides et al. [2], taking into account the capacity constraints, as with Fisher's [5] method. On the contrary, both methods have in common the use of spanning trees as a component of the formulation, and they both consider the assumption that each customer must be visited exactly once.

The next section of this article is concerned with the mathematical definition of the lower bound and the details of its calculation, as well as the lower bounds for the test problems used for computational experimentation. Section 7.3 explains how the initial solution is constructed from the lower bound. The following section describes the tabu search method, and the final section is dedicated to the computational experiments and the conclusions.

7.2 LOWER BOUND

The notation for the VRP studied here is presented in the following:

- $N = \{1, \dots, n\}$ is the set of customers;
- $N_0 = N \cup \{0\}$, where 0 represents the depot;
- $E(N_0)$ = set of edges that link the vertices of N_0 , assuming a complete and undirected graph;
- q_i = demand of customer $i, i \in N$;
- K = number of vehicles;
- Q = capacity of each vehicle (they are all of the same size);

- c_{ij} = least cost of direct travelling between i and j ; $i, j \in N_0$. The VRP is assumed to be symmetric, i.e., $c_{ij} = c_{ji}$ for any $i, j \in N_0$; and
- $x_{ij} = x_{ji}$ is the edge that links i to j ; $i, j \in N_0$, and $x_{ij} \in \{0, 1\}$, $x_{ij} = 1$, if the edge (i, j) belongs to a solution and $x_{ij} = 0$, otherwise.

A K -tree is defined as a set of $n + K$ edges that span a graph with $n + 1$ nodes. The 0-tree is the usual spanning tree. The determination of the K -tree presents no special difficulty because Fisher [4] provides a polynomial algorithm for that purpose. Fisher [5] proofs that the VRP can be formulated through a minimum cost K -tree with degree $2K$ on the depot as is shown in the following.:

$$Z^* = \min_{x \in X} \sum_{ij \in E(N_0)} c_{ij} x_{ij} \quad (7.1)$$

$$\sum_{\substack{j \in N_0 \\ j \neq i}} x_{ij} = 2 \quad \text{for all } i \in N \quad (7.2)$$

$$\sum_{i \in S} \sum_{j \in \bar{S}} x_{ij} \geq 2r(S) \quad \text{for all } S \subseteq N \text{ with } |S| \geq 2, \quad (7.3)$$

where $x = (x_{01}, x_{02}, \dots, x_{0n}, x_{12}, \dots, x_{n-1,n})$ and $X = \{x|x = 0 \text{ or } 1 \text{ and defines a } K\text{-tree satisfying } \sum_{i \in N} x_{0i} = 2K\}$. For all $S \subseteq N$, $\bar{S} = N_0 - S$ and $r(S) = \lceil q(S)/Q \rceil$.

Constraints (7.2) impose that each customer is visited exactly once. Constraints (7.3) play simultaneously the role of the subtour elimination constraints and capacity constraints in other types of formulations, assuming that routes with single customers are not allowed.

In order to solve this problem, these constraints are dualized giving a Lagrangian relaxation, $R(u, v)$, that still corresponds to a minimum K -tree problem, dependent on two sets of dual variables, u and v . The model resulting from the relaxation is presented below:

$$R(u, v) = \min_{x \in X} \sum_{ij \in E(N_0)} \bar{c}_{ij} x_{ij} + 2 \sum_{i=1}^n u_i + 2 \sum_{S \subseteq N} v_S r(S), \quad (7.4)$$

where $u_0 = 0$ and

$$\bar{c}_{ij} = c_{ij} - u_i - u_j - \sum_{\substack{S \text{ such that} \\ i \in S, j \in \bar{S} \\ \text{or} \\ i \in \bar{S}, j \in S}} v_S, \quad (7.5)$$

where $v_S \geq 0$ for all $S \subseteq N$.

Since the multipliers u_i , $i \in N$, are associated with equations they can take any value in \mathbb{R} .

The value of $R(u, v)$ depends on the vectors of multipliers u and v , and because it is proven that $R(u, v) \leq Z^*$, the set of values (u, v) that maximise

$R(u, v)$ is searched. In order to try to achieve this goal the subgradient method is applied. However, the number of capacity constraints is $O(2^n)$, which means that, even for small problems, the total number of constraints is very large and so they cannot all be generated explicitly. Therefore, since a tiny fraction of all these constraints is used, the quality of the lower bound will depend very much on which constraints are considered.

The subgradient method was applied according to the formulas defined below, where t represents the current iteration:

i) Multipliers $u_i, i \in N$

$$u_i^{t+1} = u_i^t - f_t \left(2 - \sum_{j=0, j \neq i}^n x_{ij} \right) \quad (7.6)$$

$$f_t = \lambda_t (R(u^t, v^t) - Z^*) / \sum_{i=1}^n \left(2 - \sum_{j=0, j \neq i}^n x_{ij} \right)^2. \quad (7.7)$$

ii) Multipliers $v_{S_w}, w = 1, 2, \dots, C$, where C is the number of constraints in iteration t .

$$v_{S_w}^{t+1} = \max\{0, v_{S_w}^t - f_t (2r(S_w) - \sum_{i \in S_w} \sum_{j \in \bar{S}_w} x_{ij})\} \quad (7.8)$$

$$f_t = \lambda_t (R(u^t, v^t) - Z^*) / \sum_{w=1}^C (2r(S_w) - \sum_{i \in S_w} \sum_{j \in \bar{S}_w} x_{ij})^2. \quad (7.9)$$

In this research, the initial values of the parameters u, v and λ were the same used by Fisher [5], i.e., $u = v = 0$ and $\lambda = 2$, and the value of λ is multiplied by 0.75 after every 30 iterations without improvement. The value of Z^* is unknown, so it was replaced, in each iteration, by $1.04R^*$, where R^* represents the best known value $R(u, v)$, up to the current iteration. The computational experience shows that this value is not irrelevant for the convergence of the subgradient method. A high value in the first iterations makes the search very unstable in some test problems.

The main difference between our implementation and Fisher's [5] one comes from the capacity constraints. How these constraints have been defined in both cases is explained in the next paragraphs. In both articles, the set of constraints (each constraint corresponds to a set of customers) changes dynamically during the iterations of the subgradient method.

Fisher [5] used, simultaneously, the capacity constraints generated by two different methods:

- i) Initially are chosen $K + 3$ customers farthest from the depot as seeds. Each seed originates 60 sets of customers that contain the seed and a given number of the nearest customers from it. Therefore, the difference

among the sets comes from their cardinality, which is equal or greater than two, and from the combinations of customers.

- ii)* After iteration 50, each constraint that is violated by the Lagrangian solution is added to the overall set of constraints. In this case, a constraint is a set of customers corresponding to a connected component. This is obtained by deleting in the graph corresponding to the Lagrangian solution all edges incident on the depot.

During the resolution, a constraint (S) is said to be inactive, in a given iteration, if $v_S = 0$ in that iteration. Fisher [5], after iteration 10, starts deleting all the constraints that had been inactive during 3 iterations.

In this article, we should distinguish two experiments that correspond to two different ways of producing the constraints:

A – The capacity constraints used are those that are violated in each iteration. Therefore, initially the global set of capacity constraints is empty, but in the following iterations every constraint violated is added to that set. So, this corresponds to method *ii* of Fisher [5] (but he uses at the same time method *i*), except that here the process starts in the first iteration.

B – In this case, the global set of capacity constraints is given by the method A plus the constraints that result from the resolution of the VRP in each iteration of the subgradient method. The VRP is solved by a tabu search method that allows for infeasible solutions and, in the end, each route of the solution that is infeasible is added to the global set of constraints. In order to spend a very short time, only a few iterations of the tabu search method are executed.

Both for methods *A* and *B*, the constraints that had been inactive during 2 iterations of the subgradient method, after iteration 300, are deleted.

7.2.1 Computational testing

All the programmes of this article (for the lower bound and for the VRP) have been written in C by the author and executed on a Pentium II at 350 MHz.

The set of eight test problems taken in this research is the same as that one used by Fisher [5], except four problems that are not used because they are rather small and the data are only available from the author. Originally, the first six problems are given by Christofides et al. [1] and the last two are real problems given by Fisher [5]. The main characteristics of these problems are given in Table 7.1, where the capacity ratio stands for $\sum_{i \in N} q_i / KQ$.

In relation to the lower bounds given by the k -degree centre tree (k -MST) and presented in Table 7.2 we assumed that $k = K$. In the same table are presented the lower bounds given by the K -tree. The immediate conclusion that can be drawn from Table 7.2 is that the lower bound given by the K -tree (with degree 2 constraints) is better than the one obtained with the k -degree centre tree, except for three problems. However, it should be noticed that the

K -tree lower bound will be further improved by the introduction of capacity constraints. This is the reason why we did not proceed with the k -degree centre tree.

Problem	n	K	Capacity ratio
1	50	5	0.97
2	75	10	0.97
3	100	8	0.91
4	150	12	0.93
5	199	16	0.9956
6	100	10	0.91
7	71	4	0.96
8	134	7	0.95

Table 7.1 Characteristics of the test problems.

Problem	Minimum spanning tree	k -MST	k -MST + degree-2 constraints	K -tree	K -tree + degree-2 constraints
1	376.49	381.87	481.99	445.41	486.02
2	472.33	497.66	687.15	636.55	699.35
3	562.26	582.03	746.63	673.25	757.58
4	635.28	670.76	869.58	805.96	882.12
5	691.99	750.20	1009.73	932.98	1015.62
6	417.30	482.41	729.73	635.94	663.28
7	150.16	159.09	228.77	187.93	205.64
8	615.03	632.71	788.13	688.97	762.49

Table 7.2 Lower bounds given by the k -degree centre tree and the K -tree.

It is worth noting that for problem 5 the K -tree with degree 2 constraints on customers gives a lower bound of 932.98, while Fisher [5] presents the value of 958.89. Since these two values should be equal and since our data had been checked thoroughly, we believe that there had been a typing error in Fisher's [5] paper or his data is wrong.

The results of Table 7.3 are based on the execution of 1200 iterations (using, for example, 2000 or even 3000 iterations provided no better solutions) of the subgradient method, for the capacity constraints generated by methods A and B. The information presented in Table 7.3 is the following: lower bounds given by methods A and B, maximum number of active constraints for methods A and B, number of seconds of CPU time when method A is used, Fisher's [5] lower bounds, the cost of best known solutions for the VRP, and the ratio between this cost and the lower bound given by A.

Problem	A			B		Best known VRP		
	Cost	Active constraints	CPU time (s)	Cost	Active constraints	Fisher	solution	Ratio
1	509.52	145	13	505.28	170	507.09	524.61	0.97
2	757.17	233	34	750.54	231	755.50	835.26	0.91
3	791.02	178	36	787.93	233	785.86	826.14	0.96
4	939.39	269	135	934.36	307	932.68	1028.42	0.91
5	1123.53	364	307	1118.35	385	1096.72	1310.08	0.86
6	790.04	230	32	802.63	227	817.77	819.56	0.96
7	219.96	75	9	219.90	67	237.76	241.97	0.91
8	1077.95	194	90	1010.20	203	1133.73	1162.96	0.93

Table 7.3 Lower bounds given by methods A and B.

From Table 7.3 can be concluded that method A performs better than method B, except for problem 6, in spite of the total number of capacity restrictions and the maximum number of active ones being higher, except for problems 6 and 7. Another conclusion is that Fisher's [5] way of calculating the lower bound gives better results for 3 problems and worse for 5 problems, but the average difference is quite small.

Since method A gives better results than method B, the lower bounds for the tabu search algorithm presented in this article are calculated by method A.

7.3 STARTING SOLUTION

In a minimum cost K -tree there are in general K edges that are useless for the VRP. The only exception happens when in a connected component the depot and all the customers have degree 2. In this case, the connected component corresponds to a route. Therefore, to create an initial VRP solution from a VRP lower bound, the first step has been the definition of a minimum cost spanning tree (MST) (in fact no further calculation of the MST is needed because, due to the way that the K -tree has been represented in the programme, the MST and the K edges to delete are separate, and can immediately be identified), in the graph constituted by the edges of the minimum cost K -tree, and delete the other K edges. Deleting in this MST the edges incident on the depot, a given number of connected components is obtained. In this research, four different ways of converting each of these connected components into routes have been used, as is described by the following.

Version 1 – Each connected component generates a VRP route. The order of the customers in the route is the same as they entered to form the MST that always starts with the depot. This order is implicit in the representation of the K -tree used in the programme. The main shortcoming of this procedure is that it puts in sequence customers that are not in that sequence in the MST, because the sequence is only defined if a customer has degree 2. If a customer has a degree greater than two, any of the customers linked to it, and not yet in the route, can be the next to enter. Therefore, the rule followed here is quite

arbitrary. The other versions are different ways of making this choice in a more rational way.

Version 2 – Each connected component is broken into segments or chains at the customers with a degree greater than two. These customers do not belong to any of the pieces, each one originating a segment with one customer. Then each of these segments gives rise to a route. This procedure keeps a good number of edges of the lower bound, but also produces a number of routes substantially higher than K . The objective of the next two versions is to reduce the number of routes and, at the same time, to use more edges of the lower bound.

Version 3 – For each connected component, the first chain starts at the depot and follows the sequence of customers (defined by the component) until one appears with degree greater than two. At this point it is decided to include this customer in the chain if its demand does not exceed the capacity of the vehicle; otherwise, the chain ends and a new chain is started with it. In both cases, the next customer to enter the chain is the nearest, in the component, not yet in any chain. When this chain finishes, because of a customer with degree one or greater than two, a new one is started at the last customer with degree greater than two that is linked to customers that are not yet in any chain. Each chain generated in this way is a VRP route.

Version 4 – This version is equal to version 3, except that at a customer with degree greater than two, preference is given to enter the chain to the nearest customer with degree greater than one. The goal of this criterion is to have smaller and bigger chains. The computational experience shows that this goal is achieved.

In all versions, the number of initial routes may be significantly larger than K , but this is particularly true with versions 2, 3 and 4. In order to reduce the number of routes in these last 3 versions and improve the initial solution before starting the tabu search algorithm, a simple heuristic method was applied as follows.

The routes are sorted by decreasing order of their free capacity. Then the first route of the list is inserted into one of the other routes where it is feasible (the capacity is not exceeded) and the cost of insertion is lowest. If the insertion is not feasible in any route, stop. Otherwise, sort the routes again using the same criterion and repeat the process. The insertion cost is the minimum cost of inserting the whole route, in its original sequence or the reverse, between two customers (or the depot and one customer) of the destination route.

In the end, all the routes of the VRP solution produced by any version, with more than four customers, are reorganised using the *US* algorithm of Gendreau et al. [3], with the parameter that defines the neighbourhood, p , equal to 5.

Most of the VRP solutions generated by any of the four versions are infeasible in terms of number of routes, which is greater than K , and also in terms of capacity. This is no problem for the tabu search algorithm because it can deal easily with infeasible solutions. However, these initial solutions, even if

their cost is far from the optimum, contain very useful information for the tabu search algorithm, namely, the clusters of customers and crucial links of customers inside each cluster. This was confirmed comparing optimal VRP solutions and very good VRP solutions, with these initial solutions.

7.4 TABU SEARCH ALGORITHM

In this section, the characteristics of the tabu search algorithm for the VRP, simply called TSA, are described. In order to be applied, the TSA needs a starting solution, feasible or infeasible, which is one of those presented in the previous section.

In each iteration, the TSA performs two types of trial moves: insert and swap. An insert move consists in taking a customer from one route and inserting it into another route. A swap move consists in exchanging two customers belonging to two different routes. In order to reduce the computing time, the trial swap moves are only executed in the following situations: *i*) in those iterations in which none of the potential insert moves can be performed due to their tabu status; *ii*) every ten iterations. However, as will be reported later, we have also experimented two different values for condition *ii*). The set of candidates to be moved in each iteration is constituted by all the customers.

A crucial feature of the TSA is the strategic oscillation that is achieved by crossing the boundary of the feasibility. Therefore, a move can be performed even if the resulting solution is infeasible. Each trial move is evaluated according to the cost of the solution resulting from the move, which is given by the following equation:

$$\text{Cost} = \sum_{i=1}^r [c(i) + P \times e(i)], \quad (7.10)$$

where $c(i)$ is the cost (time) of route i , r is the total number of routes in the solution, P is a penalty, and $e(i)$ is the excess of capacity in route i . If a solution is feasible, $e(i)$ is equal to zero for all the routes. P , which initially is equal to 1, is multiplied by 2 if during the last 10 consecutive iterations all the solutions have been infeasible. Conversely, P is divided by 2 if all the solutions have been feasible during the last 10 consecutive iterations.

The move chosen in each iteration, is the trial move, not tabu, that generates the minimum cost solution as defined by equation (7.10). The tabu status of a move can be overruled if a solution is feasible and is better than any feasible solution known so far, or if it is infeasible and its cost is lower than the cost of any infeasible solution already known. In these two cases, not only the tabu status of the move is overruled but also the move is immediately performed without executing the remaining trial moves. This first best strategy has the objective of accelerating the search, but obviously it may impede the discovery of a better local optimum of the current neighbourhood.

The tabu status of a move is defined in the following way: a customer that leaves a route cannot return to it during a given number of iterations, θ . The tabu tenure of a move, θ , is a random number in the interval $\theta \in [n/6, n/3]$

(obviously, only the integer part of the division is considered). This way of defining the tabu tenure has the great advantage of being related to the size of problem. Naturally, it is worth experimenting to introduce also a factor that takes into account the number of routes in the solution.

In each iteration there are only two routes that are modified. These two routes are then reorganised using the *US* algorithm of Gendreau et al. [3], with p equal to 5.

In the TSA there are two independent ways of controlling the execution time: the maximum total number of iterations (N_1) and maximum number of iterations without improvement of the best known feasible or infeasible solution (N_2). When one (or both) of these number of iterations is attained, the execution of the programme is stopped. Therefore, the total number of iterations is not known in advance, depending on the evolution of the search. If during a given number of iterations (L) there is no improvement neither in the best known infeasible solution nor in the best known feasible one, the search restarts from the best known feasible solution (or best infeasible if none is feasible). With this restart, the tabu list is emptied. The aim of these restarts is to promote intensification.

7.5 COMPUTATIONAL EXPERIMENTS

The TSA depends on an important parameter that is chosen randomly in each iteration, i.e., the tabu tenure. Therefore, the solution and the computing time, may be different each time that the TSA is executed. In order to check the robustness of the TSA, each of the versions described below was executed 5 times with the same value for the parameters that are deterministic.

The subgradient method gives a different lower bound in each iteration, but we are more interested in applying the TSA algorithm to those lower bounds that are closer to the optimal VRP solution. At the same time it is important not to apply the TSA algorithm too many times in order to keep the computing time within reasonable limits. So, taking this all into account, it was decided to use the following lower bounds:

Method 1 - The lower bounds taken to produce the initial solution are those, obtained after iteration 300 of the subgradient method, that give an improvement over the best lower bound known so far. The reasons for this are twofold: i) we verified that in all test problems the lower bound after iteration 300 is close to the maximum reached; ii) the fact that the current lower bound is better than the previous ones, may indicate that the structure (connected component to which the customers belong and their relative position inside the component) of this lower bound is promising and different from the others.

The main disadvantages of this method are that the number of iterations, i.e., 300, specified may be inappropriate for other problems, and that the number of lower bounds considered is different, for each test problem. For the test problems of this research, this number varies between 49 for problem 3, and 20 for problems 5 and 7.

Method 2 - The lower bounds used are the best twenty given by the execution of the subgradient method during 1200 iterations. This method avoids the disadvantages of method 1, but the lower bounds may have structures more similar. However, this has not been confirmed by the computational experiments and, therefore, deserves a more objective analysis.

We should note here, that the lower bounds common to both methods are very few or, for some problems, just one.

The TSA algorithm was applied exactly in the same way with method 1 and method 2. The algorithm contains several phases whose only difference are the initial solutions, i.e., the initial solutions of a phase are half (the integer part) of the best solutions found in the previous phase, until only one solution (the best one) remains. For *phase 1*, the initial solutions are obtained, from the lower bounds defined by method 1 (or method 2), by one of the four versions described in Section 7.3. The solutions recorded to the next phase are the best 10 found in *phase 1*. In any phase the following parameters are used: $N_1 = 200\lceil\sqrt{n}\rceil$, $N_2 = 500$ and $L = [3.5n]$.

In each phase are explored a given number of elite solutions that give high diversity in the solution space. On the other hand, the value of the parameters N_1 , N_2 and L has a great influence on the intensity of the search in the neighbourhood of each solution. The set of initial solutions and parameters used had the aim of getting a compromise between the quality of the final solution and the computing time. In order to establish them some preliminary experiences were made but, obviously, there is no guarantee that any better combinations cannot be found.

In Tables 7.4 and 7.5 are presented the results for both methods of selecting the lower bounds and the four versions of constructing the initial solutions from these lower bounds. The best cost, the average cost, and average computation time have been obtained from the execution of the TSA algorithm five times on a Pentium II at 350 MHz.

From the observation of Tables 7.4 and 7.5 we can conclude that, on average, the results of both methods are almost identical, but, considering only the average of the best solutions, method 2 gives slightly better results than method 1. However, as should be expected, the computation time is substantially larger when method 1 is used, because the TSA algorithm has to explore more initial solutions in this case. This is especially true for problem 3, for which method 1 generates 49 initial solutions, while method 2 takes only 20.

Tables 7.4 and 7.5 also show that the behaviour of each of the four versions in both methods is almost the same. Version 1 produces, in general, worse solutions than the other three and takes a little bit more computing time. Both facts are related and they are a direct consequence of the procedure used in version 1, since all the customers that belong to the same connected component are kept together in the same route. This means that, unless all the customers have degree two in the connected component, the order in which they appear in the initial route is arbitrary. So, some of the information contained in the lower bound is lost. Therefore, the TSA algorithm starts with a worse solution,

Problem	VERSION 1			VERSION 2			VERSION 3			VERSION 4		
	Best cost	Average cost	Average CPU time (min)	Best Cost	Average cost	Average CPU time (min)	Best Cost	Average Cost	Average CPU time (min)	Best Cost	Average cost	Average CPU time (min)
1	524.61	524.61	9.2	524.61	524.61	7.8	524.61	524.61	7.6	524.61	524.61	7.6
2	839.00	841.92	9.2	835.67	836.60	9.4	835.28	836.23	9.2	835.77	837.64	9.2
3	826.14	827.01	21.3	826.14	827.09	18.8	826.14	828.75	18.9	826.14	826.63	19.7
4	1040.72	1045.18	34.8	1038.62	1041.29	25.9	1038.14	1044.02	26.5	1037.53	1042.43	27.9
5	1369.62	1381.84	32.3	1347.00	1357.21	27.8	1337.17	1345.18	27.9	1350.48	1360.49	27.6
6	819.56	819.56	12.8	819.56	819.56	11.9	819.56	819.56	11.7	819.56	819.56	11.4
7	241.97	241.97	11.8	241.97	241.97	11.5	241.97	241.97	11.5	241.97	241.97	10.5
8	1162.96	1165.43	60.9	1165.97	1166.36	58.4	1162.96	1163.99	57.8	1162.96	1166.10	57.5
Mean	853.07	855.94	24.04	849.94	851.84	21.44	848.23	850.54	21.39	849.87	852.43	21.43

Table 7.4 VRP solutions obtained by the TSA algorithm with lower bounds chosen by method 1.

Problem	VERSION 1			VERSION 2			VERSION 3			VERSION 4		
	Best cost	Average cost	Average CPU time (min)	Best cost	Average cost	Average CPU time (min)	Best cost	Average cost	Average CPU time (min)	Best cost	Average cost	Average CPU time (min)
1	524.61	524.61	4.6	524.61	524.61	4.3	524.61	524.61	4.1	524.61	524.61	4.2
2	835.43	839.78	7.2	835.45	840.62	7.6	837.86	840.83	7.1	835.67	839.16	7.6
3	828.74	830.22	10.3	826.14	828.19	10.1	826.14	826.72	9.0	826.14	826.90	9.8
4	1043.44	1050.89	18.6	1037.73	1042.56	14.7	1041.49	1044.09	15.6	1037.95	1041.40	15.1
5	1366.27	1380.03	28.4	1344.03	1353.62	25.5	1326.61	1343.46	24.5	1332.14	1343.30	24.6
6	819.56	819.56	7.0	819.56	819.56	7.0	819.56	819.96	7.1	819.56	819.56	6.8
7	241.97	241.97	9.7	241.97	241.97	10.1	241.97	241.97	10.3	241.97	241.97	10.0
8	1166.45	1170.09	37.7	1162.96	1170.31	35.5	1162.96	1163.99	38.4	1162.96	1164.72	37.5
Mean	853.31	858.39	15.44	849.06	852.68	14.35	847.65	850.64	14.51	847.62	850.20	14.45

Table 7.5 VRP solutions obtained by the TSA algorithm with lower bounds chosen by method 2.

in terms of underlying structure, and performs more iterations but with less benefit to the solution cost.

Furthermore, Tables 7.4 and 7.5 allow to conclude that the TSA algorithm is rather robust because the percentage difference between the best solution for each test problem and the corresponding average solution is, in general, quite small. The largest average difference, for each version, considering both methods, is less than 0.6%, but the minimum is 0.22% for version 2.

The average performance of versions 2, 3 and 4 is almost identical and none of them outstrips the others for every test problems. Nevertheless, it seems that overall version 3 is a little better than version 4, and this one gives slightly better results than version 2. However, considering only method 2, version 4 is, on average, slightly better than versions 2 or 3.

Tables 7.4 and 7.5 report the results when the trial swap moves inside the TSA algorithm have been executed every ten iterations. Besides this, we have also tried two different options as described in the following:

- i)* To apply the trial swap moves in every iteration. This was only used for method 2 and problems 2, 3 and 5, because these are those for which it is the more difficult to match the best published results.

With this option the best solutions obtained are better (than those of Tables 7.4 or 7.5), for any of the problems and versions, and the same happens with the average solutions. Besides, the gap between the best solution for each problem and the corresponding average solution, is shrunk even further. For example, for problem 2 the value of 835.26 was obtained for all runs and any of the versions, so the best and the average value is the same. The best solutions that have been obtained for problems 2, 4 and 5 are, respectively, 835.26, 1031.32 and 1310.08. Since this last solution is, so far as we know, better than any published in the literature, it is given in the Appendix I.

The largest gain, among those three problems, is always obtained for problem 5 in any of the versions. This can be explained by the fact that this is the problem with the largest number of customers, routes and routing cost, and whose solution is certainly far from the optimum.

The disadvantage of this option is that the computing time increases, on average, about 3 times for problem 2, about 3.7 times for problem 4, and about 4 times for problem 5, i.e., the execution of this problem takes about 100 minutes.

- ii)* The trial swap moves have not been applied except in those iterations in which none of the trial insert moves can be performed. The experience included both methods, all versions and all the test problems, so the results allowed to construct two tables equal to Tables 7.4 and 7.5.

The influence of the swap moves on the quality of the solutions and on the computing time is the same as already identified in the option *i*. On average, the solutions are worse than those of Tables 7.4 and 7.5, either in relation to the best cost or to the average cost. The overall difference is about 0.7% for

Problem	Cost	Observation	Best published cost	Difference (%)
1	524.61	all versions of both methods	524.61	0.00
2	835.26	all versions, method 2	835.26	0.00
3	826.14	all versions, but vs.1 of mth.2	826.14	0.00
4	1031.32	version 4, method 2	1028.42	0.28
5	1310.08	version 2, method 2	1334.55	-1.83
6	819.56	all versions of both methods	819.56	0.00
7	241.97	all versions of both methods	241.97	0.00
8	1162.96	all versions, but vs.2, mth.1 and vs.1, mth.2	1162.96	0.00
Mean	843.99		846.68	-0.19

Table 7.6 Comparison of the best VRP solutions obtained by the TSA algorithm with the best solutions published.

version 1 and about 0.3% for the other three versions. However, with this option a few solutions slightly better than the corresponding best solutions of Tables 7.4 and 7.5 have been discovered, as well as equal value solutions for problems 1, 6 and 7.

With this option the computing time decreases, relatively to that presented in Tables 7.4 and 7.5, about 32% for method 2 and 23% for method 1, on average.

It is very interesting to notice that both options confirm all the conclusions already drawn from Tables 7.4 and 7.5, namely, the relative behaviour of all versions in terms of solutions quality and computing time.

In Table 7.6 are presented the best solutions discovered in this research (they are taken from Tables 7.4 and 7.5, and for three problems are given by option i as explained above), and the best published solutions that have been taken from different sources: from Taillard [8] for problems 1, 2, 3, 4, from Osman [6] for problem 5, and from Fisher [5] for problems 6 and 7, where they are proven to be optimal solutions, and from Rochat and Taillard [7] for problem 8 (for problem 5 there are solutions with lower costs, but with 17 routes).

Table 7.6 shows that the performance of the TSA algorithm is very good because it is able to find the best published solutions, obtained by several different algorithms, for all the test problems, except for one of them, and giving also a new best solution. Besides, the computing time is rather short.

7.5.1 Influence of the initial solution

One of the features that most distinguishes this TSA algorithm is the use of an initial solution given by a VRP lower bound. We are convinced, since the beginning of the computational experiments, that the performance of the TSA algorithm is, to a great extent, due to the role of the initial solutions. In order to prove this, the TSA algorithm was applied with an initial solution produced by the classical insertion heuristic, as described in the following.

The insertion heuristic, as used here, constructs the routes sequentially. Each route is started with the unrouted customer farthest from the depot; then the

customer with least insertion cost is chosen, between the customers (or depot) of this route, among the unrouted customers whose demand does not exceed the available capacity in the vehicle. This is repeated until no customer is admissible in the route under construction. In this case, a new route is started, or the process stops if all the customers already belong to a route.

The information contained in Table 7.7 is the following:

A – Solutions produced by the insertion heuristic, which are used by the TSA algorithm as initial solutions. The number of routes is not present because it happens to be the same as defined in Table 7.1 for each problem. The computing time is less than a second for all the eight problems.

B – The values in this column result from the execution of the TSA algorithm, five times, with exactly the same parameter values defined previously, for the construction of Tables 7.4 and 7.5.

C – The values given in this column are obtained in the same way as those of column *B*, with the only difference that the maximum number of iterations allowed, $N1$ and $N2$, has been increased a lot, in order to guarantee that the execution time of each problem was equal to the computing time required by version 4, method 2, presented in Table 7.5 (the execution of each run is stopped when this happens).

Column *B* of Table 7.7 shows that the TSA algorithm is capable of improving the initial solution substantially, in a very short time. However, for the most difficult problems (number 4, 5 and 8, which are the problems with the largest number of customers), the final solution is still far away from the best known solution.

Comparing columns *B* and *C*, we conclude that by executing many more iterations of the TSA algorithm, the solutions are improved. As should be expected, this improvement is small for the easier problems, because the solutions are already good, and quite large, both for the best and for the average, for the other problems. However, the solution values for problems 4, 5 and 8, are still considerably worse than those presented in Table 7.5 for version 4. Besides, it should be noted that the computing time reported in Table 7.5, for version 4, is that because the TSA algorithm is always executed with the same parameters, independently of the test problems. However, the optimal solution for the easiest problems (number 1, 6 and 7) is always found in a few seconds.

This experience shows unequivocally that the initial solutions generated from the lower bounds play an important role in the final performance of TSA algorithm. This results from the conjunction of two factors: the diversity created by the exploration of several different initial solutions and their intrinsic quality. In fact, the K -trees of two consecutive iterations of the subgradient method are, in general, quite different. For example, considering the best twenty lower bounds that have been used for problem 5, the percentage of common arcs among all the corresponding K -trees, varies from a minimum of 64% to a maximum of 86%. For all the other test problems, these values are very similar. On the other hand, the intrinsic quality of the initial solutions cannot be proven,

Problem	A		B			C		
	Cost	Best cost	Average		CPU time (min)	Best cost	Average cost	CPU time (min)
			Average cost	CPU time (min)				
1	610.52	527.67	530.35	0.3	524.61	524.61	4.2	
2	1090.95	842.33	850.22	0.4	836.62	842.54	7.6	
3	1196.03	829.72	834.55	0.5	827.39	828.57	9.8	
4	1342.78	1057.47	1061.26	0.8	1047.34	1050.70	15.2	
5	1788.30	1471.73	1502.23	1.1	1394.16	1408.44	24.6	
6	1056.2	819.56	819.56	0.4	819.56	819.56	6.8	
7	315.00	241.97	242.49	0.5	241.97	241.97	10.0	
8	1745.51	1227.5	1264.64	1.9	1167.57	1215.67	37.5	
Mean	1143.16	877.24	888.16	0.74	857.40	866.51	14.46	

Table 7.7 VRP solutions obtained by the TSA algorithm with the initial solutions generated by an insertion method.

but it seems reasonable to admit that if a lower bound is close to the optimal solution they should have similar structures. Besides, these initial solutions are submitted to the TSA algorithm, during the first phase, which allows the selection of the best half. However, it should be clear that the initial solution given by the insertion procedure is not comparable with the initial solutions generated from the lower bounds, because, in general, the number of routes is substantially different and many of these are infeasible, in relation to the vehicle capacity.

In conclusion, the initial solutions given by the lower bounds are a way of diversifying with a quality that is difficult to achieve by other means.

7.6 CONCLUSIONS

The main idea of this research was to use a good lower bound as a starting solution for a tabu search algorithm for the VRP. We tried two methods that had proven to give quite good lower bounds — the k -degree centre tree, developed by Christofides et al. [2], and the K -tree, developed by Fisher [5] — but, in the end, it was decided to use only the last one because, in general, its lower bounds are better. We also tried to take advantage of good VRP infeasible solutions to improve the lower bounds, but this was not successful.

The computational tests have shown that the idea of using the lower bound as an initial solution is fruitful, because very good VRP solutions have been obtained in a reasonable amount of computation time. The lower bound contains very useful information about the optimal solution, from which the tabu search algorithm benefits. In this article, the lower bounds are between 86% and 97% of the best known solution. Obviously, if these lower bounds can be improved, the resulting information will be much more accurate and the benefit for the tabu search algorithm will be much higher. Another reason for the good performance of the tabu search algorithm is the diversity of the solution space explored, which is caused by the use of several different good lower bounds for the same problem.

The approach followed here can also be applied to vehicle routing problems with other types of constraints, because Fisher [5] shows how his lower bound can be extended to such problems. However, for these cases the computational implementation is much more difficult and there is no indication of its performance. The results of this research show that our method finds always the optimal solutions of problems with a medium number of customers (50 to 100). This suggests that to devise a method of dividing the larger problems into smaller problems and then applying this approach to each of the subproblems is worth trying. This kind of reasoning is the basis of many techniques, like branch and bound, and, what is called in artificial intelligence, divide and conquer. A technique of partitioning has been applied successfully to the VRP by Taillard [8].

Acknowledgments: This research was supported by Fundação para a Ciência e Tecnologia under the program PRAXIS XXI, project n° 3/3.1/CEG/2661/95.

References

- [1] N. Christofides, A. Mingozi, and P. Toth. The Vehicle Routing Problem. In: *Combinatorial Optimization*, N. Christofides, A. Mingozi, P. Toth, and C. Sandi, editors, pages 315–338, Wiley, 1979.
- [2] N. Christofides, A. Mingozi, and P. Toth. Exact Algorithms for the Vehicle Routing Problem, Based on Spanning Tree and Shortest Path Relations. *Mathematical Programming*, 20:255–282, 1981.
- [3] M. Gendreau, A. Hertz, and G. Laporte. New Insertion and Post-Optimization Procedures for the Traveling Salesman Problem. *Operations Research*, 40:1086–1094, 1992.
- [4] M.L. Fisher. A Polynomial Algorithm for the Degree-Constrained Minimum K -Tree Problem. *Operations Research*, 42:775–779, 1994.
- [5] M.L. Fisher. Optimal Solution of Vehicle Routing Problems using Minimum K -Trees. *Operations Research*, 42:626–642, 1994.
- [6] I.H. Osman. Metastrategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem. *Annals of Operations Research*, 41:421–451, 1993.
- [7] Y. Rochat and E.D. Taillard. Probabilistic Diversification and Intensification in Local Search for Vehicle Routing. *Journal of Heuristics*, 1:147–167, 1995.
- [8] E. Taillard. Parallel Iterative Search Methods for the Vehicle Routing Problems. *Networks*, 23:661–673, 1993.

APPENDIX I - New best solution for problem 5, with sixteen routes:

TOTAL TIME = 1310.081
ROUTE 1: 0.53_198_197_186_39_187_139_4_155_110_179_149_0; Time = 77.253
ROUTE 2: 0.40_75_56_23_67_170_25_55_165_130_54_195_26_0; Time = 101.871
ROUTE 3: 0.50_185_79_169_121_29_24_163_134_68_80_150_177_109_0; Time = 86.901
ROUTE 4: 0.147_96_104_99_93_85_59_94_183_0; Time = 48.196
ROUTE 5: 0.58_152_2_178_57_15_43_142_42_172_144_87_117_13_0; Time = 79.755
ROUTE 6: 0.62_11_64_49_143_36_46_124_47_168_48_82_153_0; Time = 127.191
ROUTE 7: 0_129_78_34_164_135_35_136_65_71_66_188_20_128_122_0; Time = 125.535
ROUTE 8: 0_102_157_33_81_120_9_161_103_51_1_0; Time = 77.091
ROUTE 9: 0.52_106_194_7_182_123_19_107_175_159_148_88_146_0; Time = 77.245
ROUTE 10: 0_156_27_167_127_190_31_162_132_176_111_0; Time = 49.542
ROUTE 11: 0.95_97_37_100_192_14_38_140_86_113_17_84_60_166_0; Time = 102.385
ROUTE 12: 0_112_118_83_199_125_45_174_8_114_18_0; Time = 66.664
ROUTE 13: 0_6_92_151_98_193_91_191_119_44_141_16_61_173_5_89_0; Time = 77.769
ROUTE 14: 0_69_101_70_30_160_131_32_181_63_126_90_108_10_189_0; Time = 92.387
ROUTE 15: 0_105_180_21_73_72_74_171_133_22_41_145_115_137_0; Time = 70.235
ROUTE 16: 0_28_76_196_158_3_77_116_184_12_154_138_0; Time = 50.059

8 A SIMULATED ANNEALING APPROACH FOR MINIMUM COST ISOLATED FAILURE IMMUNE NETWORKS

Alfredo Candia¹ and Hugo Bravo²

¹Dpto. de Ingeniería de Sistemas-Universidad de Talca
Km. 1 Camino Los Niches, Curicó, Chile
acandia@pehuenche.UTALCA.CL

²Dpto. de Matemática-Universidad de Tarapacá
Avda. General Velásquez 1775, Arica, Chile
hbravo@vitor.faci.UTA.CL

Abstract: Isolated failure immune networks are communication networks having an interesting immunity structure. An application of Simulated Annealing algorithm to the minimum cost failure immune networks problem is presented. Computational experiments show that the metaheuristic approach outperforms previous simple heuristics. The dependency on starting solution and neighborhood scheme is also analyzed.

8.1 INTRODUCTION

Due to the great advance in the world of communications, the study of problems involving networks increases continuously. One of the basic problems is to design networks having connectivity between the nodes at a low cost and possessing some degree of immunity to failures.

One of the typical topologies used are *trees*, which have the property of preserving connectivity while there are no failures in any line or any node different to leaves (a *leaf* is a node of degree 1). Another well known topology is *two-connected* networks, see Monma and Shallcross [13] for which there are two disjoint paths between any pair of nodes. However, a small number of failures

could disconnect the network. Farley [7] considered *Isolated Failure Immune networks* (IFI networks), which have an interesting immunity structure. IFI networks work with three types of failures:

- i. (x, y) and (p, q) are two *line isolated failures* if (x, y) and (p, q) are not incident to a common node.
- ii. x and y are two *node isolated failures* if x and y are not connected.
- iii. A *line failure* (x, y) and a *node failure* p are *isolated* if (x, y) is not incident to p or to a node connected to p .

Figure 8.1 shows three examples of non isolated network failures.

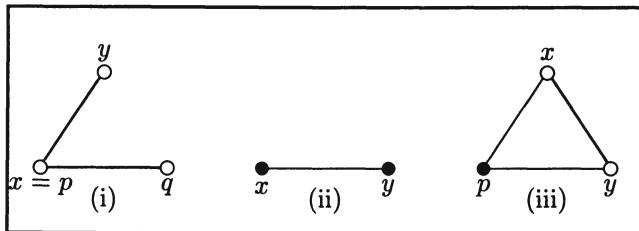


Figure 8.1 Pairs of non isolated network failures.

A set of failures is *isolated* if the failures in the set are pairwise isolated. The network is IFI if it remains connected as long as network failures are isolated.

A *2-tree* is a member of a class of undirected graphs defined recursively as follows:

- An edge is a 2-tree.
- If T is a 2-tree with $(n - 1)$ nodes, then a new 2-tree with n nodes is formed by creating a new node v and adding edges between v and the two terminal nodes of an edge in T .

Figure 8.2 shows the construction of a 2-tree beginning with the edge $(1, 2)$.

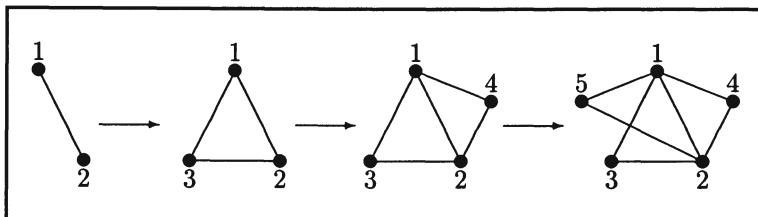


Figure 8.2 A construction of a 2-tree with 5 nodes.

Farley [7] proved that 2-trees are minimal IFI networks (minimal respect to line inclusion) and later on, Wald and Colbourn [15] proved that every minimal IFI network is a 2-tree.

It is assumed that every complete network $G = (V, E)$ has a non-negative cost function associated on the edges, so finding a minimum cost IFI network in G is equivalent to finding in G a *minimum cost spanning 2-tree*. This optimization problem in graphs, was first studied by Bern [4], followed by Cai and Maffray [5] in the more general context of the *minimum cost spanning k-tree problem*. After that, Beltrán and Skorin-Kapov [3] and Beck and Candia [2], studied the special case when $k = 2$.

In Section 8.2 we present some terminology and properties about 2-trees. In Section 8.3 Bern's results on the complexity and an exact algorithm for MS2T are presented. Section 8.4 describes simple heuristics (Greedy and Stars) for MS2T. In Section 8.5 the proposed Simulated Annealing algorithm is described and Section 8.6 discusses computational results. Finally, Section 8.7 offers some conclusions and suggestions for future research.

8.2 2-TREES AND SOME PROPERTIES

It is easy to note that a 2-tree with n nodes has exactly $(2n - 3)$ edges. Each 2-tree may be constructed using different sequences of the operations defined in the two steps in the above definition of a 2-tree.

A simple cycle $[v_0, v_1, \dots, v_l, v_0]$ is *chordless* if $(v_i, v_j) \notin E$ for i and j differing by more than $1 \bmod(l + 1)$. A *clique* of size p is a complete graph with p nodes. T denotes a 2-tree with n nodes. We present a summary of well known properties of 2-trees:

- T contains no chordless cycle of length of at least 4;
- T contains no clique of size 4;
- T has exactly $(n - 2)$ cliques of size 3;
- either T is a line or every maximal clique of T is of size 3;
- if $n > 2$ every node of T has a degree of at least 2; and
- T has no cutset of size less than 2.

2-trees (and more general, k -trees, see [5]) have been extensively studied lately. Some \mathcal{NP} -Hard problems are tractable when they are restricted to k -trees. For instance, the Steiner problem in graphs, a classical \mathcal{NP} -Hard problem, accepts a linear time algorithm when restricted to 2-trees (Wald and Colbourn [15]). Another optimization problem on k -trees is discussed in the work of Granot and Skorin-Kapov [9]; in particular, they design polynomial algorithms for several k -cable distance optimization problems. Finally, Fernández-Baca and Medepalli [8] considered the problem of finding a minimum-cost assignment of program modules for processors in a distributed system where one

of the processors has a limited memory. This problem is \mathcal{NP} -Hard, even if the communication graph is a tree; they showed that a fully polynomial-time approximation scheme exists for the case when the communication graph is a partial k -tree, that is, a graph which is embedded in a k -tree with the same node set.

8.3 MS2T: COMPLEXITY AND EXACT ALGORITHMS

The Minimum Spanning k -tree problem can be formally defined in the following way.

Let $G = (N, E)$ be an undirected graph with non-negative edge cost $c_{ij} \geq 0$, for each $(i, j) \in E$. The *Minimum Spanning k -Tree problem* (**MSkT**) is to find in G a k -tree T with node set N and edge set $E' \subseteq E$ of minimum total cost.

We note that the MS1T corresponds to the classical *minimum spanning tree problem*, for which there are some polynomial time algorithms, such as Prim-Dijkstra's algorithm ([14], [6]) and Kruskal algorithm ([11]), for instance. However, in his doctoral thesis, Bern [4] proved the following result.

Theorem 1 *MSkT is \mathcal{NP} -hard for $k \geq 2$, even if the cost function c satisfies the triangle inequality.*

Cai and Maffray [5] additionally proved that MS2T remains \mathcal{NP} -hard for maximal planar graphs.

We will need the following definitions. An *addition sequence* for a 2-tree T with $n \geq 2$ nodes is a sequence $(v_1, K_1), (v_2, K_2), \dots, (v_{n-2}, K_{n-2})$ where each v_i is a newly created node and each K_i is an already existing edge to which v_i is to be attached. The first edge K_1 will be called the *starting edge*, and v_1 is the *first new node*. We say that v_i is added to the edge K_i . (ϕ, K_1) represents an addition sequence for the 2-tree consisting of just an edge K_1 .

The following Lemma was also proved by Bern [4].

Lemma 1 *The number of legal addition sequences for k -trees with n nodes, where $n > 2$, is $\binom{n}{2} \prod_{i=1}^{n-2} i(2i - 1)$.*

The Lemma shows that a brute force algorithm, based on addition sequences would have running time no better than $\Omega(\binom{n}{2} \prod_{i=1}^{n-2} i(2i - 1))$, which is $\Omega(n!)$ a very slow algorithm.

Bern [4] also designed an exponential time dynamic programming algorithm for the exact solution of MSkT. For completeness, we present here the recurrence relations for $k = 2$, in a corrected form. The original equations given by Bern [4] contain a minor mistake. It is assumed that $|N| \geq 3$; otherwise, the problem is trivial. Let $U \subseteq N$, $|U| \geq 3$. The cost of a minimum 2-tree $T(U)$ with node set U will be denoted by $t(U)$. By $t_C(U)$ we denote the length of a minimum 2-tree $T_C(U)$ with node set U subject to the constraint that $C \subset U$ induces a 3-clique in $T_C(U)$. By $t_K(U)$ we denote the cost of a minimum k -tree $T_K(U)$ with node set U subject to the constraint that $K \subset U$ induces a 2-clique (an edge) in $T_K(U)$.

c_{uv} denotes the cost of the edge (u, v) in G , assumed infinite if there is no edge between u and v . The cost $c(u, C)$ of a node u to a set C to be $\sum_{v \in C} c_{uv}$ is also defined.

The algorithm proceeds in order of subset size, first computing $t_C(U)$ and $t_K(U)$ for each subset U of cardinality p , each $C \subset U$ such that $|C| = 3$, and each $K \subset U$ such that $|K| = 2$, before moving on to U of cardinality $p + 1$, where p runs from 3 to n . It is possible to show that there are two possibilities for computing $t_C(U)$: either $T_C(U)$ is the union of two strictly smaller 2-trees, each optimal under the constraint that C induces a clique, or C contains a node of degree 2 (a leaf). Assuming the latter case, we compute $t_C(U)$ by considering each possibility for the degree 2 node v and picking the best choice. For each choice, $T_C(U)$ is the union of a 2-tree on $U - \{v\}$ and all edges between v and $C - \{v\}$.

We have the following corrected (the term $c(C)$ was skipped in Bern's thesis) recurrence equations for $t_C(U)$, for $|U| > 3$:

$$t_C(U) = \min\{c_1, c_2\},$$

where:

$$c_1 = \min_{\emptyset \subseteq U' \subseteq U - C} \{t_C(U - U') + t_C(U' \cup C) - c(C)\}$$

and

$$c_2 = \min_{v \in C} \{c(v, C - \{v\}) + t_{C - \{v\}}(U - v)\}.$$

Since each edge in a 2-tree with more than 2 nodes is contained within a 3-clique, computing $t_K(U)$ is easy once $t_C(U)$ has been computed for each C :

$$t_K(U) = \min_{v \in U - K} \{t_{K \cup \{v\}}(U)\}.$$

At the end of the computation $t(N) = \min_C \{t_C(N)\}$. The only initial conditions are that $t_C(U)$ and $t_K(U)$ are each equal to the cost of the 3-clique on U if $|U| = 3$. We have the following theorem:

Theorem 2 (Bern [4]) *A minimum spanning 2-tree in an arbitrary n -node graph can be computed in time $O(n^3 \cdot 3^n)$.*

In [5] it is showed that MS2T remains \mathcal{NP} -Hard for maximal planar graphs.

8.4 ON SIMPLE HEURISTICS FOR MS2T

We review two simple heuristics for MS2T. This problem can be formally defined in the following way.

Let $G = (N, E)$ be an undirected complete graph with non-negative edge cost $c_{ij} \geq 0$, for each $(i, j) \in E$. The *Minimum Spanning 2-Tree problem* (MS2T) is to find in G a 2-tree T with node set N and edge set $E' \subseteq E$ of minimum total cost.

In [2], a greedy heuristic for MS2T is proposed. The heuristic accomplishes a greedy strategy, based on the recursive definition given in Section 8.2, for finding feasible solutions.

Greedy-Prim

Input: A complete graph $G = (N, E)$ with non-negative edge costs.

Output: A spanning 2-tree of G .

1. Choose an edge in G . Let T be that edge.
2. Compute the smallest cost of connecting a node not in T to some edge in T . Add the new node and the corresponding two edges to T .
3. If T includes all nodes in N then the algorithm finishes with solution T . Otherwise, go to step 2.

Figure 8.3 shows an input network for MS2T and Figure 8.4 shows the application of Greedy-Prim to the instance shown in Figure 8.3.

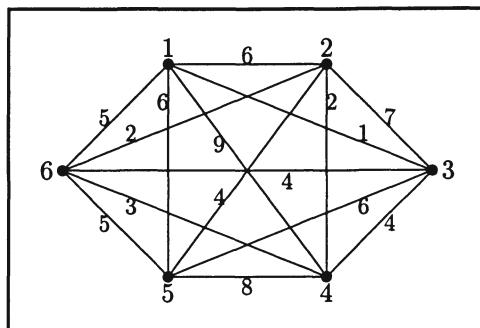


Figure 8.3 An input network for MS2T.

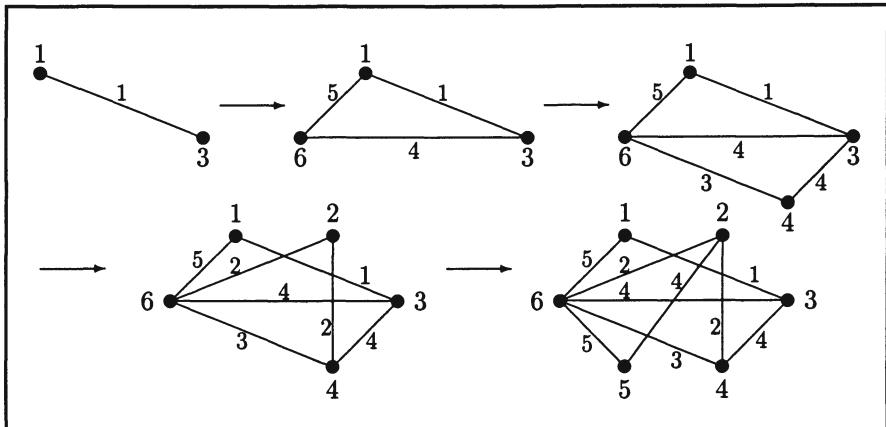


Figure 8.4 Application of Greedy-Prim to the instance at Figure 8.3.

Clearly, this algorithm is a generalization of Prim's algorithm for the minimum spanning tree problem [14] to the MS2T. There are different alternatives for implementing step 1; for instance, the starting edge is chosen randomly or is an edge of minimum cost. They also formulated a repetitive version of Greedy algorithm which eventually obtains a set of feasible solutions choosing a different starting edge in step 1. Clearly, repetitive Greedy dominates Greedy-Prim.

Stars Heuristic (SH) is one of the heuristics described in [3]. It is based on an embedding of a minimum cost spanning tree into a 2-tree. If $G = (N, E)$ is a weighted complete network, SH first finds a minimum spanning tree $T = (N, E_T)$ in G . For $i \in N$, let $N_T(i)$ represent the set of nodes adjacent to i in T . For each node i of degree greater than one in T , the corresponding star S_i is a subgraph of T induced by $\{i\}$ and all of its neighbors $N_T(i)$. For each star S_i we find a minimum spanning tree T_i on the subgraph of G induced by the node set $N_t(i)$. Finally, all such trees T_i , together with the minimum spanning tree T , form a 2-tree (the feasible solution given by SH). In [3] the correctness of this heuristic is proved.

Figure 8.5 shows the application of SH to the instance of MS2T shown in Figure 8.3 and also shows the feasible solution obtained by SH.

In [2] the empirical performance of Greedy-Prim and SH is studied. Their analysis showed that repetitive Greedy outperform SH. Due to the second phase of SH, that is, the process of embedding of a minimum cost spanning tree into a 2-tree by local optimization criteria, the performance of SH is really poor. In Section 8.6 we also show a comparison among three simple heuristics for MS2T. A tabu search based heuristic is also presented in [3], and computational experiments were conducted for very small networks.

8.5 SIMULATED ANNEALING APPROACH FOR MS2T

Simulated Annealing (SA) is an important technique for solving complex combinatorial problems, see Laporte and Osman [12] and Aarts and Lenstra [1] for metaheuristics and applications.

SA can be considered as a generalization of local search, where a probabilistic criteria for solution acceptance is used. This probabilistic criteria consists on accepting with probability 1 any solution that improves the current solution, and accepting with probability p ($0 < p < 1$) any other solution. Normally p depends on the objective function of the particular problem. Using these criteria it is possible to escape from local optima and then it is possible to reach global optima.

A short description of our SA algorithm applied to MS2T is given now.

1. Get a starting spanning 2-tree T .
2. Get an initial temperature $t = t_0$. We use $t_0 = 2 \cdot (c_{\max} - c_{\min})$.
3. While the stopping rule is not satisfied

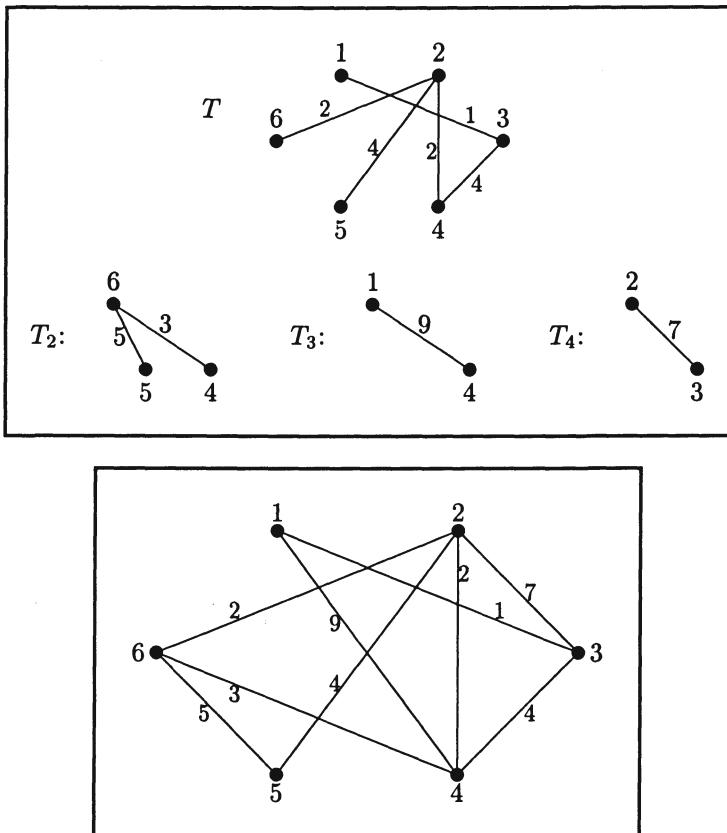


Figure 8.5 The application of SH and the spanning 2-tree given by SH.

3.1 Perform the following loop $L = 2n - 3$ times

3.1.1 Generate a random neighbor T' of T

3.1.2 Evaluate $\Delta = \text{cost}(T') - \text{cost}(T)$

3.1.3 If $\Delta \leq 0$ set $T = T'$ (accepted move)

3.1.4 If $\Delta > 0$ generate a random number $x \in (0, 1)$

If $x < e^{-\Delta/t}$ set $T = T'$ (T' is accepted) otherwise set $T = T$ (T' is rejected)

3.2 set $t = \alpha \cdot t$, α is a parameter, $0 < \alpha < 1$

4. Return T , the best spanning 2-tree found

An important component of any SA implementation is the *neighborhood scheme*. In [3], two classes of neighbors are alternately used for a given solution in the proposed tabu search algorithm, the interchange neighborhood and

the edge based neighborhood. The interchange neighborhood is based on one interchange in the aggregation sequence followed by a partial greedy scheme.

More specifically, let $S = [1, 2, \dots, i, \dots, j, \dots, n]$ be a sequence of nodes that determines the construction of a 2-tree T . The move from T to another 2-tree T' based on a single exchange of two nodes in the above sequence S is defined.

Move R_{ij} : Exchange i and j in S . Construct 2-tree T' using the new sequence in such a way that in each step of the construction the smallest increase in cost is incurred.

The neighbors of T consist of all 2-trees different from T , that can be reached from T by a single move R_{ij} . The aggregation sequence for a given 2-tree T is not unique, and the 2-tree T' obtained by move R_{ij} is not necessarily different from T . However, the exchange move R_{ij} may lead to a 2-tree T' , which is considerably different from T .

The edge based neighbors of T differ from T in 2 edges at the most sharing most of the structure with T . The basic idea is to remove an edge e from T (this may force the elimination of another edge from T) and then complete a new 2-tree T' by adding one (or two) appropriate edges. The details of the procedure are explained now. We denote $d(x) = \text{degree of node } x$.

Given a spanning 2-tree T , a neighbor T' is also a spanning 2-tree obtained from T as follows:

1. Let (x, y) be any edge of T .

2. If $d(x) = 2$ then

2.1 remove (x, y) from T

2.2 remove (z, x) from T , where z is the only neighbor of x ($z \neq y$)

2.3 an edge (u, v) is arbitrarily chosen from T and add the edges (u, x) and (v, x) preserving the 2-tree structure. (see Figure 8.6)
If $d(y) = 2$ we interchange the labels $x \leftrightarrow y$.

3. If $d(x) > 2$ and $d(y) > 2$ then a node list v_1, v_2, \dots, v_k is generated, where v_i is adjacent to both x and y , for $i = 1, \dots, k$. There are four cases:

3.1 the list is empty. This case is impossible if T is a 2-tree.

3.2 the list contains only one node v .

If $d(v) = 2$ then there is no 2-tree (if $n > 3$).

If $d(v) > 2$ we choose z ($z \neq x, z \neq y$), neighbor of v and adjacent to x or y . If z is adjacent to y (respect. x), remove (x, y) and add (x, z) (resp. (y, z)). (see Figure 8.7)

3.3 the list contains exactly two nodes v_1 and v_2 . In this case, remove (x, y) and add (v_1, v_2) . (see Figure 8.8)

3.4 the list contains 3 nodes or more ($k \geq 3$). In this case, the elimination of (x, y) and insertion of a new edge does not generate a new 2-tree.

Another edge must be tested for elimination.

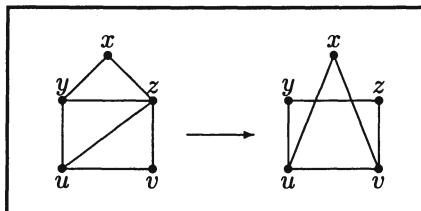


Figure 8.6 Neighbor according with 2.3.

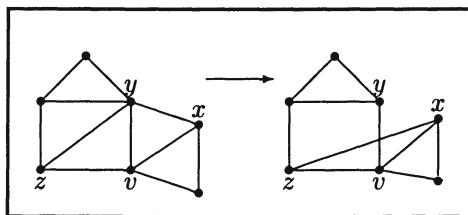


Figure 8.7 Neighbor according with 3.2.

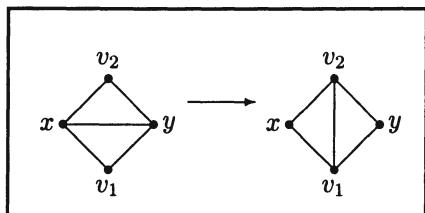


Figure 8.8 Neighbor according with 3.3.

So, the tabu search described in [3] alternates between edge based and recursion based neighbors. The search starts with a feasible solution T . At each iteration they evaluate all possible edge based neighbors of T and choose the one that leads to the lowest cost 2-tree (greedy type move). After carrying out this process for n iterations (n : number of nodes in T) they switch to the recursion based neighbors. At each iteration the best simple exchange in the recursion construction sequence is performed (another greedy type move).

After n iterations they switch back to the edge based moves and repeat the process for n alterations.

Separate tabu lists for both neighborhood schemes were used in [3]. For the edge based search, an edge leaving the solution became a tabu attribute. For the recursion based search, where two nodes i and j in the construction sequence are to be exchanged, a single edge (i, k) from the current solution becomes a tabu attribute. The tabu status of a tabu edge becomes inactive only if its inclusion in the current solution leads to a solution better than the best one found so far.

Our SA algorithm differs from tabu search [3] mainly in the way a neighbor is chosen; for tabu search the best neighbor is always considered at each iteration, instead of SA where a neighbor is selected with a probabilistic acceptance criteria.

8.6 EXPERIMENTAL RESULTS

As commented in Section 8.4, the tabu search based heuristic of Beltrán and Skorin-Kapov [3] was only applied to small instances, that is, graphs with no more than 25 nodes. Their computational experiments were performed on Sun Sparc 2 workstation and no computational time was reported. We suppose that they did not carry out bigger instances due to high computational times. Also the reported average error increases with size problems. For that reasons we try another metaheuristic for solving MS2T, namely Simulated Annealing.

Our algorithms were coded in C language and executed on a Pentium 200 MHz. We consider 40 instances in four groups of 10 complete graphs with 10, 25, 50 and 100 nodes. The edge-costs are in the interval [1,100].

First of all, in Table 8.1 we present a comparison among Greedy-Prim, Greedy-K and Stars. Greedy-K is another heuristic presented in [3] using a different greedy scheme (similar to Kruskal algorithm for Minimum Spanning Trees). Greedy-K starts including the cheapest edge in the solution and, in contrast to Greedy-Prim, it is required that all connected components of the current solution to be 2-trees. This occasionally results in the addition of more than one edge in a single iteration.

We only present here the results for the last group of 100 nodes. For the other groups the results are similar.

From Table 8.1 it is clear that Greedy-Prim outperform clearly Greedy-K and Stars confirming the results obtained in [2].

For SA experiments, we first implemented ten SA versions, fixing the following parameters: repetitions=5, factor=1.0, minpercent=0.03, cutoff=0.5, and varying freeze-lim, size-factor, factor-temp and t-lim. From these experiments and analizing the trade-off between time and solution quality, we fixed the latter parameters as: freeze-lim=5; size-factor=1; factor-temp=0.95 and t-lim=0.01. As initial feasible solution we used the one obtained by Greedy-Prim heuristic. As starting temperature t_0 we used $t_0 = 2(c_{\max} - c_{\min})$ because a feasible solution differs from a neighbor in two edges at the most. As equilibrium criteria we used $L = 2n - 3$ iterations for each temperature level according to the

Problem	Stars cost	t (sec)	Greedy-Prim cost	t (sec)	Greedy-K cost	t (sec)
100-1	2300	4	696	8	1884	300
100-2	2128	4	714	8	1615	360
100-3	1869	4	605	8	1581	940
100-4	1979	3	716	8	1593	630
100-5	2049	4	710	9	1709	445
100-6	2021	4	769	9	1694	434
100-7	2253	4	709	8	1735	390
100-8	2168	4	723	8	1722	622
100-9	2299	4	701	8	1827	346
100-10	1977	4	689	8	1653	360

Table 8.1 Comparison among 3 simple heuristics for MS2T.

number of edges a feasible solution has. Our stopping rule was the same used by Johnson et al. [10]. After every temperature reduction we switch the type of neighbors used.

Table 8.2 shows the best SA results found and compared with the heuristic Greedy-Prim. The SA results include two cases: only edge based neighbors were used and then the two types of neighbors were used.

Prob	Greedy-Prim	Best SA	Prob	Greedy-Prim	Best SA
10-1	282	273	25-1	302	273
10-2	216	206	25-2	326	326
10-3	277	265	25-3	369	365
10-4	175	164	25-4	320	308
10-5	181	174	25-5	318	318
10-6	185	183	25-6	347	329
10-7	166	166	25-7	319	308
10-8	178	165	25-8	343	340
10-9	184	166	25-9	264	264
10-10	231	207	25-10	324	292
50-1	467	454	100-1	696	693
50-2	517	493	100-2	714	704
50-3	479	479	100-3	605	601
50-4	483	483	100-4	716	699
50-5	487	475	100-5	710	710
50-6	499	490	100-6	769	759
50-7	450	438	100-7	709	687
50-8	475	474	100-8	723	722
50-9	438	438	100-9	701	695
50-10	476	475	100-10	689	682

Table 8.2 Comparison between Greedy-Prim and best SA solution.

From Table 8.2 it is clear that for almost every test instance (80%), SA outperforms Greedy-Prim. This means that SA normally improves the starting solution given by Greedy-Prim.

We also studied the dependency on the starting solution in our SA approach. Table 8.3 presents SA results considering two starting solutions: Solution given by Stars heuristic and given by Greedy-Prim. \bar{x} -S and \bar{x} -G denote the average of SA results over a group (10 instances) considering the starting solution given by Stars heuristic and Greedy-Prim, respectively. \bar{t} denotes the time average over a group. The fifth column denotes, for each group, the number of instances for which SA reaches the best solution available, and finally, the last column shows best \bar{x} , the group average of SA results, but considering now only the best result on each instance (from 5 repetitions).

Problems	\bar{x} -S	\bar{x} -G	\bar{t} -S	\bar{t} -G	best-S	best-G	best \bar{x} -S	best \bar{x} -G
10	198.5	198.7	16	12	$\frac{8}{10}$	$\frac{9}{10}$	197.0	196.9
25	358.1	321.1	98	86	$\frac{1}{10}$	$\frac{9}{10}$	340.2	316.4
50	619.7	475.3	404	295	$\frac{1}{10}$	$\frac{9}{10}$	599.4	470.2
100	946.7	702.0	1697	1560	$\frac{1}{10}$	$\frac{7}{10}$	929.2	699.3

Table 8.3 Studying dependency on starting solution.

From Table 8.3 it is clear that SA starting with the solution given by Greedy-Prim outperforms SA starting with the solution given by Stars heuristic. However, this table also shows another facet of the SA approach; SA approach is able to *strongly* improve the very bad solution given by Stars heuristic (showed in Table 8.1 for $n = 100$). Unfortunately, it is not able to reach the results given by SA starting with Greedy-Prim and also it is more time-consuming. Therefore, our SA approach shows a dependency on the starting solution.

We also analized the dependency on the neighbors used. Table 8.4 shows SA results obtained in two cases: when only edge based neighbors are used (-1 in Table 8.4) and when the two types of neighbors are alternately used (-2 in Table 8.4). The starting solution used was Greedy-Prim in both cases.

Problems	\bar{x} -1	\bar{x} -2	\bar{t} -1	\bar{t} -2	best-1	best-2	best \bar{x} -1	best \bar{x} -2
10	201.5	198.7	6	12	$\frac{5}{10}$	$\frac{9}{10}$	199.2	196.9
25	322.6	321.1	22	86	$\frac{1}{10}$	$\frac{9}{10}$	319.5	316.4
50	477.1	475.3	56	295	$\frac{1}{10}$	$\frac{9}{10}$	473.8	470.2
100	703.2	702.0	222	1560	$\frac{1}{10}$	$\frac{9}{10}$	701.2	699.3

Table 8.4 Studying dependency on the neighborhood used.

We note that all SA results are clearly in favor of using two types of neighbors. However, it is also clear that times strongly increase; this trade-off is very common in SA algorithms. Very analogous results were obtained when the starting solution Greedy-Prim was changed to Stars heuristic.

8.7 CONCLUSIONS AND FUTURE WORK

We have described an application of Simulating Annealing algorithm to the Minimum Spanning 2-tree problem, which has an interesting interpretation in the field of Topological Network Design.

For a set of 40 test instances, the proposed SA approach outperforms previous simple heuristics (Greedy-Prim, Greedy-K and Stars). The dependency on the starting solution was also studied; two initial solutions of very different cost values were considered. The SA version starting with the solution of high cost was able to strongly improve that value but never near the solution given by the SA version starting with the good solution (Table 8.3). With respect to the dependency on the neighborhood used, better results were obtained when the two types of neighbors were considered; furthermore, the difference is consistent for all instance groups. However, it is also clear that time strongly increase.

As future work it would be desirable to have a good lower bound available for the optimal value; for example, following the ideas developed in [3]. In their study, they used the percentage above the lower bound δ , where:

$$\delta = 100 * (\text{best heuristic solution-lower bound})/\text{lower bound}.$$

However, the average difference between upper and lower strongly increases with the number of nodes. Therefore, that process needs to be improved. The design of other metaheuristics (Tabu Search, Genetic Algorithms, GRASP, etc.) for studying better solutions for MS2T is welcome.

Acknowledgments: The authors gratefully acknowledge discussions with Professor Nelson Maculan who provided us Bern's thesis. We also thank Professor Pavol Hell for a stimulating discussion about k -trees and for providing us some papers. The suggestions of the referees have led to significant improvements in the paper.

References

- [1] E. Aarts and J.K. Lenstra. *Local Search in Combinatorial Optimization*. Wiley, 1997.
- [2] H. Beck and A. Candia. Minimum Spanning k -Trees. In: *Proceedings of the XV International Conference of the Chilean Computer Society*, pages 71–79, 1995.
- [3] H. Beltrán and D. Skorin-Kapov. On Minimum Cost Isolated Failure Immune Networks. *Telecommunication Systems*, 3:183–191, 1994.
- [4] M. Bern. *Networks Design Problems: Steiner Trees and Spanning k -Trees*. Ph.D. Thesis, UCLA, 1987.
- [5] L. Cai and F. Maffray. On the Spanning k -Tree Problem. *Discrete Applied Mathematics*, 44:139–156, 1993.

- [6] E.W. Dijkstra. A Note on Two Problems in Connection with Graphs. *Numerische Mathematik*, 1:269–271, 1959.
- [7] A. Farley. Networks Immune to Isolated Failures. *Networks*, 11:255–268, 1981.
- [8] D. Fernández-Baca and A. Medepalli. Allocating Modules to Processors in a Distributed System with Limited Memory. In: *Proceedings of the XI International Conference of the Chilean Computer Science Society*, pages 349–369, 1991.
- [9] D. Granot and D. Skorin-Kapov. On Some Optimization Problems on k -Trees and Partial k -Trees. *Discrete Applied Mathematics*, 48:129–145, 1994.
- [10] D. Johnson, C. Aragón, L. McGeoch, and C. Schevon. Optimization by Simulated Annealing: An Experimental Evaluation, Part 1: Graph Partitioning. *Operations Research*, 37:865–892, 1989.
- [11] J.B. Kruskal, Jr. On the shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. *Proceedings of the American Mathematical Society*, 7:48–50, 1956.
- [12] G. Laporte and I. Osman (editors). Metaheuristics in Combinatorial Optimization. *Annals of Operations Research*, 63, 1996.
- [13] C. Monma and D. Shallcross. Methods for Designing Communications Networks with Certain Two-Connected Survavility Constraints. *Operations Research*, 37, 1989.
- [14] R. Prim. Shortest Connection Networks and Some Generalizations. *Bell Systems Technical Journal*, 36:1389, 1957.
- [15] J. Wald and C. Colbourn. Steiner Trees, Partial 2-Trees and Minimum IFI Networks. *Networks*, 13:159–167, 1983.

9

A GRASP INTERACTIVE APPROACH TO THE VEHICLE ROUTING PROBLEM WITH BACKHAULS

Carlos Carreto¹ and Barrie Baker²

¹Departamento de Informática
Instituto Politécnico da Guarda
Av. Francisco Sá Carneiro N 50, 6301-559 Guarda, Portugal
ccarreto@ipg.pt

²School of Mathematical and Information Sciences
Coventry University
Priory Street, Coventry CV1 5FB, England
b.baker@coventry.ac.uk

Abstract: Interactive methods for vehicle routing are appealing because they integrate the insight and experience of the user and the power and precision of heuristics in an interactive environment. Although best known results for benchmark problems are attributed to tabu search, interactive methods give results within about 3% of these solutions, whilst retaining the advantage of user control. A new interactive construction method using GRASP will be presented. The method allows vehicles to make all their deliveries, and then make pick-ups before returning to the depot. A Windows implementation allows the user to control the solution process by selecting initial outline routes of one or more deliveries followed by pick-ups. The user can also try to improve the solution, especially if it is infeasible, using interactive tools. Results will be shown for benchmark problems.

9.1 INTRODUCTION

The basic Vehicle Routing Problem (VRP) is a well known combinatorial optimisation problem. Because of its central importance in distribution management, it has been the subject of study for the last forty years. In this problem

a set of customers with known demands is to be serviced by a fleet of vehicles from a single distribution centre. The objective of the problem is to provide each vehicle with a sequence of deliveries so that all customers are serviced, and the total distance travelled by the fleet is minimised.

In our present study we consider an extension to the basic VRP, known as the Vehicle Routing Problem with Backhauls (VRPB), in which there is a mixed set of customers: linehaul and backhaul customers. Linehaul customers require the delivery of a given quantity of product from the depot, whereas a given quantity of product must be picked up from backhaul customers and transported to the depot.

The objective of the problem is to find a set of vehicle routes that service all the linehaul and backhaul customers such that: (i) each vehicle is associated with only one route; (ii) for each route the total loads associated with linehaul and backhaul customers do not exceed, separately, the vehicle capacity; (iii) the distance of each route does not exceed the maximum distance the associated vehicle can travel; (iv) on each route the backhaul customers, if any, are served after all linehaul customers; (v) the total distance travelled by the fleet is minimised.

The VRPB is NP-Hard, implying that the solution of realistic problem instances to optimality is likely to require an inordinate computational effort. As a consequence, when we have to solve large scale real life problems, we should not insist on finding an optimal solution, but instead try to find an acceptable solution within an acceptable amount of computational time. There has been considerable research effort in this area and many heuristic methods have been developed with the goal of obtaining an acceptable solution.

In the present study we investigate the incorporation of interactive tools into these algorithm paradigms. The aim of the interactivity is to integrate the insight and experience of the user, and the power and precision of the heuristics in an interactive environment. An interactive approach enables us to search for a solution by means of a trial and error procedure, in which man and machine divide the tasks in accordance with their best respective capabilities.

The machine is far superior at calculations and exhaustive search tasks. The human user is superior in judging fuzzy situations. Through interaction the user is able to control the solution process by selecting initial parameters, selecting algorithms and adjusting solutions. Furthermore, special knowledge of the real life problem can be included. In this way the user guides the heuristics towards promising parts of the solution space.

Interaction gives more flexibility in manipulating data, selecting alternatives and incorporating local knowledge during the search process. Therefore faster and better solutions can be obtained. Furthermore, an interactive system is more easily accepted, because the user feels that he is part of the solution process and sees the system as a versatile tool, not as a replacement, and solutions are also easy to accept because of the user input.

In the last decade we have witnessed rapid advances in information technology, which have resulted in enormous increases in processing power and man-machine interaction capabilities. It is now a feasible idea to involve humans throughout the solution process to solve routing problems. Baker [1], has shown that excellent results for the VRP can be obtained by an interactive approach which extended the Generalised Assignment approach of Fisher and Jaikumar [3]. This requires the user to start the solution process by selecting the initial number of vehicles and their customer seeds. In this paper, the approach is extended to include more powerful interactive tools in a Windows environment, incorporating some of the more recent developments in heuristics, and then applying them to the more difficult VRPB.

9.2 THE INTERACTIVE SOLUTION METHOD

The solution method we developed uses a clustering heuristic, requiring the user to select seeds and then the remaining customers are clustered according to the vehicles defined by these seeds, based on the position in each route where the insertion cost is minimised. Allocation priority is given to customers with a more obvious insertion route, and to customers that can only go on a small number of routes.

Three different types of customers can arise in a VRPB instance: linehaul customers, backhaul customers, and mixed customers that are at the same time both linehaul and backhaul customers, i.e., customers with both delivery and pickup demands. The solution method, however, only considers two types of customers, linehaul and backhaul customers. Any customer with both delivery and pickup demands is split into a linehaul and a backhaul customer with the same location. These two customers are treated separately by the solution method and can end up being served by different vehicles. Figure 9.1 shows how these three types of customers are represented in the interactive setting. Linehaul customers are represented by red triangles with their apex pointing up, and backhaul customers are represented by blue triangles with their apex pointing down. The area of each triangle is proportional to the corresponding demand.

The vehicle routes we end up with are most likely to be mixed vehicle routes, i.e., routes which visit both linehaul and backhaul customers. The construction of these mixed routes is implicitly oriented due to precedence constraint (iv). This precedence constraint is implicitly implemented in the interactive setting and through the solution method. The basis for this implementation is to consider that each route has two parts: a linehaul part where the linehaul customers are served and a backhaul part which serves the backhaul customers. At the end of the solution process we can have three different types of routes: mixed routes that serve linehaul and backhaul customers, linehaul routes that only serve linehaul customers, and backhaul routes that only serve backhaul customers.

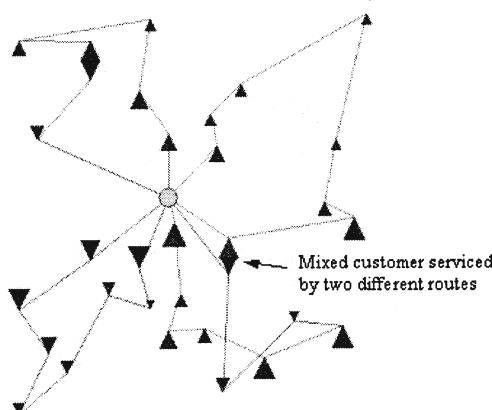


Figure 9.1 Example of a VRPB instance with the three types of customers and the three types of routes.

In order to distinguish better between the two parts of a route in the interactive setting, each part is represented by a different colour. The linehaul part that corresponds to the path between the depot and the last linehaul customer, is represented by a red line. The path between the last linehaul customer and throughout the backhaul customers until the depot, is the backhaul part and is represented by a blue line. These lines normally connect to the central point in the base of each customer triangle. For instances with linehaul and backhaul customers at the same location, it can be hard to identify the two parts of the route because the lines can end up on top of each other. To solve this problem the user can choose a route representation where lines connect to the apex of each triangle instead of the base of the triangle. Although connecting to the apex of the triangle introduces some error into the representation of the routes, since the location of the customer is at the base of the triangle, in practice this error does not affect the effectiveness of the interactive solution method. Instead, it permits a more effective use of the interactive tools. In the solution shown in Figure 9.1 we can see examples of the three different types of routes with a route representation connecting the apex of each triangle.

We identify three different phases in our interactive solution method: *Seeds Selection*, *Routes Construction and Improvement*, and *Solution Refinement*. The next sections describe these phases.

9.2.1 Seed selection

The seeds are the customers that for some reason the user forces to be serviced by specific vehicles. The user selects the seeds for each vehicle in accordance with his local knowledge and experience of a real life problem, or by identifying certain patterns such as clusters of customers, customers with very high demands or isolated customers. During the selection process the user is helped

by the computer, which checks for capacity or distance constraint violations. The computer also checks for any precedence constraint violation, i.e., the user can select a mix of linehaul and backhaul customers as seeds of a specific route, but the selection of linehaul customers after the selection of backhaul customers is not allowed.

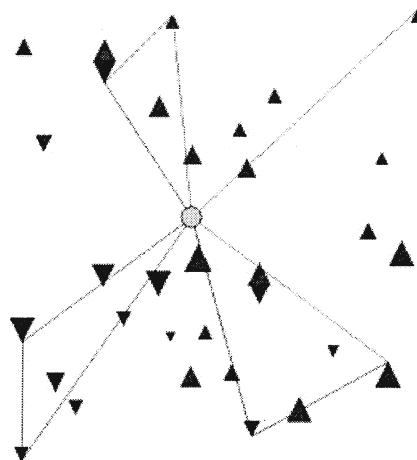


Figure 9.2 Selection of seven seeds.

The selection of the seeds can be seen as a critical phase since the seeds are the starting point for the solution method. If they are not well chosen, the solution will be poor or infeasible. It is also the seed selection that determines the number of vehicles used. Although being a critical phase, the interaction makes it easy for the user to try different seeds in order to obtain a satisfactory solution. The user can do this by selecting completely new seeds or making minor adjustments during the solution process as explained in Section 9.2.3. Figure 9.2 shows the 7 selected seeds for the solution shown in Figure 9.1.

9.2.2 Route construction and improvement

The routes construction and improvement phase clusters the remaining customers according to the vehicles defined by the seeds while applying a modified 3-optimal heuristic in order to reduce the total distance travelled by each vehicle. This clustering is done by a new heuristic, without user intervention, based on a GRASP approach. The heuristic may find a feasible solution where all the customers have been allocated to the vehicles; or the clustering may result in an infeasible solution, because there was a violation of a capacity or distance constraint.

The clustering heuristic allocates each customer to the position in a route where the insertion cost is minimised, giving allocation priority to customers

with a more obvious insertion route, and to customers for which the number of routes they can go on is smallest.

Because of the precedence constraint, the minimum insertion cost is determined taking into account the different types of customers and the different types of routes.

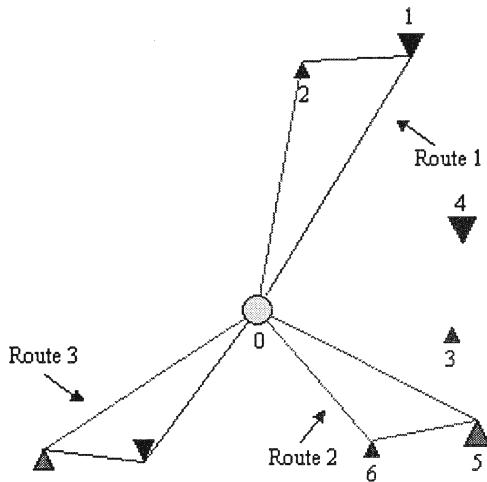


Figure 9.3 VRPB example problem.

Let C_{rj} be the minimum cost of inserting customer j in route r , and d_{ij} be the distance between customer i and customer j . Let P_{rj} be the insertion position corresponding to the minimum insertion cost.

Figure 9.3 shows an example problem where there are 6 seeds that define three routes, and there are two unallocated customers of different types.

The minimum cost of inserting customer 4 in route 1 is now given by:

$$C_{14} = \min\{(d_{04} + d_{41} - d_{01}), (d_{14} + d_{24} - d_{12})\}. \quad (9.1)$$

Notice that route 1 is a mixed route with a delivery part and a pickup part. Because customer 4 is a backhaul customer, we do not consider the cost of inserting customer 4 between the depot and customer 2 because the insertion of customer 4 in that position would violate the precedence constraint.

In a similar manner, the minimum cost of inserting customer 3 in route 1 is given by:

$$C_{13} = \min\{(d_{03} + d_{23} - d_{02}), (d_{13} + d_{23} - d_{12})\}. \quad (9.2)$$

Notice that customer 3 is a linehaul customer and therefore its insertion between customer 1 and the depot would violate the precedence constraint.

For routes with only one type of customer, such as route 2, the minimum insertion cost for customers of a different type, is calculated by considering the insertion positions between the extreme customers of the route and the depot.

This is possible because the direction of travel is arbitrary for a route with only one type of customer. Therefore, the minimum cost of inserting customer 4 in route 2 is given by:

$$C_{24} = \min\{(d_{04} + d_{54} - d_{05}), (d_{04} + d_{46} - d_{06})\}. \quad (9.3)$$

We do not consider the insertion of customer 4 between customers 5 and 6 because this would violate the precedence constraint.

During the calculation of C_{rj} , the clustering heuristic checks for constraint violations. If the weight of customer j , w_j , is bigger than the remaining delivery capacity of route r , or if C_{rj} is bigger than the remaining delivery distance of route r , than the insertion of customer j in route r is prevented by defining $C_{rj} = \infty$. We should note that if there is a very small violation of the distance constraint, there is a possibility that re-optimisation of the route at this stage would reduce the distance to a feasible value. However, the heuristic does not check for this possibility.

The clustering heuristic allocates one customer at a time in a sequential order determined by a greedy function, which gives allocation priority to customers with a more obvious insertion route. This is performed by calculating the biggest differences between the second smallest and the smallest insertion costs. Let S_j^1 and S_j^2 be, respectively, the smallest and the second smallest insertion costs for customer j . And let D_j be the difference between S_j^2 and S_j^1 .

In Figure 9.3, S_3^1 corresponds to the insertion of customer 3 in route 2, and S_3^2 corresponds to its insertion in route 1. It is clear from the figure that S_3^2 is much bigger than S_3^1 . For customer 4, S_4^1 corresponds to its insertion in route 1, and S_4^2 corresponds to its insertion in route 2. From the figure we can see that the value of S_4^1 should be very similar to the value of S_4^2 . So, D_3 is bigger than D_4 . The smallest insertion cost for customer 3, is much smaller than its second alternative, when compared with customer 4, and therefore the clustering heuristic allocates customer 3 first, since its insertion route is more obvious.

The procedure just described is performed also taking into account the maximum number of routes each customer can go on, i.e., the clustering heuristic gives allocation priority to customers for which the number of routes they can go on is smallest. Customers that can go on only one route should be inserted at the beginning of the construction while the route constraints retain a larger amount of slack.

Each time a customer is inserted in route r , all the minimum insertion costs C_{rj} and corresponding insertion positions P_{rj} must be recalculated for each customer j that is still not allocated. This process is repeated until all customers are allocated, or no further allocations are possible.

9.2.3 Solution refinement

After the routes construction and improvement phase, the user can try to refine the solution in order to improve it (especially if it is infeasible), by using interactive tools. With no formal rules to follow, the interactive system we

developed is sufficiently flexible to allow creativity and imagination from the user, and different refinement techniques are possible for using these tools. The simplest example consists of the removal of some customers from their current routes. This is followed by a new call to the construction and improvement heuristics, allowing these customers to be inserted into different routes. This is a very effective procedure when there is some overlap between routes, and can lead to several different feasible solutions in a very short period of time.

The following three figures illustrate this simple interactive improvement procedure. Figure 9.4 shows a possible solution where we can identify two fuzzy areas, represented in the figure by the two circles.

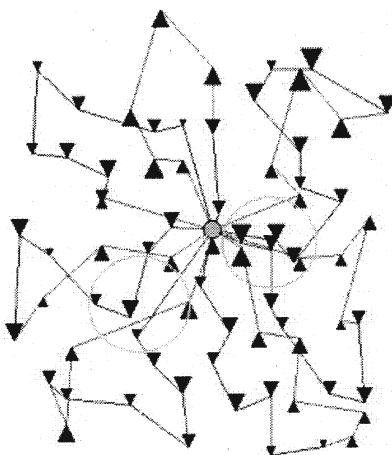


Figure 9.4 Solution obtained with two fuzzy areas.

We can see that the clustering heuristic had a poor performance in these areas and as a result there is some overlap between routes. These areas are very easily identified by the user.

Figure 9.5 illustrates the application of the technique described above where some customers have been removed from their current routes, around the fuzzy areas.

Figure 9.6 shows the solution obtained after a new call to the routes construction and improvement heuristics using the situation of Figure 9.5 as the starting point. In this case the solution improved considerably, as we can see in Figure 9.6. In practice the user may not obtain such good results immediately. A typical situation is one where the user has to experiment a few times to find an improvement.

Another possibility is removing two complete routes where the structure looks poor, and then inserting two seeds at the extreme edges of the corresponding region. This technique might extend to removing more than two routes.

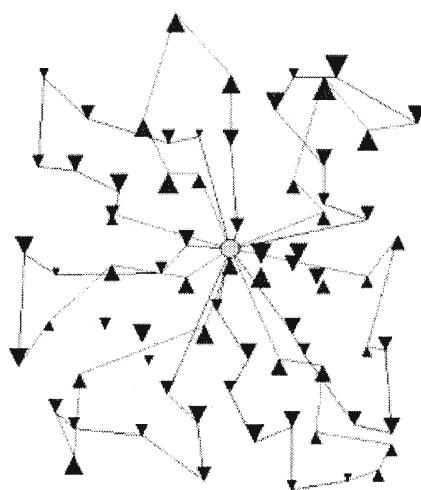


Figure 9.5 Removal of some customers from their current routes.

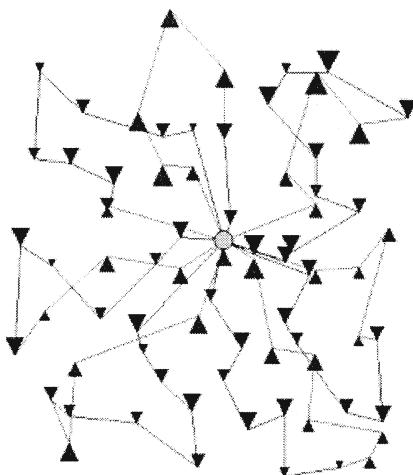


Figure 9.6 New solution after a new call to the routes construction and improvement heuristics.

There are no formal rules to identify the customers or routes that should be removed. The user is provided with interactive tools to analyse and edit the routes, and he uses his insight and experience to identify specific areas where customer removal should take place.

These techniques can be used to obtain several different solutions in a very short time. However, when the constraints are tight, this technique might not

be sufficient and several restarts with different seeds might be necessary to produce a feasible solution. To help with the search for a feasible solution, we implemented an automatic restart procedure for the routes construction and improvement phase, based on a GRASP approach.

The GRASP approach is ideal for the route construction and improvement phase because it is a simple method of getting some reasonable results quite quickly, with the option of allowing it to run longer in order to get better results, and because it is also easy to integrate in the interactive setting and easy to “hide” from the inexpert user.

9.3 THE GRASP APPROACH IN THE INTERACTIVE SETTING

GRASP (Greedy Randomised Adaptive Search Procedure), is a restart procedure where each iteration consists of a construction phase, followed by a local search phase that finds a locally optimal solution. See Resende [9], and Festa and Resende [2], for a survey of GRASP. This method has been applied with success to the VRP by Hjorring [6], and Kontoravdis and Bard [7].

In our approach the solution obtained in the construction phase can be either feasible or infeasible. We are interested in saving the best feasible solution at each iteration, but if the heuristic fails to find a feasible solution, the best infeasible solution is displayed as it gives the user something to start with in the solution refinement phase. The best infeasible solution is defined as the infeasible solution with the minimum total weight unallocated, as this is considered to provide the best potential for finding a feasible solution.

Each iteration of the GRASP algorithm starts with the initial seeds defined by the user. A greedy randomised solution x is obtained in a construction phase, followed by a local search phase that finds a local minimum x' starting from x .

Let $C_f(x')$ and $C_i(x')$ be two functions which are defined for feasible and infeasible solutions, respectively. $C_f(x')$, gives the total distance travelled by the vehicles, and $C_i(x')$ gives the total weight unallocated. The interactive GRASP approach is described in Algorithm 1.

Algorithm 1 - Interactive GRASP approach {

```

    user selects initial seeds;
    best_feasible_cost = ∞;
    best_infeasible_cost = ∞;
    repeat
        start with the initial seeds;
        construct greedy randomised solution x;
        find local minimum x' starting from x;
        if (x' is feasible)
            if (Cf(x') < best_feasible_cost) {
                x* = x';
                best_feasible_cost = Cf(x');
                display x*;
```

```

        }
    else
        if (( $C_i(x') < \text{best\_infeasible\_cost}$ ) and ( $x^* = \{\}$ )) {
             $x^{**} = x'$ ;
             $\text{best\_infeasible\_cost} = C_i(x')$ ;
            display  $x^{**}$ ;
        }
    until (stopping criterion is true)
}

```

The construction phase is implemented in the clustering heuristic described in Section 9.2.2.

At each iteration, let R_j be the number of routes customer j can go on. If there are customers with R_j equal to one, the heuristic selects the one with the biggest difference, D_j , between the smallest and the second smallest insertion costs. If there are no customers with R_j equal to one, the heuristic selects the customer to insert from a restricted candidate list (RCL). The customer selected in this way, j^* , is inserted in position $P_{r^*j^*}$ corresponding to the minimum insertion cost, where i^* is the route with smallest insertion cost for customer j^* .

Algorithm 2 - Clustering heuristic of the construction phase {

```

    initialise  $C_{rj}$  and  $P_{rj}$  for each unallocated customer  $j$ 
    and route  $r$ ;
    initialise  $D_j$  and  $R_j$  for each unallocated customer  $j$ ;
    initialise  $R^0$  and  $R^1$ ;
    while (customer allocations are still possible) {
        if ( $R^0 > 0$ ) and (a feasible solution has been found)
            terminate construction phase;
        if ( $R^1 > 0$ )
             $j^* = \text{customer } j \text{ with biggest } D_j \text{ and } R_j = 1$ ;
        else
             $j^* = \text{random selection from the RCL}$ ;
             $r^* = \text{route with smallest insertion cost for } j^*$ ;
            insert customer  $j^*$  in position  $P_{r^*j^*}$ ;
            mark customer  $j^*$  as being allocated;
            update remaining capacity and distance for route  $r^*$ ;
            apply 3-optimal to route  $r^*$  if necessary;
            update  $C_{r^*j}$ ,  $P_{r^*j}$ ,  $D_j$  and  $R_j$  for each unallocated
            customer  $j$ ;
            update  $R^0$  and  $R^1$ ;
    }
    apply 3-optimal to each route;
}

```

The insertion of customers proceeds until all customers are allocated, or no further allocations are possible.

The clustering heuristic of the construction phase is presented in detail in Algorithm 2, where R^0 and R^1 are, respectively, the number of customers that can no longer go on any route, and the number of customers that can go on only one route.

If R^0 is greater than zero, then no feasible solution is possible. However, further allocations may be possible and, if no feasible solution has been found at a previous GRASP iteration, the construction phase continues searching as the best infeasible solution could still be improved. If a feasible solution has been found at a previous iteration, then the construction phase terminates.

As explained before, customers are allocated sequentially giving priority to those that can only go on a small number of routes, and those with the biggest differences between the smallest and second smallest insertion costs. Therefore, customers that can go on only one route are allocated first. If there are no such customers, the clustering heuristic selects the next customer to insert from a restricted candidate list (RCL).

Let k be the chosen length of the RCL and T be the subset of customers j , not yet allocated to a route, and with R_j equal to two.

If T is empty and more than k customers remain unallocated, then the RCL consists of the k customers with biggest D_j , and the next customer to allocate is selected at random from the RCL. Each customer in the RCL can be selected with a probability proportional to their demand, given by

$$\frac{w_j}{\sum_j w_j}, \quad (9.4)$$

where w_j is the demand for customer j and the summation is over those customers in the RCL. This gives some priority to customers with bigger demands since, as the routes become full, it becomes more difficult to find a route which these customers can go on.

Algorithm 3 - Selection from the restricted candidate list {

```

 $T = \{\text{unallocated customers with } R_j = 2\};$ 
if ( $|T| = 0$ ) {
    if (more than  $k$  customers remain unallocated)
        construct RCL with  $k$  customers with biggest  $D_j$ ;
    else
        construct RCL with all unallocated customers;
        assign probabilities proportional to their demands;
     $j^* = \text{customer selected at random from RCL};$ 
}
else
    if ( $|T| = 1$ )
         $j^* = \text{element in } T;$ 
    else {
        if ( $|T| > k$ )
```

```

    construct RCL with  $k$  customers in  $T$  with biggest  $D_j$ ;
else
    construct RCL with all customers in  $T$ ;
assign probabilities proportional to their demands;
 $j^*$  = customer selected at random from RCL;
}
}

```

If T is not empty and $|T| > k$, the RCL consists of the k customers from T with biggest D_j . The method of selecting the next customer to allocate from the RCL is the same as when T is empty. If $|T| \leq k$, the RCL consists of all customers in T , and if T only has one element, this customer is selected. Algorithm 3 describes this selection procedure.

At the end of the construction phase, the 3-optimal heuristic is applied again to each route. The application of the 3-optimal heuristic at this stage could be seen as the local search phase of the GRASP approach. But although this procedure is necessary to guarantee 3-optimality in each route, it is a very poor local search phase, since it acts on each route individually. Our next step in the development of the interactive GRASP approach will be to implement a local search phase capable of a complete rearrangement of the solution structure given by the construction phase, with the possible exception of the original seeds structure.

The 3-optimal heuristic called during the clustering process, is a local search heuristic based on an arc exchange procedure. Some of these arc exchanges can result in a violation of the precedence constraint. To avoid this, the 3-optimal heuristic was modified in order to check for precedence constraint violation each time an arc exchange is tried.

9.4 COMPUTATIONAL RESULTS

The GRASP interactive approach (GIA), was tested on 14 benchmark problems selected from a total of 62 published by Goetschalckx and Jacob-Blecha [5]. The 62 problems are divided into 14 groups. The problems in each group are exactly the same except that they have different vehicle capacity and distance limits. We selected one problem from each group corresponding to the one with the fewest vehicles.

All distances were calculated as real values without truncation and the final solution was rounded to the nearest integer. The results were obtained on a Pentium II running at 300MHz. The application was written in Borland C++ Builder 3.

Table 9.1 gives the computational results for the fourteen test problems obtained by our interactive approach. For each problem the columns give the number of linehaul customers L , the number of backhaul customers B and the number of vehicles used K . The last three columns compare the best solutions obtained by the GIA with the best published solutions given by Toth and Vigo [10]. The column denoted by % shows the percentages above best published

solutions of our best results. The solutions for the 14 test problems were obtained in interactive sessions of between 2 and 5 minutes for each problem. The 14 interactive sessions totalled 40 minutes.

Name	<i>L</i>	<i>B</i>	<i>K</i>	best published	GIA	%
<i>A</i> ₄	20	5	3	155795	155795	0.00
<i>B</i> ₃	20	10	3	169368	169372	0.00
<i>C</i> ₄	20	20	4	195365	200899	2.83
<i>D</i> ₄	30	8	5	205834	205834	0.00
<i>E</i> ₃	30	15	4	206658	208837	1.05
<i>F</i> ₄	30	30	4	233861	237089	1.38
<i>G</i> ₆	45	12	4	213457	219740	2.94
<i>H</i> ₃	45	23	4	247449	249022	0.64
<i>I</i> ₄	45	45	6	295988	300425	1.50
<i>J</i> ₃	75	19	6	282535	285160	0.93
<i>K</i> ₄	75	38	7	359642	366727	1.97
<i>L</i> ₄	75	75	7	390247	396937	1.71
<i>M</i> ₄	100	25	7	356311	358634	0.65
<i>N</i> ₆	100	50	8	384461	391353	1.79

Table 9.1 Test problems proposed by Goetschalckx and Jacob-Blecha.

Table 9.1 shows that our interactive approach performed quite well, obtaining solutions comparable in quality to the best published. The average is 1.24% above the best published solutions.

9.5 CONCLUSION

Our interactive method retains the advantages of user control, allowing the insight and experience of the user to be combined with the power and precision of modern heuristics. The method has given results averaging 1.24% above the best known solutions for benchmark problems, even without the full local search phase in the GRASP being implemented. We expect that after implementing a more powerful local search phase, even better results will be achieved.

References

- [1] B.M. Baker. Further Improvements to Vehicle Routing Heuristics. *JORS*, 43:1009–1012, 1992.
- [2] P. Festa and M. Resende. GRASP: An Annotated Bibliography. In: *Essays and Surveys on Metaheuristics*, P. Hansen and C.C. Ribeiro, editors, Kluwer, 2001 (this volume).
- [3] M. Fisher and R. Jaikumar. A Generalised Assignment Heuristic for Vehicle Routing. *Networks*, 11:109–124, 1981.
- [4] M. Gendreau, A. Hertz, and G. Laporte. The Traveling Salesman Problem with Backhauls. *Computers and Operations Research*, 23:501–508, 1996.

- [5] M. Goetschalckx and C. Jacobs-Blecha. The Vehicle Routing Problem with Backhauls. *European Journal of Operational Research*, 42:39–51, 1998.
- [6] C.A. Hjorring. *The Vehicle Routing Problem and Local Search Metaheuristics*. PhD Thesis, University of Auckland, 1995.
- [7] G. Kontoravdis and A.J.F. Bard. A GRASP for the Vehicle Routing Problem with Time Windows. *ORSA Journal on Computing*, 7:10–23, 1995.
- [8] N. Mladenovic and P. Hansen. Variable Neighborhood Search. *Computers and Operations Research*, 24:1097–1100, 1997.
- [9] M.G.C. Resende. Greedy Randomized Adaptive Search Procedures (GRASP). To appear in: *Encyclopedia of Optimization*, C. Floudas and P.M. Pardalos, editors, Kluwer.
- [10] P. Toth and D. Vigo. A Heuristic Algorithm for the Symmetric and Asymmetric Vehicle Routing Problems With Backhauls. *EJOR European Journal of Operational Research*, 113:528–543, 1999.

10 PARALLEL COOPERATIVE APPROACHES FOR THE LABOR CONSTRAINED SCHEDULING PROBLEM

Cristina C.B. Cavalcante¹, Victor F. Cavalcante¹,
Celso C. Ribeiro², and Cid C. de Souza¹

¹Institute of Computing
Universidade Estadual de Campinas
Caixa Postal 6176
Campinas, SP 13083-970, Brazil
{cris,vfc,cid}@ic.unicamp.br

²Catholic University of Rio de Janeiro
Department of Computer Science
R. Marquês de São Vicente 225
Rio de Janeiro 22453-900, Brazil
celso@inf.puc-rio.br

Abstract: In this paper we consider the labor constrained scheduling problem (LCSP), in which a set of jobs to be processed is subject to precedence and labor requirement constraints. Each job has a specified processing time and a labor requirements profile, which typically varies as the job is processed. Given the amount of labor available at each period, the problem consists in determining starting times so as to minimize the overall makespan, subject to the precedence and labor constraints. We propose two parallel cooperative algorithms for LCSP: an asynchronous team and a parallel tabu search strategy. Both algorithms make use of cooperative processes that asynchronously exchange information gathered along their execution. Computational experiments on benchmark instances show that these parallel algorithms produce significantly better solutions than all sequential algorithms previously proposed in the literature.

10.1 PROBLEM DEFINITION

In broad terms, the labor constrained scheduling problem (LCSP) involves sequencing a set of jobs on multiple machines, subject to precedence constraints given in terms of a direct (acyclic) graph. Each job has a specified processing time and a labor requirements profile, which typically varies as the job is processed. Given the amount of labor available at each time period, the problem consists in finding a job sequence within each machine, so as to minimize the completion time of the last job to be finished (i.e., the makespan), subject to the precedence and labor constraints.

The particular version of LCSP motivating this study is a simplification of an industrial problem from BASF A.G. This problem is NP-hard and was first discussed in [20]. Since then, some methods have been proposed for its solution: constraint programming [21], linear programming-based heuristics [28, 29], sequential tabu search [5], and integer programming [30]. The last three approaches are discussed in [6].

The main features of the LCSP instances from the industrial application cited above are the following. We are given a set J of n jobs. Each job j is formed by a sequence of p_j unit-duration tasks. All such tasks must be processed immediately one after the other and therefore, job preemption is not permitted. The labor requirements profile is specified by a vector $(\ell_{j,1}, \dots, \ell_{j,p_j})$, where $\ell_{j,s}$ denotes the numbers of workers needed to perform the s -th task of job j . Typically labor requirements take values in the set $\{2, 3, 6, 9, 12, 18\}$ while the total amount of workers L available at any time instant, is either 18 or 24. The jobs are partitioned into groups called *orders*. An order is composed by a set of jobs with identical labor profiles and linked by a long path in the precedence graph. A few precedence constraints exist between jobs on different orders. Besides, all the jobs in an order are executed in a single machine and the total number of orders is small enough so as that each machine can be assigned to a unique order. All machines are considered to have the same speed.

A solution to LCSP is a schedule given by the starting times of all jobs in J . A schedule is feasible if the starting times of the jobs respect the precedence relations and, for each instant t in the planning horizon, the total amount of workers required by all jobs being executed at t is not greater than L . The problem is then to find a feasible scheduling that minimizes the makespan. Figure 10.1 illustrates a small example of such an instance.

We notice that there exists a *reverse instance*, associated to each instance of LCSP. The reverse instance is obtained from the original one by reversing the labor requirements profiles of all the jobs and the direction of all precedence constraints (arcs of the digraph). It is not hard to see that feasible schedules of the original instance and the reverse one are in a one-to-one correspondence. Thus, the optimal makespan remains the same independently of whichever of these two instances is solved. This fact is exploited in some algorithms for LCSP.

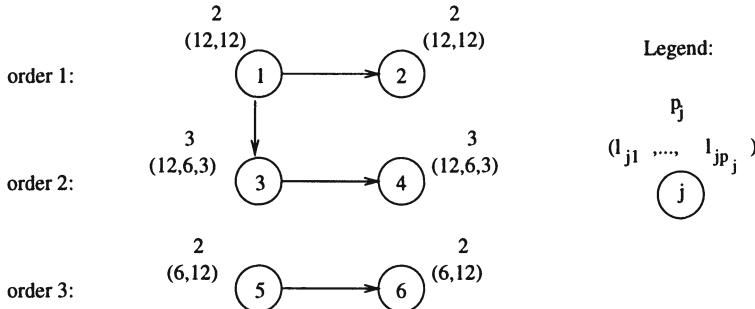


Figure 10.1 An instance of LCSP.

In this paper, we present two parallel cooperative algorithms for LCSP: an asynchronous team and a parallel implementation of tabu search. Both of them make use of cooperative processes that asynchronously exchange information gathered during their execution. The goal is to improve on the individual performance of each process by embedding them within a cooperative team of algorithms. The paper is organized as follows. Sections 10.2 and 10.3 describe, respectively, the asynchronous team and the parallel tabu search for LCSP. Section 10.4 presents some computational results obtained with the application of these two algorithms to a set of benchmark instances of LCSP. Finally, in Section 10.5 we give some concluding remarks.

10.2 AN ASYNCHRONOUS TEAM FOR LCSP

An *asynchronous team* (a-team) is a software organization in which multiple algorithms, each acting as an autonomous agent, cooperate with each other exchanging information through shared memories [31, 36] (see also [2, 11, 25, 32, 37] about references and applications of asynchronous teams, as well as [27] for a similar approach in another context). The main purpose of this cooperation is to increase the chances of finding better solutions than those produced by each agent acting alone. Usually, there are four types of agents in an a-team designed to solve a specific combinatorial problem:

- construction agents: algorithms that generate complete or partial solutions of the problem;
- de-construction agents: algorithms that generate a partial solution from one or more complete solutions;
- improvement agents: local search algorithms that modify complete solutions; and
- destroyer agents: algorithms that remove solutions from shared memories.

In a generic a-team, these four types of agents are continuously active: new solutions are built, existing solutions are de-constructed or modified, and solutions of low quality are destroyed. Shared memories store complete or partial solutions and can be accessed at any moment by all agents.

Graphically, an a-team is represented by a set of arcs and boxes, which refer to agents and memories, respectively. Figure 10.2 shows an example of a-team. Agents *A* and *B* read from memory 1 and write in memories 1 and 3 respectively. Agent *C* reads from memory 2 and writes in memory 1. Agent *D* reads in memory 3 and can write either in memory 1 or in memory 2 (this situation is represented by the large box containing both box 1 and box 2). Agent *E* reads from memory 2 and writes in memory 3. Agent *F* initializes memories 1 and 2, while agent *G* is a destroyer responsible for deleting solutions from those two memories.

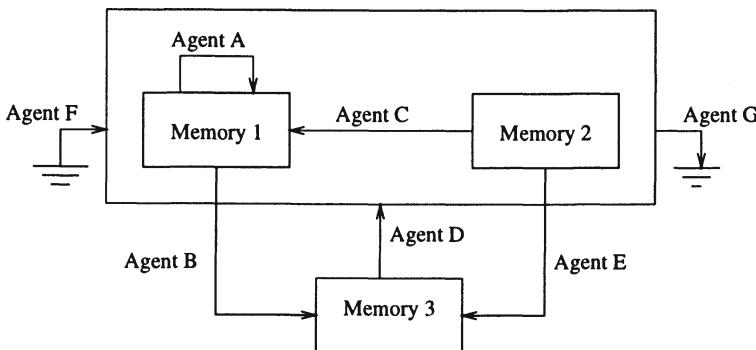


Figure 10.2 Example of an a-team.

An important characteristic of an a-team is that data (i.e., solutions of the problem to be solved) cycle within its structure and are renewed along this process. Agents retrieve, modify, and store information in shared memories. Thus, solutions generated by an algorithm are potentially reused as input for other algorithms of the team. This continuous flow of information allows the interaction between the agents and, eventually, leads to a refinement of the solutions generated.

It is worth noticing that the a-team structure is particularly well suited for distributed computing. The autonomy of the agents, the asynchronous communication, and the absence of a centralized control allow the simultaneous execution of the different algorithms on the many processors available in a computer network. Successful applications of this technique to other hard combinatorial problems were reported in [7, 19, 23, 31] and motivated its investigation in the context of the LCSP.

The a-team we proposed for LCSP is depicted in Figure 10.3. There are two shared memories: “Complete Solutions” memory (CSM) and “Partial Solutions” memory (PSM). An initializer agent generates the initial set of complete solutions. De-construction agents read solutions from CSM and generate

partial solutions that are stored in PSM. Construction agents read solutions from PSM, complete them, and write the new solutions obtained in CSM. Improvement agents read solutions from CSM, modify them, and write the improved solutions in CSM. A destroyer agent is responsible for removing bad solutions from CSM.

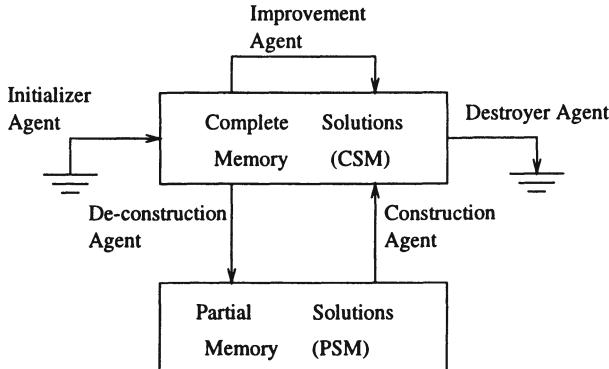


Figure 10.3 A-team for LCSP.

We associate a job sequence s with each solution of an LCSP instance. However, this sequence alone is not enough to represent a solution, unless a precise rule is given that generates a feasible schedule from it. Given a job sequence s , one can build a solution by scanning the jobs in the order they appear in s and scheduling them at the earliest possible time so that all precedence and labor constraints are satisfied. Another solution can be built by scanning the jobs in s in reverse order, considering the reverse instance. Finally, we associate with sequence s the schedule with minimum makespan among the two above described solutions. Thus, the cost of any given sequence s can be easily computed in time $O(nPT)$, where P and T denote respectively the duration of the longest job and the time horizon. However, we observed in practice that this cost computation takes linear time in the number of jobs n . In the remaining of the text, we also refer to a job sequence s as a solution for LCSP since, according with the above rule, it corresponds to a unique schedule of the jobs on the machines. Using this solution representation, the processes of the proposed a-team for LCSP are defined as follows.

10.2.1 Process ATC (complete solutions memory)

This process has five basic functions: to initialize and to maintain the CSM; to initialize all other processes of the a-team; to send complete solutions to de-construction and improvement agents; to receive complete solutions from construction and improvement agents; and to terminate the computation. These functions are detailed below.

CSM initialization. The CSM is maintained by ATC as a list with `CSM_SIZE` elements. The first step of ATC initializes the list with the following solutions: the best solution generated by a greedy heuristic based on a priority rule; the best solution obtained by a heuristic based on classes of schedules; and the remaining solutions generated by a heuristic with a randomized priority rule. Priority rule based greedy heuristics construct schedules from scratch. At each step, they select the next job to schedule as the one minimizing some pre-defined greedy function representing the priority rule. Typical functions (cf. [15, 22]) are:

- minimum slack: choose the job with minimum *slack time*, i.e., the job minimizing the difference between the latest and earliest starting times computed from the precedence relations and the planning horizon;
- longest critical path: choose the job which together with its successors in the precedence graph forms an induced subgraph with the largest critical path;
- longest order duration: choose the job belonging to the order with the largest total duration;
- most remaining order duration: choose the job belonging to the order with the largest remaining duration (i.e., the sum of the processing times of the currently unscheduled jobs in the order);
- shortest processing time: choose the job with the smallest processing time;
- least resource demand: choose the job requiring less workers; and
- random: randomly choose the job.

The initialization strategy allows for both diversity and good quality solutions to start the search. Solutions are stored in increasing order of their costs and no repetitions are allowed.

A-team initialization. After the initialization of the CSM, process ATC initializes the remaining processes of the team: ATP (partial memories); IBC and DBC (de-construction agents); HND, HH, and SPLCA (construction agents); and IMPJ, IMPCP, IMPALL, and IMPSWP (improvement agents). Next, ATC starts its autonomous execution by (i) waiting for new complete solutions sent by construction or improvement agents, or (ii) sending already solutions to de-construction or improvement agents.

Solution reception. Whenever a construction or improvement agent finds a good quality complete solution, a message is sent to process ATC containing solely the cost of this solution. If this cost is not larger than $1 + \gamma\%$ times the cost of the best solution in the CSM, then ATC sends a message to the calling agent informing that it is ready to receive the actual solution, represented by

a job sequence. The threshold parameter $\gamma > 0$ is used to avoid poor quality solutions in CSM, as well as to cut the amount of long messages sent to ATC containing solutions which could prevent it from quickly treating other messages requests. Solutions received by ATC which are already in CSM are discarded. Otherwise, a solution from CSM is chosen and replaced by the new one. The choice of the solution to be eliminated follows a linear distribution probability: the best solution has a null probability of being eliminated, while for the others this probability increases linearly up to the worst solution in CSM. Reports on computational experiments with other combinatorial problems show that this strategy maintains the diversity of the solutions in CSM, while keeping high quality solutions, see e.g. [7].

Solution sending. ATC sends messages with one complete solution to improvement agents and with two complete solutions to de-construction agents. The solutions to be sent are randomly chosen, so as that all solutions have equal chances of being treated by those agents.

Termination. The decision of when to terminate the computation is left to process ATC. Whenever **MAX_SOLUTIONS** have been accepted to enter into the CSM, ATC stops the execution of all remaining processes. Afterwards, it halts its own execution and returns the best solution in the CSM.

10.2.2 Processes IBC and DBC – De-construction agents

A de-construction agent starts with one or more complete solutions and produces a partial solution. In the case of LCSP, a partial solution is represented by a set of subsequences of jobs. The proposed a-team has two de-construction agents, both based on consensus algorithms [31]. A consensus algorithm tries to capture similarities or dissimilarities between two or more good quality solutions.

The first de-construction agent is implemented via the IBC process and is based on an *intersection criterion*, while the second is implemented by process DBC and is based on a *difference criterion*. These processes are continuously asking process ATC for complete solutions from CSM. The partial solutions generated are then sent to process ATP (cf. Subsection 10.2.3).

We now explain the two criteria used to implement the consensus algorithms. Let s_a and s_b be sequences representing two different solutions. The intersection criterion creates a set of subsequences in which s_a and s_b coincide. For the difference criterion, assume that s_a has a better cost than s_b . Then, a partial solution is created which contains the subsequences in s_a that do not appear in s_b . In both cases the comparison between the sequences is done in a position by position basis. Thus, it is straightforward to implement these algorithms with time complexity $O(n)$. These ideas are illustrated in Figure 10.4.

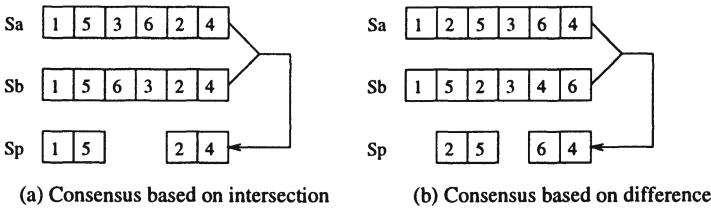


Figure 10.4 Generating a partial solution s_p using the intersection and difference consensus criteria.

10.2.3 Process ATP – PSM management

This process handles the management of the partial solutions memory. It acts like a buffer that receives partial solutions from the de-construction agents and, whenever requested, sends them to the construction agents. PSM is initially empty. Each element is composed by a set of subsequences of jobs. PSM grows as partial solutions are received from de-construction agents, and diminishes when solutions are sent to construction agents. Notice that it may be the case that a construction agent requests a partial solution for ATP when the PSM is empty, in which case the request cannot be served.

10.2.4 Processes HND, HH, and HA – Construction agents

The goal of the construction agents in the a-team is to build complete solutions, starting from partial ones. Jobs not belonging to the partial solution enter it on a one by one basis in an order determined by a complete solution previously generated by one of the heuristic methods specified below. Each of these jobs is inserted in the current partial solution in such a way that (i) subsequences from the partial solution are not broken by the insertion of new jobs, and (ii) the new job must be inserted prior to its successors and after all its predecessors.

We now describe the heuristics that generate the job sequence that guides the construction agents. Schedules produced by these methods can be classified as follows:

- non-delay schedules: schedules where no resource (worker) remains unused (idle) if there exists a job that can start earlier and use it; and
- active schedules: schedules where no job can start earlier without delaying the starting of another job or violating the labor constraints.

Figure 10.5 illustrates these two definitions for schedules generated for the example in Figure 10.1.

The heuristics are based on the classical algorithms of Giffler and Thompson (see e.g. in [3]) which generate respectively all active and non-delay schedules for the job shop scheduling problem. We have adapted them to return a single active or non-delay schedule for the LCSP. To describe the heuristics we introduce the following notation:

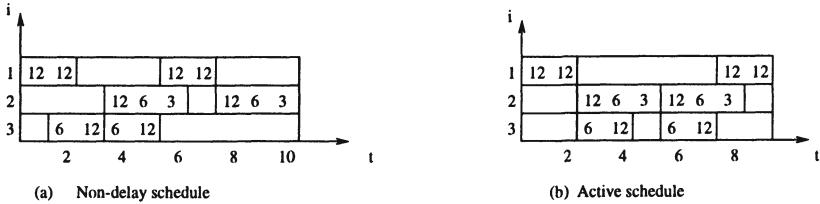


Figure 10.5 Classes of schedules.

σ_j : earliest time that job j can start, considering precedence and labor constraints;

$\phi_j = \sigma_j + p_j$: earliest time that job j can finish, considering precedence and labor constraints;

P_k : a partial schedule with k jobs; and

S_k : the set of schedulable jobs at step k , i.e. jobs whose predecessors already belong to P_k .

The pseudo-codes of heuristics HA and HND which produce, respectively, active and non-delay schedules for LCSP, are given below. We notice that both algorithms make use in step 4 of one of the priority rules described in Subsection 10.2.1 to break ties, whenever necessary.

Heuristic HA:

1. Set $k \leftarrow 0$, $P_k \leftarrow \emptyset$, and $S_k \leftarrow \{\text{jobs with no predecessors}\}$. Initialize the amount of resources available at any time instant $u = 1, \dots, T$ with $\text{labor}[u] = L$.
2. For all $j \in S_k$ do:

$$\sigma_j \leftarrow \max\{e_j, \min\{t : \ell_{j,u-t+1} \leq \text{labor}[u] \forall u = t, \dots, t + p_j - 1\}\}$$

$$\phi_j \leftarrow \sigma_j + p_j$$
3. Compute $\phi^* = \min\{\phi_j : j \in S_k\}$.
4. Choose $j' \in S_k$ such that $\sigma_{j'} < \phi^*$. Ties are broken by one of the priority rules of Subsection 10.2.1. If ties persist, they are broken by a random choice.
5. Build a partial schedule P_{k+1} by adding j' to P_k with starting time $\sigma_{j'}$.
6. (a) Set $\text{labor}[u] \leftarrow \text{labor}[u] - \ell_{j',u-\sigma_{j'}} \forall u = \sigma_{j'}, \dots, \sigma_{j'} + p_{j'} - 1$.

(b) Set $e_j \leftarrow \max\{e_j, \sigma_{j'} + p_{j'}\}$ for every successor j of job j' .

(c) $k \leftarrow k + 1$

(d) Set $S_k \leftarrow \{\text{all jobs not yet scheduled whose predecessors are in } P_k\}$.
7. If $k < n$ go back to step 2. Otherwise, stop.

Heuristic HND:

1. Set $k \leftarrow 0$, $P_k \leftarrow \emptyset$, and $S_k \leftarrow \{\text{jobs with no predecessors}\}$. Initialize the amount of resources available at any time instant $u = 1, \dots, T$ with $\text{labor}[u] = L$.

2. For all $j \in S_k$ do:
 - $\sigma_j = \max\{e_j, \min\{t : \ell_{j,u-t+1} \leq labor[u] \forall u = t, \dots, t + p_j - 1\}\}$
3. Compute $\sigma^* = \min\{\sigma_j : j \in S_k\}$.
4. Choose $j' \in S_k$ such that $\sigma_{j'} = \sigma^*$. Ties are broken by one of the priority rules of Subsection 10.2.1. If ties persist, they are broken by a random choice.
5. Build a partial schedule P_{k+1} by adding j' to P_k with starting time $\sigma_{j'}$.
6. (a) Set $labor[u] \leftarrow labor[u] - \ell_{j',u-\sigma_{j'}} \forall u = \sigma_{j'}, \dots, \sigma_{j'} + p_{j'} - 1$.
- (b) Set $e_j \leftarrow \max\{e_j, \sigma_{j'} + p_{j'}\}$ for every successor j of job j' .
- (c) $k \leftarrow k + 1$
- (d) Set $S_k \leftarrow \{\text{all jobs not yet scheduled whose predecessors are in } P_k\}$.
7. If $k < n$ go back to step 2. Otherwise, stop.

Finally, heuristic HH can be viewed as the generation of hybrid schedules between non-delay and active ones. This idea was introduced by Storer, Wu, and Vaccari [33] and implemented via an adaptation of the algorithms of Giffler and Thompson. This is done through the introduction of a variable $\delta \in [0, 1]$, such that for $\delta = 1$ the algorithm generates an active schedule, otherwise it generates a hybrid schedule. Details are given in the pseudo-code below. The motivation for using this strategy is to increase the diversity of the solutions stored in the memory.

Heuristic HH:

1. Set $k \leftarrow 0$, $P_k \leftarrow \emptyset$, and $S_k \leftarrow \{\text{jobs with no predecessors}\}$. Initialize the amount of resources available at any time instant $u = 1, \dots, T$ with $labor[u] = L$.
2. For all $j \in S_k$ do:
 - $\sigma_j = \max\{e_j, \min\{t : \ell_{j,u-t+1} \leq labor[u] \forall u = t, \dots, t + p_j - 1\}\}$
3. Compute $\sigma^* = \min\{\sigma_j : j \in S_k\}$ and $\phi^* = \min\{\phi_j : j \in S_k\}$
4. Choose $j' \in S_k$ such that $\sigma_{j'} \leq \sigma^* + \delta(\phi^* - \sigma^*)$. Ties are broken by one of the priority rules of Subsection 10.2.1. If ties persist, they are broken by a random choice.
5. Build a partial schedule P_{k+1} by adding j' to P_k with starting time $\sigma_{j'}$.
6. (a) Set $labor[u] \leftarrow labor[u] - \ell_{j',u-\sigma_{j'}} \forall u = \sigma_{j'}, \dots, \sigma_{j'} + p_{j'} - 1$.
- (b) Set $e_j \leftarrow \max\{e_j, \sigma_{j'} + p_{j'}\}$ for every successor j' of job j .
- (c) $k \leftarrow k + 1$
- (d) Set $S_k \leftarrow \{\text{all jobs not yet scheduled whose predecessors are in } P_k\}$.
7. If $k < n$ go back to step 2. Otherwise, stop.

The three heuristics run quite fast in time complexity $O(n^2 + T)$, assuming that the largest job duration is bounded by a constant. Nine different values for $\delta = 0.1, 0.2, \dots, 0.9$ are used for heuristic HH and the best solution among all runs is retained. Preliminary tests have indicated that heuristic HH leads to better results when the ties are broken in step 5 using a priority rule based on the longest critical path.

Construction agents continuously ask process ATP for partial solutions. Whenever one of such requests is satisfied, the corresponding agent generates

a complete solution according to the associated heuristic. Next, it sends a message containing the cost of this complete solution to process ATC. If ATC accepts the cost of this solution, the corresponding sequence of jobs is sent by the construction agent to it.

10.2.5 Processes *IMPJ*, *IMPCP*, *IMPALL*, and *IMPSWP* – Improvement agents

The improvement agents designed for the LCSP continuously send messages to process ATC, requesting complete solutions. Whenever they receive a solution, they use it to start a local search procedure. If the local search encounters a better solution, its cost is sent to process ATC process. As before, the latter checks whether the acceptance criterion is fulfilled, in which case the new complete solution is sent to it.

Four improvement agents have been implemented. They differ by the neighborhoods they use for local search. Neighborhoods are characterized by two basic move operations. Given a job sequence s and two jobs i and j , movement *Insert*(i, j) inserts job j immediately in front of job i in the sequence. Movement *Swap*(i, j) interchanges the positions of jobs i and j in the sequence. Only movements leading to feasible schedules are permitted, i.e. a movement is performed solely if the precedence constraints are obeyed.

To detect moves violating precedence constraints, we proceed as follows. First, for each job j we pre-compute the set PPL_j of jobs which are neither successors nor predecessors of j . Given a feasible sequence s , the forbidden movements are: (i) *Insert*(i, j) for all pairs $i-j$ such that j is a successor of i not belonging to PPL_i , and (ii) *Swap*(i, j) for all pairs $i-j$ such that j does not belong to PPL_i . However, avoiding the two types of movements in (i) and (ii) is not enough to guarantee that the new sequence corresponds indeed to a feasible solution. This is because we still depend on the positions of the predecessors and successors of jobs i and j , which can only be computed in running time, when the actual sequence s is known. Straightforward routines running in time $O(n)$ were implemented to check the feasibility of *Insert* and *Swap* movements.

Thus, given a job sequence s , the following neighborhood variants are used in each of the four improvement agents of the a-team:

- *IMPJ*: solutions obtained by applying all possible insertion movements *Insert*(i, j) to a randomly chosen job j ($O(n)$ movements);
- *IMPCP*: solutions obtained by applying the insertion movement *Insert*(i, j) to every pair $i-j$ of jobs such that one of them is in the critical path of the precedence graph ($O(n^2)$ movements);
- *IMPALL*: solutions obtained by applying the insertion movement *Insert*(i, j) to every pair $i-j$ of jobs ($O(n^2)$ movements); and
- *IMPSWP*: solutions obtained by applying all possible swap movements *Swap*(i, j) to a randomly chosen job j ($O(n)$ movements).

The improvement agents are based on the computation of best-improving moves for each local search iteration. As explained before, the makespan computation for each solution can be done in time $O(n)$ and, therefore, the cost of exploring the neighborhood is $O(n^2)$ (resp. $O(n^3)$) for *IMPJ* and *IMPSWP* (resp. *IMPCP* and *IMPALL*).

A detailed schema of the a-team designed for the LCSP is given in Figure 10.6. Computational results will be reported in Section 10.4.

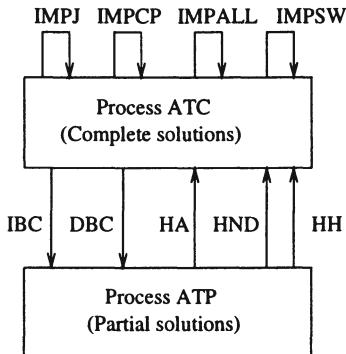


Figure 10.6 A-team for LCSP (detailed).

10.3 PARALLEL TABU SEARCH FOR LCSP

Tabu search is an adaptive metaheuristic for solving combinatorial optimization problems, which guides local search methods to continue the search beyond local optimality [16, 17, 18]. It makes use of flexible adaptive memory to modify the neighborhood of the current solution as the search progresses. Moves toward solutions deteriorating the current one are accepted under certain circumstances. Parallel tabu search implementations described in the literature are a promising alternative to further improve solution quality and algorithm robustness. Successful applications and implementations of tabu search are described e.g. in [1, 9, 10, 12, 13, 14, 24, 26, 34, 35].

Cung et al. describe and compare strategies for the parallel implementation of metaheuristics in [14], providing a classification scheme based on that originally proposed in [38] for the parallelization of local search heuristics. Strategies for the parallelization of tabu search range from the speedup of some phases of the algorithm to its complete redesign, in which several searches are performed in parallel and cooperate along their execution. In the case of *single-walk* strategies, a unique search path is explored. Either the neighborhoods or the problem domain are distributed among the processors. In the case of strategies based on neighborhood partition, each processor explores part of the neighborhood and the best move overall is retained. Within the class of *multiple-walk* strategies, each processor performs a potentially different sequential algorithm, using different parameters or starting from different initial solutions. We make the

distinction between algorithms that perform independent searches and those that cooperate. In the case of *independent* strategies, processes do not communicate and the best solution is collected among those found by each processor. Process communicate within *cooperative* strategies by exchanging high quality solutions and other relevant information gathered along their trajectories, so as to drive the search towards more promising regions of the solution space. Single- and multiple-walk strategies can be implemented as *synchronous* or *asynchronous* algorithms.

The parallel tabu search procedure described in Subsection 10.3.2 follows an asynchronous cooperative multiple-walk strategy. Processors performing different sequential tabu search algorithms exchange information in an asynchronous manner, through a central memory which is handled by a central processor. Therefore, this algorithm acts very much like as an a-team. Indeed, it can be viewed as an a-team whose agents are sequential tabu search algorithms. The TSLCSP sequential tabu search algorithm for LCSP on which this team is based was introduced and described in details in [6]. Its main characteristics are summarized below.

10.3.1 Sequential tabu search

Solution representation and neighborhood definitions are the same as those described in Section 10.2. In broad terms, our sequential short-term-memory-based tabu search algorithm TSLCSP can be described as follows. Starting from an initial schedule, it proceeds by choosing at each iteration the best admissible movement (i.e., either an insertion or a swap movement which is not prohibited or satisfies the aspiration criterion) from a set of candidate ones. Only movements leading to feasible schedules are considered. At the end of each iteration, the current solution is replaced by the new schedule obtained by local search. Algorithm TSLCSP returns the best schedule found over all iterations. We now briefly discuss some components of this algorithm.

Initial solution. Initial solutions are generated by three different approaches:

- *Best_PRH*: the best solution generated by the greedy heuristics with the priority rules defined in Section 10.2.1;
- *Best_SSH*: the best solution generated by the heuristics based on classes of schedules discussed in Section 10.2.4; and
- *Sol_PLH*: a solution obtained from the optimal solution of the linear program relaxation of LCSP (see [6] for details).

Objective function. Two alternative objective functions are considered. The first one certainly is the makespan, denoted here by F_1 . The second function is denoted by F_2 and is defined as follows. For each job j in a schedule s , let σ_j be its starting time, p_j its duration, and α_j the size of the critical path induced

in the precedence graph by job j and its successors. Thus,

$$F_2(s) = \sum_{j=1}^n \alpha_j \sigma_j + (F_1(s) + 1) \sum_{j \in n} T \alpha_j,$$

where T is the horizon under consideration and $F_1(s) = \max\{\sigma_j + p_j : j = 1, \dots, n\}$ is the makespan.

The objective function F_2 was first introduced in [6]. A simple proof by contradiction shows that an optimal schedule with respect to the above objective function is also optimal with respect to the makespan (see [8] for details). We remind that, in any case, the corresponding schedule associated with a feasible sequence is computed by taking the least cost schedule between those obtained from the direct and the inverse instances.

Aspiration criterion. A tabu restriction can be dropped whenever the corresponding move leads to a schedule better than the currently best.

Stopping criteria. The tabu search algorithm stops whenever one of these two conditions is detected: either a maximum number `MAX_TOTAL_ITER` of iterations is attained or a maximum number `MAX_BAD_MOVES` of iterations are performed without improvement in the best solution found.

Post-optimization. We try to improve on the best solution by applying a deterministic local search based on insertion movements. For each job j , all insertion movements $Insert(i, j)$ are examined, where i is a direct successor of j or $i \in PPL_j$ (so as to avoid the generation of neighbors which violate the precedence constraints).

Neighborhood. The different movement definitions considered here lead to different neighborhood definitions. To speedup the neighborhood search, our strategy only considers a candidate set formed by movements leading to a restricted subset of the full neighborhood. The search does not consider all possible pairs of jobs $i-j$ for insertions and swaps, but instead just a restricted subset of pairs. Notice that insertion and swap movements are the same as defined in Section 10.2 for the improvement agents of the a-team but, for the reasons given above, they are used to define smaller neighborhoods.

Different neighborhood definitions, together with different tabu restrictions and tabu tenures, were considered and tested in [6]. Among all configurations compared in that study, eight configuration have been retained, as those which empirically have been shown to lead to the best solutions. Those configurations are briefly summarized below. As before, s is defined as the current sequence of jobs representing a feasible solution.

- ***TSLCSP1***: We use a restricted neighborhood in which we explore the ten first feasible solutions obtained from s by applying insertion movements $Insert(i, j)$ to pairs of randomly chosen jobs $i-j$. The tabu restriction imposes that job j cannot be moved for a certain number (tabu tenure) of iterations, corresponding to an integer value randomly chosen in the interval $[[0.5\sqrt{n}], [0.8\sqrt{n}]]$.

- *TSLCSP2*: The neighborhood definition is the same as for *TSLCSP1*. However, in this case the tabu restriction prohibits job j to be inserted immediately before job i in the sequence and a dynamic tabu tenure randomly chosen in the interval $[1.2\sqrt{n}], [1.5\sqrt{n}]]$ is used.
- *TSLCSP3*: In this case, we explore a restricted neighborhood in which the ten first feasible solutions obtained from s by applying swap movements to pairs of randomly chosen jobs $i-j$ are investigated. Both the tabu tenure and the tabu restriction are the same adopted for *TSLCSP1*, but the latter is extended also to encompass the prohibition of movements involving job i . Movements involving anyone of jobs i and j are forbidden.
- *TSLCSP4*: The neighborhood definition is the same as for *TSLCSP3*. The tabu restriction prevents jobs $i-j$ from being swapped for a certain number of iterations randomly chosen in the interval $[0.9\sqrt{n}], [1.1\sqrt{n}]]$.
- *TSLCSP5*: The restricted neighborhood encompasses all feasible solutions which can be obtained from the current solution s by applying insertion movements *Insert*(i, j) to all pairs of jobs involving some specific randomly chosen job j . The tabu restrictions and the tabu tenures are the same as for *TSLCSP1*.
- *TSLCSP6*: The restricted neighborhood is the same as for *TSLCSP5*, while the tabu restrictions and the tabu tenures are the same as for *TSLCSP2*.
- *TSLCSP7*: In this case, we consider a restricted neighborhood in which we explore all feasible solutions obtained from s by applying insertion moves *Insert*(i, j) to all pairs of jobs involving some specific randomly chosen job i . The tabu restrictions and the tabu tenures are the same adopted for *TSLCSP1*.
- *TSLCSP8*: Finally, we use the same neighborhood as for *TSLCSP7*, while the tabu restrictions and the tabu tenures are the same as for *TSLCSP2*.

Each of these eight configurations was tested in [8] for the six possible combinations of initial solution (*Best_PRH*, *Best_SSH*, and *Best_PLH*) and objective function (F_1 and F_2). For each pair of initial solution and objective function, five runs of the eight configurations were made for all instances of the benchmark data set. None of them have dominated the others in terms of solution quality. The best solution found among those runs was kept and used as one of the possible starting points in the parallel version of tabu search algorithm described in the next section.

10.3.2 The tabu team

The structure of the search processes follows the same idea used in [1] to build a parallel tabu search for the circuit partitioning problem. The eight configurations for the TSLCSP algorithm described in Section 10.3.1 were selected to

act as agents of the parallel tabu search scheme since, as reported in [8], they have produced the best computational results for the sequential tabu search algorithm.

The framework of the parallel tabu search algorithm proposed for problem LCSP is depicted in Figure 10.7. Eight search processes corresponding to each of the eight configurations of the basic sequential tabu search cooperate through a central process, which maintains a pool containing the best solutions found by the search processes. The search processes start from different initial solutions, as described in Section 10.4, and use different search strategies to explore the solution space. Whenever a search process improves the best solution it has already visited, it sends the new solution to the central process. After a certain number of iterations without improving its best solution, a search process requests a new solution from the pool of elite solutions kept by the central process. Thus, cooperation among the search processes is done through the central process and consists in the reuse of solutions obtained by other search processes. Globally, this mechanism can be viewed as an intensification scheme, since it forces the exploration of the neighborhood of a solution which has been visited earlier. However, from the point of view of the algorithm running on the processor which receives a solution, it resembles to a diversification step, since the search algorithm jumps from its current solution to a new one which, in general, does not belong to the current neighborhood. Each search process decides to send solutions to or to request solutions from the central process based exclusively on its own search path. Thus, communication between search and central processes is clearly asynchronous. The operation of the search and central processes is further detailed below.

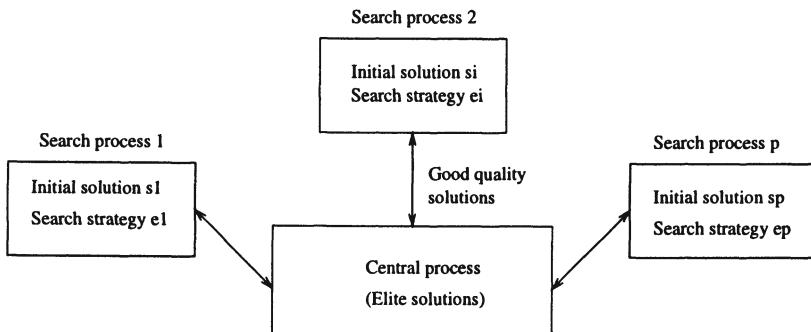


Figure 10.7 Parallel tabu search for problem LCSP.

Central process. Similarly to the a-team case, the central process has the following basic functions: (i) to start up the search processes; (ii) to manage the pool of elite solutions, and (iii) to determine the end of the search. Most of these tasks are executed precisely in the same way as those performed by the a-team described in Section 10.2. The major difference consists in the form by which solutions are stored in memory and exchanged between processes.

The data stored for each solution in the pool are: its cost, the job sequence, and a list of forbidden processes to which this solution cannot be sent, since they have already visited it. The size of the pool is determined by a parameter `POOL_SIZE`. The pool is initially empty and solutions are stored in increasing order of cost, without repetitions. Whenever a new solution is accepted for insertion in the pool and the latter is full, the currently worst solution is discarded and replaced by the new one. The list of forbidden processes associated with some solution in the pool is initialized with the process that sent it when it entered into the pool. The central process updates this list in two situations: (i) whenever a solution is sent to another process by request to start an intensification phase, and (ii) whenever another process finds the same solution and tries to insert it into the pool.

After all search processes halt, the central process applies a simple local search procedure based on insertion moves to all solutions currently in the pool. This post-optimization step essentially checks for the possibility of improving the solutions in the pool by testing all possible insertion movements, as previously described for the sequential tabu search algorithm.

Search processes. There are three situations in which a search process has to communicate with the central one: (i) when it sends a good quality solution, (ii) when it requests a new solution to start an intensification phase, or (iii) when it halts its execution. A search process sends a new solution to the central process whenever this new solution improves the best solution it previously visited. As for the a-team in the previous section, the search process first sends only the cost of the solution to the central process. The latter, according to the acceptance rule, decides whether or not this solution should be stored in the pool. If so, the search process is requested to send the job sequence corresponding to the new solution.

To start an intensification phase, a search process periodically requests a new solution to the central process to restart its search. There are two situations leading to such requests: (i) a certain upper bound `PAR_MAX_BAD_MOVES` on the number of iterations without improvement in the best solution visited by this process is reached, and (ii) a certain number `MAX_NOT_INTENSIFIED_ITER` of iterations were performed since the last time a request for a new solution coming from this process failed to be served. Requests for new solutions are made and served according to the following procedure. First, the search process sends a message to the central process asking for the solution. The central process checks in the pool for the minimum cost solution whose list of forbidden processes do not contain the one which originated the request. If such a solution is found, it is sent to the search process and its list of forbidden processes is updated. The search process sets this solution as its current one, updates the current best solution if necessary, reinitializes its search memory, and the search continues. On the other hand, if the central process pool was not able to serve the request with a solution from the pool, the search process continues its search from point where it was interrupted and runs until it requests a new solution to the pool or one of the termination criteria is reached.

Finally, there are two conditions which determine the end of the computation of a given search process: (i) a certain upper bound `PAR_MAX_TOTAL_ITER` on the total number of iterations executed by this process is attained, or (ii) two consecutive requests for new solutions have not been served by the central process and the search process was unable to improve its best solution in between.

10.4 COMPUTATIONAL EXPERIMENTS

We have implemented both the a-team and the parallel tabu search for LCSP using the C++ language. Communication between the processes is performed using the *Parallel Virtual Machine* (PVM) library, version 3.4. The computational results presented here were obtained on a workstation Sun SPARC 1000, with 308 MB of memory and 8 processors.

A-team and parallel tabu search codes have been tested on all the 25 instances from the benchmark data set (descriptions and data available at <http://www.dcc.unicamp.br/cid/SPLC/welcome.html>) described in [4]. Two of these instances were provided by BASF (*Ins_4o_24j_A* and *Ins_10o_88j_A*), while the remaining ones were randomly generated. Instance names are denoted by strings of the form *Ins_X o.Yj_Z*, where *Y* is the number of jobs, *X* is the number of orders, and *Z* is an optional character to distinguish instances with the same number of orders or jobs. The instances solved correspond to scenarios with 18 workers available at each period.

Several parameter settings have been tested in preliminary experiments to tune the codes. Those leading to the best results were kept for the final experiments. These tests were run on the five smaller instances whose optima are known and on the largest 88-job instance provided by BASF. We give the final values assigned to the parameters at the end of the experiments. The a-team code uses $\gamma = 10\%$ (acceptance criterion in the complete memory solution), `CSM_SIZE` = 100 (size of CSM), and `MAX_SOLUTIONS` = 15000 (termination criterion defined by the number of solutions which entered CSM). The parallel tabu search implementation also uses $\gamma = 10\%$ (acceptance criterion for a solution to enter into the pool), `MAX_NOT_INTENSIFIED_ITER` = 100 (maximum number of iterations between two requests of new solutions to the pool), `PAR_MAX_BAD_MOVES` = 100 (maximum number of non-improving moves before the request of a new solution to the pool), and `PAR_MAX_TOTAL_ITER` = 1000 (maximum number of iterations performed by each search process).

Each search process uses one of the neighborhood configurations *TSLCSP1* to *TSLCSP8* described in Section 10.3.1. To improve solution quality and to reduce the influence of the initial solutions, each run of the parallel tabu search algorithm is performed in two passes, using two different initialization strategies. In the first pass, a *random* strategy initializes each process with a pair initial solution - objective function randomly chosen from the set $\{Best_PRH, Best_SSH, Sol_PLH\} \times \{F_1, F_2\}$. Next, in the second pass, a *best-choice* strategy initializes each process with the pair initial solution - objective function set as the one which produced the best solution among the six combinations for

the corresponding neighborhood configuration. Whenever they are processed independently, none of these strategies seems to clearly dominate the other. Table 10.1 summarizes comparative results obtained for the set of benchmark instances. For each instance, we indicate the number of jobs and the cost of the best solutions independently found by the random and the best-choice strategies. Numbers in bold face represent best values. The two strategies found equivalent solutions for 12 out of the 25 instances. The best-choice strategy produced the best solution for nine of the remaining instances, while the random strategy was better for only four instances.

Instance	Jobs	Random	Best-choice
<i>Ins_4o_21j_A</i>	21	82	82
<i>Ins_4o_23j_A</i>	23	58	58
<i>Ins_4o_24j_A</i>	24	68	68
<i>Ins_4o_24j_B</i>	24	72	72
<i>Ins_4o_27j_A</i>	27	67	67
<i>Ins_6o_41j_A</i>	41	141	140
<i>Ins_6o_41j_B</i>	41	110	110
<i>Ins_6o_41j_C</i>	41	126	128
<i>Ins_6o_44j_A</i>	44	117	117
<i>Ins_6o_44j_B</i>	44	137	137
<i>Ins_8o_63j_A</i>	63	260	259
<i>Ins_8o_63j_B</i>	63	316	314
<i>Ins_8o_63j_C</i>	63	297	294
<i>Ins_8o_65j_A</i>	65	406	406
<i>Ins_8o_65j_B</i>	65	384	383
<i>Ins_10o_84j_A</i>	84	635	634
<i>Ins_10o_84j_B</i>	84	554	550
<i>Ins_10o_85j_A</i>	85	783	791
<i>Ins_10o_87j_A</i>	87	582	581
<i>Ins_10o_88j_A</i>	88	450	450
<i>Ins_10o_100j_A</i>	100	1468	1468
<i>Ins_10o_102j_A</i>	102	1158	1155
<i>Ins_10o_106j_A</i>	106	1087	1087
<i>Ins_12o_108j_A</i>	108	1271	1275
<i>Ins_12o_109j_A</i>	109	1324	1328

Table 10.1 Best makespans obtained with random and best-choice initialization strategies.

Table 10.2 shows, for each test instance, the number of jobs, and the makespan of the best schedules found by the a-team and the two-pass parallel tabu search algorithm, together with those obtained by the other three cited sequential approaches used to solve this problem: constraint programming (CP) [4], linear programming-based heuristics (LPH), and sequential tabu search (STS) [6]. To allow a better evaluation of the efficiency gain resulting from the parallel approaches, we have also included the best results (BCA) obtained among those generated individually by the construction agents of the a-team.

These heuristics have been tested in [8], from where the results in Table 10.2 are taken, and show that they are outperformed by all other methods. For each instance, the result displayed for the sequential tabu search algorithm corresponds to the best value found after three runs (each using a different initial seed for the random number generator) with the eight tabu processes running independently and performing $8 \times \text{MAX_TOTAL_ITER} = 8000$ iterations each. For each instance, the value of the best solution found is displayed in bold face in Table 10.2. All heuristics but the construction agents of the a-team found solutions with the same makespan for the first five instances. These values have been proved to be optimal in [6]. The construction agents of the a-team found optimal solutions only for the first two instances.

The two parallel approaches presented in this paper never failed in finding a better solution than the best one found by the sequential algorithms. Solutions generated by the a-team are better than or match the best one found by the sequential methods for 17 out of the 25 instances. Moreover, for nine instances the results produced by the a-team are strictly better. This comparison is still more striking in the case of parallel tabu search, which produced results at least as good as those found for the sequential methods for all 25 instances. For 15 of such instances, parallel tabu search produced solutions strictly better than all sequential techniques applied to this problem.

For 13 out of the 25 instances, parallel tabu search found strictly better solutions than the a-team, while the a-team was better in the case of only four instances. Moreover, parallel tabu search seems to be much more effective exactly for the larger, more difficult instances. Indeed, the parallel tabu search algorithm found strictly better solutions than all other algorithms (including the a-team) for nine among the ten largest instances. These results justify the use of the parallel approaches to tackle the LCSP. Moreover, they show that the cooperation between the different algorithms is crucial to obtain high quality solutions. As it can be seen by comparing the results obtained by the sequential and parallel tabu search algorithms, in all but one case both parallel tabu search strategies have been able to produce solutions of better quality than those obtained by each tabu search agent running independently. The only exception refers to instance *Ins_8o_63j_C* when the random strategy is used to generate initial solutions.

The results reported in Tables 10.1 and 10.2 for the parallel algorithms have been obtained without too much effort in tuning the best parameter values. Since the parallel algorithms have some randomized steps, we have investigated their robustness running both the a-team and the parallel tabu search three times for each instance, using different initial seeds. For both the a-team and the parallel tabu search algorithm, the relative error with respect to the best value observed for the worst and for the median values found never exceeded 1.5%, showing that randomness in the parallel algorithms seems to have a small influence in their performances. This behavior has been already observed in [6] for the sequential tabu search algorithm.

Instance	Jobs	AT	PTS	STS	BCA	CP	LPH
<i>Ins_4o_21j_A</i>	21	82	82	82	82	82	82
<i>Ins_4o_23j_A</i>	23	58	58	58	58	58	58
<i>Ins_4o_24j_A</i>	24	68	68	68	69	68	68
<i>Ins_4o_24j_B</i>	24	72	72	72	73	72	72
<i>Ins_4o_27j_A</i>	27	67	67	67	69	67	67
<i>Ins_6o_41j_A</i>	41	143	140	141	150	152	142
<i>Ins_6o_41j_B</i>	41	111	110	110	115	110	112
<i>Ins_6o_41j_C</i>	41	126	126	128	136	134	130
<i>Ins_6o_44j_A</i>	44	116	117	117	123	122	118
<i>Ins_6o_44j_B</i>	44	137	137	137	146	149	137
<i>Ins_8o_63j_A</i>	63	259	259	261	283	281	273
<i>Ins_8o_63j_B</i>	63	316	314	316	347	344	323
<i>Ins_8o_63j_C</i>	63	301	294	296	325	344	308
<i>Ins_8o_65j_A</i>	65	403	406	406	421	445	411
<i>Ins_8o_65j_B</i>	65	382	383	384	408	411	402
<i>Ins_10o_84j_A</i>	84	641	634	636	708	730	—
<i>Ins_10o_84j_B</i>	84	567	550	556	606	616	—
<i>Ins_10o_85j_A</i>	85	793	783	791	879	912	—
<i>Ins_10o_87j_A</i>	87	585	581	582	615	610	—
<i>Ins_10o_88j_A</i>	88	456	450	460	693	473	—
<i>Ins_10o_100j_A</i>	100	1467	1468	1468	1519	1587	—
<i>Ins_10o_102j_A</i>	102	1158	1155	1166	1266	1239	—
<i>Ins_10o_106j_A</i>	106	1098	1087	1094	1163	1166	—
<i>Ins_12o_108j_A</i>	108	1277	1271	1277	1341	1412	—
<i>Ins_12o_109j_A</i>	109	1336	1324	1343	1461	1476	—

Table 10.2 Best values found for benchmark instances of LCSP.

For all but the two smaller instances, we illustrate in Figure 10.8 the observed relative elapsed times to the best solution for the a-team and the parallel tabu search algorithm (the latter running with only one initialization pass and using the best-choice strategy). Elapsed times have been measured in exclusive mode, with each parallel algorithm running as a single user. We notice that the elapsed times to the best solution seem to be considerably smaller for the parallel tabu search algorithm (less than one third of the time taken by the a-team for most cases). Parallel tabu search finds better solution within the same elapsed times and are more likely to benefit from improved stopping criteria leading to earlier termination of either algorithm.

10.5 CONCLUSIONS

We proposed two parallel and cooperative algorithms for the labor constrained scheduling problem. The first is an a-team scheme, while the second is an asynchronous cooperative multiple-walk parallel tabu search strategy. The computational results obtained for a set of LCSP benchmark instances have

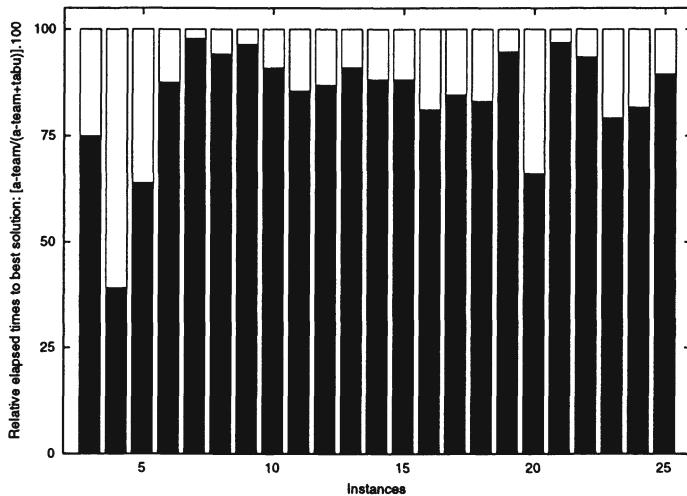


Figure 10.8 Relative elapsed times to the best solution for the parallel approaches.

shown that both parallel algorithms proposed are able to find high quality approximate solutions. In particular, they have matched or improved the best known solutions for all benchmark instances. Moreover, the computational results have also shown that both strategies benefit directly from the synergy induced by running several algorithms in parallel, since they have found better solutions than any individual sequential algorithm running in the same overall computation time.

A direct comparison of the two parallel approaches favors the tabu search algorithm. For the majority of the instances tested in our experiments, the parallel tabu search code generates solutions with smaller makespans and faster than the a-team code. A possible explanation for this performance comes from the fact that the agents in the tabu search code are more complex than those forming the a-team. In spite of this, it is somehow intriguing that the a-team was able to produce strictly better solutions than the tabu code in four out of the 25 instances of the benchmark data set (see Table 10.2).

Acknowledgments: The research of C.C.B. Cavalcante was supported by FAPESP under grant 96/10270-8. The work of C.C. Ribeiro was supported by CNPq (grants 302281/85-1 and 202005/89-5) and FAPERJ (grant 150966/99). The work of C.C. de Souza was supported by CNPq (grant 300883/94-3) and FINEP (ProNEx-107/97).

References

- [1] R.M. Aiex, S.L. Martins, C.C. Ribeiro, and N.R. Rodriguez. Cooperative Multi-Thread Paralell Tabu Search with an Application to Circuit Partitioning. *Lecture Notes in Computer Science*, 1457:310–331, 1998.

- [2] L. Baerentzen, P. Avila, and S. Talukdar. Learning Network Design for Asynchronous Teams. *Lecture Notes in Artificial Intelligence*, 1237:177–196, 1997.
- [3] K.R. Baker. *Introduction to Sequencing and Scheduling*. Wiley, 1974.
- [4] C.B. Cavalcante, Y. Colombani, S. Heipcke, and C.C. Souza. Scheduling under Labour Resource Constraints. *Constraints*, 5:415–422, 2000.
- [5] C.B. Cavalcante and C.C. Souza. A Tabu Search Approach for Scheduling Problem under Labour Constraints. Technical Report, State University of Campinas, Institute of Computing, IC-97-13, 1997.
- [6] C.B. Cavalcante, C.C. Souza, M.W. Savelsbergh, Y. Wang, and L.A. Wolsey. Scheduling Projects with Labor Constraints. To appear in: *Discrete Applied Mathematics*.
- [7] V.F. Cavalcante. *Asynchronous Teams for the Job Shop Scheduling Problem: Construction Heuristic* (in Portuguese). M.Sc. Dissertation, State University of Campinas, Institute of Computing, 1995.
- [8] C.C.B. Cavalcante. *Scheduling Under Labour Constraints: Heuristics and Lower Bounds* (in Portuguese). M.Sc. Dissertation, State University of Campinas, Institute of Computing, 1998.
- [9] J. Chakaprani and J. Skorin-Kapov. Massively Parallel Tabu Search for the Quadratic Assignment Problem. *Annals of Operations Research*, 41:327–341, 1993.
- [10] J. Chakaprani and J. Skorin-Kapov. Connection Machine Implementation of a Tabu Search for the Traveling Salesman Problem. *Journal of Computing and Information Technology*, 1:29–36, 1993.
- [11] P. Chang, J. Dolan, J. Hemmerle, S. Talukdar, and M. Terk. Asynchronous Teams: An Agent-Based Problem-Solving Architecture. *Artificial Neural Networks in Engineering*, 7:73–78, 1997.
- [12] T.G. Crainic, M. Toulouse, and M. Gendreau. Parallel Asynchronous Tabu Search for Multicommodity Location-Allocation with Balancing Requirements. *Annals of Operations Research*, 63:277–299, 1993.
- [13] T.G. Crainic, M. Toulouse, and M. Gendreau. Synchronous Tabu Search Parallelization Strategies for Multicommodity Location-Allocation with Balancing Requirements. *OR Spektrum*, 17:113–123, 1995.
- [14] V.-D. Cung, S.L. Martins, C.C. Ribeiro, and C. Roucairol. Strategies for the Parallel Implementation of Metaheuristics. In: *Essays and Surveys in Metaheuristics*, C.C. Ribeiro and P. Hansen, editors, Kluwer, 2001 (this volume).

- [15] E.W. Davis and J.H. Patterson. A Comparison of Heuristics and Optimum Solutions in Resource Constrained Project Scheduling. *Management Science*, 21:944–955, 1975.
- [16] F. Glover. Tabu Search – Part I. *ORSA Journal on Computing*, 1:190–206, 1989.
- [17] F. Glover. Tabu Search – Part II. *ORSA Journal on Computing*, 2:4–32, 1990.
- [18] F. Glover and M. Laguna. *Tabu Search*. Kluwer, 1997.
- [19] E.G. Haddad. *Asynchronous Teams for the Job Shop Scheduling Problem: Improvement Heuristics* (in Portuguese). M.Sc. Dissertation, State University of Campinas, Institute of Computing, 1996.
- [20] S. Heipcke. *A New Constraint Programming Approach to Large Scale Resource Constrained Scheduling*. Diploma-Thesis, Mathematisch Geographische Fakultät, Katholische Universität Eichstätt, 1995.
- [21] S. Heipcke and Y. Colombani. A New Constraint Programming Approach to Large Scale Resource Constrained Scheduling. Presented at the Workshop on Models and Algorithms for Planning and Scheduling Problems, Cambridge, 1997.
- [22] J.H. Patterson. Project Scheduling: The Effects of Problem Structure on Heuristic Performance. *Naval Research Logistics Quaterly*, 20:95–123, 1976.
- [23] H.P. Peixoto. *A Methodology for the Specification of Asynchronous Teams for Combinatorial Optimization Problems* (in Portuguese). M.Sc. Dissertation, State University of Campinas, Institute of Computing, 1995.
- [24] S.C. Porto and C.C. Ribeiro. Parallel Tabu Search Message-Passing Synchronous Strategies for Task Scheduling under Precedence Constraints. *Journal of Heuristics*, 1:107–223, 1996.
- [25] J. Rachlin, R. Goodwin, S. Murthy, R. Akkiraju, F. Wu, S. Kumaran, and R. Das. A-Teams: An Agent Architecture for Optimization and Decision-Support. *Lecture Notes in Artificial Intelligence*, 1555:261–276, 1999.
- [26] C. Rego and C. Roucairol. A Parallel Tabu Search Algorithm using Ejection Chains for the VRP. In: *Metaheuristics: Theory and Applications*, I.H. Osman and J.P. Kelly, editors, pages 253–295, Kluwer, 1996.
- [27] Y. Rochat and É.D. Taillard. Probabilistic Diversification and Intensification in Local Search for Vehicle Routing. *Journal of Heuristics*, 1:147–167, 1995.

- [28] M.W. Savelsbergh, R.N. Uma, and J. Wein. An Experimental Study of LP-based Approximation Algorithms for Scheduling Problems. In: *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 453–462, 1998.
- [29] M.W. Savelsbergh, Y. Wang, and L.A. Wolsey. Computational Experiments with a Large-Scale Resource Constrained Project Scheduling Problem. Note, Georgia Institute of Technology, 1996.
- [30] C.C. Souza and L.A. Wolsey. Scheduling Projects with Labour Constraints. Technical Report, State University of Campinas, Institute of Computing, IC-97-13, 1997.
- [31] P.S. Souza. *Asynchronous Organizations for Multi-Algorithm Problems*. Ph.D. Dissertation, Electrical and Computer Engineering Department, Carnegie Mellon University, 1993.
- [32] P.S. Souza and S.N. Talukdar. Genetic Algorithms in Asynchronous Teams. In: *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 392–397, Morgan Kaufmann, 1991
- [33] R.H. Storer and S.D. Wu, and R. Vaccari. New Search Spaces for Sequence Problems with Applications to Job Shop Scheduling. *Management Science*, 38:1495–1509, 1992.
- [34] E.D. Taillard. Robust Taboo Search Techniques for the Quadratic Assignment Problem. *Parallel Computing*, 17:443–455, 1991.
- [35] E.D. Taillard. Parallel Taboo Search Techniques for the Job Shop Scheduling Problem. *ORSA Journal on Computing*, 6:108–117, 1994.
- [36] S.N. Talukdar, L. Baerentzen, and A. Gove. Asynchronous Teams: Cooperation Schemes for Autonomous Agents. *Journal of Heuristics*, 4:295–321, 1998.
- [37] S.N. Talukdar, S.S. Pyo, and T. Giras. Asynchronous Procedures for Parallel Processing. *IEEE Transactions on Power Apparatus and Systems*, PAS-102:3652–3659, 1983.
- [38] M.G.A. Verhoeven and E.H.L. Aarts. Parallel Local Search. *Journal of Heuristics*, 1:43–65, 1995.

11 A SCATTER SEARCH ALGORITHM FOR THE MAXIMUM CLIQUE PROBLEM

Luís Cavique¹, César Rego², and Isabel Themido³

¹Escola Superior de Comunicação Social
Instituto Politécnico de Lisboa, Portugal
lcavique@escs.ipl.pt

²Hearin Center for Enterprise Science
School of Business Administration, University of Mississippi
University, MS 38677, USA
crego@bus.olemiss.edu

³CESUR, Instituto Superior Técnico
Universidade Técnica de Lisboa, Portugal
ithemido@cesur.civil.ist.utl.pt

Abstract: The objective of the Maximum Clique Problem (MCP) is to find the largest complete subgraph in a given graph. The problem is known as NP-hard and we have developed a heuristic algorithm based on a Scatter Search (SS) framework to find a lower bound for this maximization problem. The proposed algorithm was developed with two search features: a guidance search and a local search feature. For the first feature a Scatter Search algorithm was chosen with the purpose of extensively exploring regions with strategically combined solutions. The new solutions obtained in the combination phase are thereafter improved by a neighborhood search procedure based on tabu search for implementing the second feature. The computational results obtained with DIMACS clique benchmark instances show that the proposed algorithm finds solutions comparable to the ones provided by some of the most competitive algorithms for the MCP.

11.1 INTRODUCTION

We consider the Maximum Clique Problem (MCP) defined as follows. Given an undirected graph $G = (V, A)$ and $A(v_i)$ denoting the set of vertices v_j such that $(v_i, v_j) \in A$, then a graph $G1 = (V1, A1)$ is called a subgraph of G if $V1 \subseteq V$, and for every $v_j \in V1$, $A1(v_i) = A(v_i) \cap V1$. A graph $G1$ is said to be complete if there is an arc for each pair of vertices. A complete subgraph is also called a clique. A clique is maximal, if it is not contained in any other clique. In the MCP the objective is to find a complete subgraph of largest cardinality in a graph. The clique number is equal to the cardinality of the largest clique of G .

The MCP is an important problem in combinatorial optimization with many applications which include: market analysis [25], project selection [6], and signal transmission [3]. The interest for this problem led to the algorithm thread challenge on experimental analysis and algorithm performance promoted by Second DIMACS Implementation Challenge [21].

The MCP is a hard combinatorial problem, classified as NP-Complete [21]. In this paper we are interested in heuristic approaches which provide good solutions to this problem, but not necessary the optimum.

It is known that the Maximum Independent Set, the Minimum Vertex Cover and the Maximum Clique are equivalent problems for which a number of heuristic algorithms have been recently proposed: Genetic Algorithms [1], Neural Networks [18, 19], Randomized Search – GRASP, and Simulated Annealing [8], and Tabu Search [2, 11, 26].

The main goal of this paper is to present an experimental study for solving the Maximum Clique Problem (MCP) using a Scatter Search framework.

Scatter Search is an evolutionary strategy introduced by Glover [12] that has proved useful for solving discrete and non-linear optimization problems. The method considers linear combinations of solution vectors with heuristic improvement and rounding process (when vector components are required to be integer). We propose a Scatter Search algorithm for the MCP where structured solution combinations weighted by a “filtering vector” play the role of linear combinations. For the heuristic improvement we consider a simple tabu search procedure based on appropriate neighborhood structures.

The remainder of this paper is organized as follows. Section 11.2 introduces the Scatter Search method. The proposed Scatter Search algorithm for the MCP is described in Section 11.3. In Section 11.4 we report the computational results and discuss the experimental analysis. Finally, we draw conclusions in Section 11.5.

11.2 SCATTER SEARCH

Scatter Search (SS) was introduced by Glover [12] in a heuristic study on integer linear programming problems which also provides a number of formative ideas of Tabu Search. Scatter Search is an evolutionary method that combines solution vectors by linear combinations to produce new ones through successive

generations. The method also includes the use of a heuristic rounding process for mapping fractional components into discrete ones, as required in pure and mixed integer problems. In more recent SS designs the rounding process is sometimes replaced with a generalized improvement method that beyond making the initial trial solution feasible also incorporates some local improvement mechanism to produce enhanced trial solutions.

Scatter Search relies heavily on strategy for deciding how to combine solutions. Since Tabu Search (TS) underlies the same philosophy, it becomes appropriate to produce combined strategies. A natural way to combine SS with TS is to use tabu search for the improvement method. Another strategy for integrating TS and SS principles consists of replacing the “vector space” with the “neighborhood space” which gives rise to a TS strategy known as Path Relinking (PR).

A basic difference between SS and PR methods is the technique used for combining solutions: SS considers linear combinations of solution vectors, while in PR solutions are combined by generating paths between solutions using some local search neighborhoods, and selecting new solutions encountered along these paths. Thus, PR can be viewed as a generalization of SS and therefore is described by the same general template [15].

The SS template can be divided into 2 phases: an initial phase and an evolutionary phase. As in the evolutionary methods, the highest evaluated solutions found by the local search phase are used to update the set of solutions, called Reference Set (*RS*). The process is repeated, using the new solutions in the next iteration until the *RS* does not converge from one iteration to the next.

Scatter Search general procedure

- i) Initial phase: a set of solutions is generated and the best solutions are chosen to become part of the reference set.
- ii) Evolutionary phase: repeat the following steps until some stopping criterion is met:
 - a) New solutions are generated using weighted structured combinations of strategically selected reference subsets.
 - b) A set of the best generated solutions will be included in the reference set.

This SS procedure is a very general process with many possible implementations. In order to specify a general framework, the SS/PR template proposed in Glover [15] is used. The template comprises 5 sub-routines, which can be sketched as follows:

- I Diversification Generation Method: starting with different seed solutions a set of trial solutions is generated.

- II Improvement Method: given a trial solution as input, a heuristic process is applied to improve the solution; if the initial trial solution is not feasible the heuristic procedure should be able to restore the solution feasibility during the improvement process.
- III Reference Set Update Method: maintains the reference set consisting of a set of the best solutions found, widespread throughout the solution space.
- IV Subset Generation Method: uses a structured process that extracts solutions from the reference set and produces a subset, that will be used in the next method.
- V Solution Combination Method: combines the subset solution into one or more trial solutions.

In the Initial Phase, a set of solutions is generated and the subset of the best solutions found initializes the Reference Set (RS). A different starting point is used for each solution produced by the Diversification Generation Method, then the Improvement Method is applied. Finally, the Reference Set Update Method is called and the replacement is committed according to the “replacement criterion”. These steps corresponding to steps I, II, and III are repeated a certain number of times: $n|RS|$, with $n > 1$.

In the Evolutionary Phase the continuous improvement of the RS is expected using structured combinations of the solution reference set. The first method used is the Subset Generation Method, which selects subsets of solutions from the RS to be combined by the Solution Combination Method that uses these subsets to produce new trial solutions. As in the initial phase, the Improvement Method and the Reference Set Update Method are used. In summary, the sub-routines IV, V, II and III are called in sequence. The process is repeated a certain number of iterations while the RS converges.

Illustrative examples of the Scatter Search template in the context of two different classical integer programming problems can be found in Laguna [22], Rego and Leão [24].

In our implementation we also introduced the possibility of using diverse solutions in the subsets, so that solutions with different characteristics can be combined. In order to create new solutions, the Diversification Generation Method is used. In this fashion all sub-routines are called in the Evolutionary Phase with diverse solutions.

11.3 SCATTER SEARCH ALGORITHM FOR THE MCP

This section will describe in detail each of the five sub-routines of our Scatter Search algorithm applied to the Maximum Clique Problem, where some special features and techniques have been introduced for the implementation to this problem.

Before describing the algorithm, we should note some of the most relevant features which motivated the implementation of the proposed Scatter Search algorithm.

By contrasting Scatter Search with other evolutionary methods and general local search algorithms we can identify a number of fundamental features that may be important to create more powerful heuristic algorithms. We will focus on features specially relevant to problems that characteristically do not provide relevant information about the variable values of the optimum solution, as typically occurring in MCPs.

Scatter Search shares with other evolutionary methods the population based approach, which is more significant for the search guiding in MCPs than using moves from a single solution to another, as considered in classical local search methods. This assumption concerning the relevance of the population based neighborhoods is based on the fact that the solution space of MCPs contains a vast number of strong local optima, and most of them with the same clique value. We define “strong” local optimum as one that shares at most a few vertices with an optimal solution. On the other hand, local optima for the MCP usually differ from each other by a large percentage of vertices. As a result, to move from one local optimum to another it is usually necessary to significantly destroy the current solution subgraph and therefore no relevant information can be inherited at subsequent iterations. This is probably the reason most metaheuristic algorithms for MCP and its equivalents heavily rely on randomization and re-starting processes. However, it seems reasonable to assume optimal solutions for MCPs are made of vertices scattered in different local optima and therefore we consider population based approaches as potentially more appropriate for these classes of problems than using local neighborhood search approaches. SS provides a useful framework to implement and achieve such a strategy. Instead of exploring solutions that are further apart, as in GA, SS explores pre-determined regions extensively at each iteration. The objective is to find optimal or near-optimal combinations for each region (see, for example Cung et al. [7]). Recently, there have been similar efforts made in techniques called perfect-offspring by Lourenço, Paixão and Portugal [23], and the optimum-child by Balas and Niehaus [1] in the GAs context. However, using exact methods for finding optimal combinations are often out of reach, thus Scatter Search typically uses an heuristic improvement method for mapping the initial generated “offspring” onto enhanced trial solutions.

We describe now the methods used in our Scatter Search algorithm as well as their relationship.

11.3.1 Diversification generation method

The aim of the diversification generation method is to create a set of solutions as scattered as possible within the solution space while also using as many variables (or solution attributes) as possible. In the MCP, all vertices in the graph G should be present in RS .

When the algorithm starts, RS is initialized with a set of diverse solutions obtained by a constructive procedure, which starting from a single vertex, adds at each step a new vertex to the current clique until a maximal clique is found. Starting from a different vertex not yet included in RS , the procedure is

repeated as many times as the cardinality of the reference set ($|RS|$ usually between 10 and 40). The clique value is used to order the solutions in RS . This method is implemented by calling the primal heuristic procedure with a single vertex as input parameter and $NMAX = 1$. This procedure will be described in next method.

11.3.2 Improvement method

Some of the most efficient heuristic algorithms for the MCP are based on tabu search. Gendreau, Soriano and Salvail [11] and Soriano and Gendreau [26] present three versions of a tabu search algorithm: the first and second versions differ in the type of tabu restrictions, using one and two types of tabu lists, respectively. The third version is based on the probabilistic variant of tabu search. Inspired by the neighborhood search strategy used by this algorithm, Battiti and Protasi [2] developed an efficient reactive tabu search algorithm. The core of these algorithms is the consideration of a neighborhood structure derived from one of the heuristic procedures proposed by Johnson [20], which we call the “primal” version, since it only works in the feasible solution space. This author also proposes a “dual” counterpart of this “primal” procedure which progresses in the infeasible region of solutions and stops when the feasibility is reached.

In the context of our Scatter Search algorithm, the improvement method has two phases: given solution that is typically infeasible, the method first undertakes to recover a feasible one; and afterward it attempts to increase the objective function value. In order to implement these two phases we consider a combined strategy using the primal and dual neighborhood of Johnson as follows.

We define $A(S)$ as the set of vertices which are adjacent to vertices of a current solution S . Let $n = |S|$ be the cardinality of a clique S and $A^k(S)$ be the subset of vertices with k arcs incident in S . $A(S)$ can be divided into subgroups $A(S) = \cup A^k(S)$, $k = 1, \dots, n$. The cardinality of the vertex set $|V|$ is equal to the sum of the adjacent vertices $A(S)$ plus the non adjacent ones $A^0(S)$, resulting in $|V| = \sum |A^k(S)| + n$, $k = 0, \dots, n$.

The subsets $A^n(S)$ and $A^{n-1}(S)$ are the most relevant for the algorithm. If $A^n(S) \neq \emptyset$, then it is possible to add one vertex to the current solution, thus increasing the clique value. If $A^{n-1}(S) \neq \emptyset$, then a vertex swap can be performed, maintaining constant the objective function value. For any other subset $A^{n-m}(S) \neq \emptyset$ it is possible to swap m solution vertices with one vertex of $A^{n-m}(S)$.

In local search algorithms, a new solution S' is chosen in the neighborhood of the current solution S . The neighborhood structures $N(S)$ are classified for the MCP according to the feasibility (primal, dual) and the type of move (add, drop, swap).

For a given solution S , we define a neighborhood as primal $Np(S)$ if it generates a feasible solution S' . Otherwise, when dealing with non complete subgraphs, the neighborhood structured is called dual $Nd(S)$.

Combining the two classifications we obtain the following five neighborhood structures:

$$Np^+(S) = \{S' : S' = S \cup \{v_i\}, v_i \in A^n(S), S \text{ and } S' \text{ are cliques}\}$$

$$Np^-(S) = \{S' : S' = S \setminus \{v_i\}, v_i \in S, S \text{ and } S' \text{ are cliques}\}$$

$$Np^0(S) = \{S' : S' = S \cup \{v_i\} \setminus \{v_k\}, v_i \in A^{n-1}(S), v_k \in S, S \text{ and } S' \text{ are cliques}\}$$

$$Nd^+(S) = \{S' : S' = S \cup \{v_i\}, v_i \in A^{n-1}(S), \text{ might be a clique, } S' \text{ is not a clique}\}$$

$$Nd^-(S) = \{S' : S' = S \setminus \{v_i\}, v_i \in S, S \text{ is not a clique, } S' \text{ might be a clique}\}.$$

According to the foregoing neighborhood definitions, we can classify the algorithms of Soriano and Gendreau [26], Battitti and Protasi [2] which combine Np^+ and Np^- neighborhoods as primal tabu search algorithms. Consequently, we classify our improvement method as a *primal-dual tabu search procedure*, described as follows.

In the primal neighborhood search, the method combines Np^+ , Np^- and Np^0 neighborhood structures. At each step one new solution S' is chosen from the neighborhood $Np^+(S)$ of the current solution S . If it is possible to increase the clique value, one vertex is added to the clique. If Np^+ is empty the method swaps vertices using the neighborhood structure Np^0 . Thus, the feasible solution space is explored by alternating between Np^+ and Np^0 neighborhoods until no more moves are possible. Otherwise, when neither feasible insertions nor swap moves are allowed, the dual heuristic procedure is called to “kick” the solution to the infeasible solution space using the neighborhood structure Nd^+ and the process goes backwards removing one vertex with Nd^- until a clique is found. The best found solution S^* is updated whenever the clique value is increased.

The primal and dual neighborhood search procedures are described in Figures 11.1 and 11.2, respectively.

We consider the following notation:

S^* – the highest cardinality maximal clique found so far,

T – the tabu list,

NI – the number of iterations from the last improvement of the objective function value,

$NMAX$ – the limit for the maximum number of iterations without increasing the largest clique size found, and

$d(v_i)$ – the degree of vertex v_i .

Procedure Primal-Tabu ($S : S$ is a Clique)Initialization: $S^* \leftarrow S$, $T = \emptyset$, $NI = 0$ While $|S| > 1$ and $NI < NMAX$

1. While $A^n(S) \setminus T \neq \emptyset$
 - a) Select $v_i = \arg \max\{|A^n(S \cup \{v_i\})| : v_i \in A^n(S) \setminus T\}$
 - b) Apply $Np^+(S)$ and set $S \leftarrow S'$
 - c) If $|S| \geq |S^*|$, set $S^* \leftarrow S$, $NI = 0$. Otherwise $NI = NI + 1$
 2. If $A^{n-1}(S) \setminus T \neq \emptyset$
 - a) Select a pair $(v_i, v_k) = \arg \max\{|A^n(S \cup \{v_i\} \setminus \{v_k\})| : v_i \in A^{n-1}(S) \setminus T, v_k \in S\}$
 - b) Apply $Np^0(S)$ and set $S \leftarrow S'$, $T \leftarrow T \cup \{v_i\}$
- Else If $|S| > 1$
- c) Select $v_i = \arg \min\{d(v_i) | v_i \in S\}$
 - d) Apply $Np^-(S)$ and set $S \leftarrow S'$, $T \leftarrow T \cup \{v_i\}$

End of the procedure: Return the clique S^*

Figure 11.1 The primal neighborhood search procedure.**Procedure Dual-Tabu ($S : S$ is a subgraph)**Initialization: $T = \emptyset$

1. If S is a clique
 - d) Select $v_i = \arg \max\{d(v_i) | v_i \in A^{n-1}(S) \setminus T\}$
 - e) Apply $Nd^+(S)$ and set $S \leftarrow S'$, $T \leftarrow T \cup \{v_i\}$
2. While S is not a clique
 - e) Select $v_i = \arg \min\{d(v_i) | v_i \in S \setminus T\}$
 - f) Apply $Nd^-(S)$ and set $S \leftarrow S'$, $T \leftarrow T \cup \{v_i\}$

End Procedure: Return a clique S

Figure 11.2 Dual neighborhood search procedure.

The alternating consideration of the primal and dual methods are joined in the Primal-Dual Tabu search procedure defining the complete improvement method used by our Scatter Search algorithm. The tabu list is set up for the different types of moves used in the primal and dual procedures. The main algorithm of the improvement method can be sketched as in Figure 11.3.

Procedure Primal-Dual Tabu ($S : S$ is a subgraph)

Initialization: $S^* \leftarrow S$, $NI = 0$

While $NI < NMAX$

1. Call *Dual-Tabu* (S)
2. Call *Primal-Tabu* (S)
3. If $|S| \geq |S^*|$ set $S^* \leftarrow S$, $NI = 0$. Otherwise, set $NI = NI + 1$

End Procedure: Return a maximal clique S^* .

Figure 11.3 Procedure Primal-Dual Tabu.

In the algorithm description we consider all variables as local to each procedure and the parameters used at each procedure call correspond to the variable returned by the procedure executed in the previous call. This improvement procedure can be easily transformed into a stand alone tabu search algorithm starting with a graph G as input parameter, returning a maximal clique S^* as the output.

The Primal-Dual Tabu search procedure is more general than the Primal Tabu method, due to the way it deals with the initial solutions and explores the solution space. The starting subgraph can be either a clique or a non-complete subgraph. Besides its generalization capability, another important feature is the primal and dual complementary approaches which also implement a strategic oscillation allowing trajectories to cross infeasible regions of solutions.

In traditional local search methods, the solutions develop until they reach a “critical level”, usually a local optimum, and then stop. More competitive heuristics seek forward and backward moves in relation to the critical level, in order to overcome some lack of potentiality of the neighborhood structure when faced with special local optimum configurations. Thus, instead of stopping at the critical level, the method changes the decision rules allowing entry into infeasible regions. The solution trajectory control brought by the different neighborhood structures creates successive constructive and destructive sequence moves. Moving closer and further from the critical level on successive passes, the objective function trajectory draws an oscillation movement pattern as shown in Glover [14].

For a detailed description and use of the neighborhood structures in a stand alone tabu search algorithm, see Cavique et al. [5].

11.3.3 Reference set update method

In the initial phase a set of $n|RS|$ solutions are generated, with $n > 1$, with the best ones becoming part of the reference set. The reference set RS is implemented as an ordered array $R[i]$ ($i = 1, \dots, n$), where $RS[1]$ is the worst solution and $RS[n]$ is the best one.

The reference set update method must be carefully set up with diverse and high quality solutions to avoid the phenomenon of *premature convergence* of RS , which occurs when all the solutions are similar. To prevent this “pitfall”, the reference set RS is divided into two groups: the set of best solutions RS^b and the set of diverse solutions RS^d , $RS = RS^b \cup RS^d$. We consider $|RS^b| = \alpha|RS|$, with $0 < \alpha < 1$, hence $|RS^d| = |RS| - |RS^b|$.

Regarding the replacement policy, two complementary strategies can be identified: the replacement of the worst solution and replacement of the best solution. The worst solution replacement is the most common policy used in population based approaches.

Adopting the “replace the worst” process RS tends to keep all the best solutions found, underestimating the diversity of the sample. The “replace the best” policy seems too elitist, and disdains the good solutions found during the search. Therefore, we adopted a variant which we call the “journal replacement” policy that replaces the worst solution with the new best solution found, reporting all the “hits” of the search. In order to adopt this replacement process for the subdivided RS , the replacement condition is the quality of the new solution that updates the RS must be better than any solution in RS^b . The best solution is $RS[n]$ and the worst solution in RS^b is $RS[n - |RS^b|]$.

The policies mentioned above can be formalized as follows:

- worst solution replacement: if $|S| > |RS[1]|$, update RS by deleting solution $RS[1]$ and inserting the current solution S into the ordered array.
- best solution replacement: if $|S| > |RS[n]|$, update RS by deleting solution $RS[n]$ and inserting the solution S in the ordered array.
- journal replacement: if $|S| > |RS[n - |RS^b||]$, update RS by deleting $RS[1]$ and insert S in the ordered array.

Thus a steady state updating method is processed, where only a few solutions are replaced at each iteration, not only permitting the maintenance of diverse solutions but also keeping the best solutions found. This replacement method produces a RS that does not consist of all the best solutions found (measured by the objective function) but only the $|RS^b|$ of the best solutions.

11.3.4 Subset generation method

This method generates the following types of solution subsets which are combined in the next method. The method generates subsets with two, three or more elements in a relatively reduced computational effort. Instead of performing the combinations $C(n, 2)$, $C(n, 3)$ and $C(n, m)$, the method adds new elements to the generated subsets as follows:

1. Generate 2-element subsets by combining all the elements two by two. By choosing a new pair (X, Y) of solutions from the reference set at each iteration, a subset with two elements is obtained: $\text{Subset-2} = \{X, Y\}$.

2. 3-element subsets are derived from the 2-element subset, adding a new solution $RS[k_1]$, where k_1 denotes the index of the best solution in RS^b not in the 2-element subset. A subset with three elements is obtained: Subset-3 = $\{X, Y, RS[k_1]\}$.
3. 4-element subsets are derived from the 3-element subset by adding a new solution $RS[k_2]$, where $RS[k_2]$ is not in the 3-element subset. A subset with four elements is obtained: Subset-4 = $\{X, Y, RS[k_1], RS[k_2]\}$.

Let Ω be the set of all subsets used at each iteration: $\Omega = \{\text{Subset-2}, \text{Subset-3}, \text{Subset-4}\}$. To eliminate repetition of the elements in the subsets, the reference set with diverse solutions RS^d is used for the two by two combinations, instead of the complete RS . The total number of different subsets will be equal to the number of subsets multiplied by the number of possible combinations: $|\Omega|C_2^{|RS^d}|$.

In this method we also include a new feature by adding a distant (or diverse) solution D maximizing the distance from the region R , defined as the union of the vertices in the solution's subset, $R = X \cup Y \cup RS[n] \cup RS[n - 1]$. In this way, a new point D “far from” the solution cluster is obtained at each iteration to maintain an appropriate diversity of solutions in the reference set.

The number of possible subsets is increased by three, replacing X and Y for D . For the subset with 2-elements, we obtain the following subsets: Subset-2A = $\{X, Y\}$, Subset-2B = $\{X, D\}$, Subset-2C = $\{D, Y\}$. The same process can be applied to Subset-3 and Subset-4. In addition, the set of all subsets Ω becomes larger than the initial one, $\Omega = \{\text{Subset-2A}, \text{Subset-2B}, \dots, \text{Subset-4B}, \text{Subset-4C}\}$.

Using this strategy we can find diverse subsets to complement the initial ones while permitting a diversification strategy that can be applied when needed. This method will be used for implementing the bypass strategy in the next method.

11.3.5 Solution combination method

This method uses each subset generated in the last section and combines the subset solutions, returning one or more trial solutions.

In Scatter Search new solutions are retrieved from linear combinations of solutions in the vector space: $S = A + \lambda(B - A)$. If $0 < \lambda < 1$. Convex combinations are generated and the method works within the convex hull, defined by the solutions used in the combination. In contrast, if $\lambda > 1$, the method will explore solutions beyond the region between the two solutions to be combined, or outside the convex hull defined by the subset when more than two solutions are considered in the combination.

To create solution combinations we consider a filter vector applied to the union of solutions, called λ -filter. A partial solution is retrieved by $\lambda(A \cup B)$, where each λ -filter creates a initial trial solution to be improved. The λ -filters are used in Scatter Search as a form of structured combinations of solutions. Instead of drifting within the solution space defined by the reference set RS , the

SS procedure searches each region extensively by applying different λ -filters. Each λ -filter generates a trial solution to be improved by the Improvement Method. A sequence of previously planned λ -filters generates a set of solutions within and beyond regions defined by two or more solutions in which new trial solutions will be chosen for updating the *RS* in an evolutionary fashion.

λ -filters implementation

Given a subset with n -elements, a λ -filter will extract partial information from each of these elements, to create a trial solution. For a defined pattern, for instance “001”, the generation of a set of λ -filter variants is given by shifting the initial sequence.

Consider the subset of solutions A , B , and C , the filter will be applied to the union of these elements. If the value in the λ -filter index is zero then the value of the trial solution index will also be zero. If the value is one, the value of the trial solution index is equal to the value of the union of A , B , and C , as follows:

$$\begin{array}{rcl} A \cup B \cup C & 101 & 010 \\ \lambda\text{-filter} & 001 & 001 \\ \hline \text{trial solution} & 001 & 000 \end{array}$$

Different filters can be obtained for different subsets. For a 2-element subset, the number of indices with value “1” is equal to half of the length of the λ -filter. Thus, the trial solution returns approximately the same number of elements as the original solution. The same approach can be applied to the 3-element subset with 1/3 of the length of the λ -filter with value “1”, and for the 4-element subset with a ratio equal to 1/4.

The Bypass Strategy

Applying λ -filters in subsets with diverse solutions, a bypass strategy is created. For the subset $\{A, B\}$, the diverse solution D that maximizes the distance from the set $\{A, B\}$ is used as a reference point to create two new subsets $\{A, D\}$ and $\{D, B\}$. Instead of finding only solutions between A and B , it is possible to bypass the path using solution D , finding intermediate solutions between A and D and between D and B .

Besides the creation of regions with distant solutions, the structured combinations can be performed within and outside of the defined region. If the solution S is within a region, the intersection must be assured: $S \subseteq \lambda(A \cup B) \cup (A \cap B)$. Otherwise, if the solution is outside the region, the intersection of solutions must be removed: $S \subseteq \lambda(A \cup B) \setminus (A \cap B)$.

11.3.6 Summary of the scatter search procedure

Let Ψ be the set of all filters and Ω the set of all subsets, used at each iteration. The number of filters and the number of subsets is given by the cardinality

of each set, $|\Psi|$ and $|\Omega|$, respectively. Hence the number of evaluated trial solutions is $|\Psi||\Omega|$.

The different gradations in the intensification phase are due to the values assigned to $|\Psi|$ and $|\Omega|$, permitting a planned and structured framework, instead of a random drift. The number of $|\Psi||\Omega|$ evaluations performed at each iteration leads to an aggressive exploration of each region. The general idea is to find the best possible trial solutions in each region defined by each solution. The method gradually updates the *RS* and intensifies the search in each subregion.

The available time to run the program for a particular instance is given by θ , and the run time for the improvement method is given by t . The number of iterations is equal to the two by two combinations of the diverse solutions in the *RS*. The total time θ is equal to the following expression:

$$\theta = \eta C_2^{|\text{RS}^d|} |\Psi| |\Omega| t, \quad \text{with } \eta > 1.$$

In this way, the number of solutions in the Reset can be pre-calculated based on the parameters t , Ψ , Ω , and θ . Or vice-versa, given *RS*, Ψ , Ω , and t , the total time θ can be computed.

The *SS* procedure is described in Figure 11.4, using a pre-defined number of iterations and the five sub-routines of the *SS/PR* template.

Procedure Scatter-Search

I – Initial Phase:

 Initialize the *RS* using the Diversification Generation Method;

II – Evolutionary Phase:

 While $NI < NMAX$

 Generate the next pair of solutions (X, Y) ;

 Subset Generation Method creates Ω ;

 For each subset in Ω

 For each filter in Ψ

 Apply the Solution Combination Method;

 Apply the Improvement Method;

 Apply the Reference Set Update Method;

 Update NI ;

End Procedure.

Figure 11.4 Procedure Scatter-Search.

11.4 EXPERIMENTAL ANALYSIS

We use a set of the most challenging clique DIMACS benchmark instances to evaluate the performance of the proposed Scatter Search algorithm. These

include the “brock” graphs, which contain large cliques “hidden” within much smaller cliques [4], and Sanchis graphs, which are also constructed to be hard to solve: the “gen” and the “san” graphs. We should mention the DIMACS benchmark set is much larger and several other studies on the MCP problem have considered other combinations of test problems. Because our improvement method shares some characteristics of the neighborhood search procedure used in Soriano and Gendreau [26], we consider this algorithm as a reference for comparison purposes.

In Table 11.1, we report comparative results of our Scatter Search algorithm with all three proposed versions of Soriano and Gendreau’s (S&G) tabu search algorithm: Single List TS, Double List TS, and Probabilistic TS. Results are presented for 26 graph instances from the three graph families: 8 of “brock”, 5 of “gen” and 13 of “san”. For each instance, we report the instance name, the optimal solution value, the best solution found, the time till best, the total running time and the number of iterations, for the scatter search algorithm. The remaining columns contain the solution value for S&G algorithms. Bold characters indicate solution values for the algorithm that dominates all the others for the corresponding instance.

The stopping criterion of the scatter search algorithm is defined as the total number of iterations or when the optimal solution is reached. In the latter situation the “time till best” corresponds to the total time used by the algorithm.

Our experiments were performed on a PC Pentium II 400 MHz. The program was coded in C and the Borland C/C++ compiler was used under a Windows 95 operating system. We consider the following set of parameters for our method for all runs: $|RS| = 20$, $|RS^d| = 18$, $|\Psi||\Omega| = 5$, $NMAX = \eta C_2^{|RS^d|}$, with $\eta > 1$, $C_2^{|RS^d|} = 153$, and the tabu list tenure $|T| = \sqrt{|V|}$, where V is the vertex set of the instance. We consider $\eta = 1.2$.

From the upper part of Table 11.1, we can see the scatter search algorithm found the optimal solution for half of the eight “brock” instances, which are known to be very hard to solve. For the “gen” and “san” instances, the optimal solutions were found for all the instances, showing the robustness of the algorithm. The results further show the scatter search algorithm can be advantageously compared with some of the most competitive algorithms for the MCP in terms of solution quality. It appears the scatter search algorithm uses more computation time than the approaches of Soriano and Gendreau [26] (although different machine architectures for running the codes make comparisons imprecise). However, this extra time is clearly compensated by the quality of the solutions produced. The scatter search algorithm dominates the three S&G algorithm variants for 5 problems and finds all the best solutions obtained by every S&G variant for the remaining problems. Also, considering our improvement method uses a somewhat naive tabu search strategy which only relies on a simple version of short-term memory, the results seem to demonstrate clear evidence of the usefulness of the scatter search approach to efficiently explore the solution space in maximum clique problems.

Problem	Opt	Best	time	total	Nb.	S-TL	D-TL	P-TS ^a
	sol.	till best		time	iter			
Brock200_1	21	21	137	137	15	20	21	21
Brock200_2	12	11	21	3162	2	11	11	11
Brock200_3	15	15	2378	2378	169	14	14	14
Brock200_4	17	17	87	87	10	16	16	16
Brock400_1	27	25	583	19624	10	24	24	25
Brock400_2	29	25	1546	19217	29	24	24	25
Brock400_3	31	25	435	19443	7	24	25	24
Brock400_4	33	33	4816	4816	78	25	25	25
gen200_p0.9_44	44	44	56	56	3	44	40	40
gen200_p0.9_55	55	55	51	51	3	55	55	55
gen400_p0.9_55	55	55	1215	1215	13	52	51	54
gen400_p0.9_65	65	65	455	455	4	65	50	65
gen400_p0.9_75	75	75	146	146	2	75	52	75
san200_0.7_1	30	30	17	17	2	30	30	17
san200_0.7_2	18	18	742	742	52	15	18	15
san200_0.9_1	70	70	38	38	2	70	70	70
san200_0.9_2	60	60	38	38	2	60	60	60
san200_0.9_3	44	44	82	82	5	44	44	37
san400_0.5_1	13	13	105	105	2	13	9	8
san400_0.7_1	40	40	105	105	2	40	40	21
san400_0.7_2	30	30	530	530	6	19	30	18
san400_0.7_3	22	22	3580	3580	53	18	17	18
san400_0.9_1	100	100	287	287	2	100	100	100
Sanr200_0.7	18	18	24	24	2	18	18	18
Sanr200_0.9	42	42	116	116	7	42	41	42
Sanr400_0.7	21	21	155	155	2	20	21	21

^a reports the best solution produced over 5 runs

Table 11.1 Computational results of the Scatter Search algorithm for the MCP (CPU times in seconds).

11.5 CONCLUSIONS

Scatter Search provides a systematic and unified framework to embody local search in an evolutionary strategy, exploring selected regions extensively with the objective of finding effective solution combinations.

The objective of this paper is to provide a first experimental analysis of the use of scatter search for solving the maximum clique problem, with the goal of gaining insights that may be useful in the solution of related problems arising in a variety of practical applications. Several innovative features are incorporated in the various subroutines of the scatter search template to apply the method to the MCP.

The improvement method used in our scatter search algorithm includes and extends the neighborhood structures used in the tabu search algorithm developed by Soriano and Gendreau [26], which is one of the most efficient algorithms for the MCP. Soriano and Gendreau tested several diversification strategies with the same neighborhood search procedure and showed that the use of long-term

memory has a great impact in the quality of the solutions produced. In contrast, the tabu search procedure used in the improvement method of the scatter search algorithm only uses a short-term memory based on a fixed length tabu list. However, the quality of the results produced by our algorithm demonstrates the efficiency of the scatter search approach in achieving a suitable diversification strategy. Also, the scatter search implementation shows the robustness of the algorithm – for the Sanchis graphs it reached all the optimal solutions and found the optimal solutions in half of the instances even for the “brock” graphs. These results demonstrate the efficacy of the proposed algorithm for the MCP, while disclosing the usefulness of the scatter search method as a systematic procedure.

Finally, it is important to notice that some of the most efficient algorithms for maximum independent set problem and vertex covering problems (which are equivalent to the MCP) are based on randomized multi-start types of algorithms. Our finding motivate future research to investigate the potential of the scatter search approach to create more strategic procedures to generate trial solutions for multi-start approaches for this class of problems.

Acknowledgments: Cesar Rego’s research has been supported in part by the Office of Naval Research (ONR) grants N000140010598 and N000140010769.

References

- [1] E. Balas and W. Niehaus. Optimized Crossover-Based Genetic Algorithms will be the Maximum Cardinality and Maximum Weight Clique Problems. *Journal of Heuristics*, 4:107–122, 1998.
- [2] R. Battiti and M. Protasi. Reactive Local Search for the Maximum Clique Problem. *Algorithmica*, 29:610–637, 2001.
- [3] C. Berge. *Theory of Graphs and its Applications*. Methuen, 1962.
- [4] M. Brockington and J.C. Culberson. Camouflaging Independent Sets in Quasi-Random Graphs. *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, 26:75–89, 1996.
- [5] L. Cavique, C. Rego, and I. Themido. Estruturas de Vizinhança e Algoritmos de Procura Local para o Problema da Clique Máxima. To appear in: *Revista de Investigação Operacional*.
- [6] N. Christofides. *Graph Theory: An Algorithmic Approach*. Academic Press, 1975.
- [7] V.-D.Cung, T. Mautor, P. Michelon, and A. Tavares. A Scatter Search Based Approach for the Quadratic Assignment Problem. In: *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 165–169, 1997.

- [8] T.A. Feo, M.G. Resende, and S.H. Smith. A Greedy Randomized Adaptive Search Procedure for Maximum Independent Set. *Operations Research*, 42:860–878, 1994.
- [9] C. Fried, A. Hertz, and D. de Werra. Stabulus: A Technique for Finding Stable Sets in Large Graphs with Tabu Search. *Computing*, 42:35–44, 1989.
- [10] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman & Co, 1979.
- [11] M. Gendreau, P. Soriano, and L. Salvail. Solving the Maximum Clique Problem using a Tabu Search Approach. *Annals of Operations Research*, 41:385–403, 1993.
- [12] F. Glover. Heuristics for Integer Programming using Surrogate Constraints. *Decision Science*, 8:156–166, 1977.
- [13] F. Glover. Tabu Search – Part I. *ORSA Journal of Computing*, 1:190–206, 1989.
- [14] F. Glover. Tabu Thresholding: Improved Search by Nonmonotonic Trajectories. *ORSA Journal on Computing*, 7:426–442, 1995.
- [15] F. Glover. A Template for Scatter Search and Path Relinking. *Lecture Notes in Computer Science*, 1363:13–54, 1998.
- [16] F. Glover. Scatter Search and Path Relinking. In: *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, editors, pages 297–316, McGraw Hill, 1999.
- [17] F. Glover and M. Laguna. *Tabu Search*. Kluwer, 1997.
- [18] A. Jagota. Efficient Approximating Max-Clique in a Hopfield-Style Network. In: *IEEE, International Joint Conference on Neural Networks*, vol. 2, pages 248–253, Baltimore, 1992.
- [19] A. Jagota, L. Sanchis, and R. Ganesan. Approximately Solving Maximum Clique using Neural Network Related Heuristics. *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, 26:169–203, 1996.
- [20] D.S. Johnson. Approximation Algorithms for Combinatorial Problems. *Journal of Computer and System Science*, 9:256–278, 1974.
- [21] D.S. Johnson and M.A. Trick (editors). *Clique, Coloring and Satisfiability*. *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, 26, 1996.
- [22] M. Laguna. Scatter Search. Research Report, University of Colorado, Boulder, 1999.

- [23] H. Lourenço, J. Paixão, and R. Portugal. Meta-Heuristics for the Bus-Driver Scheduling Problem. Economic Working Papers Series 304, Universitat Pompeu Fabra, Barcelone, 1999.
- [24] C. Rego and P. Leão. A Scatter Search Tutorial for Graph-Based Permutation Problems. Research Paper HCES-10-00, Hearin Center for Enterprise Science, University of Mississippi, 2000.
- [25] B. Roy. *Algèbre moderne et théorie des graphes*. Tome 2, Dunod, 1970.
- [26] P. Soriano and M. Gendreau. Tabu Search Algorithms for the Maximum Clique. *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, 26:221–242, 1996.

12 THE NOISING METHODS: A SURVEY

Irène Charon and Olivier Hudry

École nationale supérieure des télécommunications
46, rue Barrault, 75634 Paris cedex 13, France
Irene.Charon@enst.fr, Olivier.Hudry@enst.fr

Abstract: The aim of this paper is to summarize the principles and the applications of the *noising methods*, a recent family of combinatorial optimization metaheuristics. We describe their commons features and their variants and we give the list of their applications to different combinatorial optimization problems. We also show how the simulated annealing algorithm and the threshold accepting algorithm can be considered as noising methods when the components of the noising methods are properly chosen.

12.1 THE ORIGINS OF THE NOISING METHODS

At the beginning of the 1990's, we were studying the application of genetic algorithms to combinatorial optimization problems with strong structural constraints, like the ones depicted below. Unfortunately, we got only poor results, even when compared to those provided by repeated descents. So, we thought that maybe the genetic algorithms would be more appropriate if the environment, measured in some ways by the function f that we want to optimize, could fluctuate, as it happens in real life with genetics.

Thus we considered that f is the limit of an evolving function, or equivalently that we do not know precisely the values taken by f : these values are perturbed by *noises*; the range of the noises decreases to zero so that the evolving function converges towards f . Of course, to get an idea of the new performances of our genetic algorithms, we applied the same evolving process to the repeated descents. While the results got by the genetic algorithms remained as poor as before in our experiments, the surprise came from the repeated descents: the results that they provided at the end of this evolving process were much better than those that we got previously within the same CPU time.

The principle of the noising methods was born: to perturb the values taken by f by decreasing noises while applying a local search method. Then we designed in 1992 the first version of a noising method and it appeared in 1993 [10]. Since this date, we have been studying different ways to perturb f and different variants of what has become a family of metaheuristics that we still call *noising methods* because they share the same basic principles.

As for the other metaheuristics (for references on metaheuristics, see for instance [1, 36, 37, 39]), the noising methods are not designed to solve only one special type of problems, but to be applicable to various kinds of combinatorial optimization problems. Such problems may be described as follows: given a finite set S and a function f defined on S , find the optimum (the maximum or the minimum) of f over S and an element of S optimizing f . As maximizing a function f is the same as minimizing $-f$, we may assume without loss of generality that we deal with minimization problems. Thus we shall consider from now on a problem with the following structure:

$$\text{Minimize } f(s) \text{ for } s \in S.$$

The aim of this paper is to present the principles and the applications of the noising methods. We detail their common features and their variants and we list their applications to different combinatorial optimization problems. We also study the links between them and two other metaheuristics: the simulated annealing algorithm and the threshold accepting algorithm; we show how these two metaheuristics can be considered as noising methods when the components of the noising methods are properly chosen.

The paper is organized as follows: the following section recalls some basic considerations about the notion of *neighborhood*. Then, in Section 12.3, we describe the main principles of the noising methods. Section 12.4 is devoted to the links between the noising methods and simulated annealing or threshold accepting methods. The applications of the noising methods are listed in Section 12.5, just before the conclusions.

12.2 ELEMENTARY TRANSFORMATIONS, NEIGHBORHOODS AND DESCENTS

Many metaheuristics (sometimes called also *local search methods*) applied to combinatorial optimization problems are based on *elementary* (or *local*) *transformations*. We call *transformation* any operation which changes a solution s of S into a solution s' of S . Elementary transformations constitute a subset of the set of transformations; they usually consist in changing one feature of s without changing its global structure: for instance, if s is a binary string, an elementary transformation could be to change, for a given i , the bit located in position i in s into its complement. For a given elementary transformation scheme, the set of the solutions s' that we can get by applying such an elementary transformation to a given solution s is called the *neighborhood* $N(s)$ of s , and the elements s' of $N(s)$ are called the *neighbors* of s .

Thanks to such an elementary transformation, we can design an *iterative improvement method* (also called a *descent* for a minimization problem): from the current solution s , we consider a neighbor $s' \in N(s)$ of s ; if we have $\Delta f(s, s') < 0$ with $\Delta f(s, s') = f(s') - f(s)$, then s' becomes the new current solution, otherwise we keep s as the current solution and we try another neighbor of s . Then, we do it again with the current solution until it is impossible to find a neighbor of the current solution s^* which is better than s^* :

$$\forall s' \in N(s^*), f(s') \geq f(s^*).$$

Thus, the solution s^* provided by a descent is a local minimum with respect to the adopted elementary transformation (or, similarly, with respect to the neighborhood induced by it) and is not necessarily an optimal solution.

There exist several ways to explore the neighborhood of a solution s . We recall three of them below. It is of course possible to imagine some other ways to explore the neighborhood, including by mixing the three following strategies.

- It can be done *at random* (as in a classic simulated annealing): then s' is randomly chosen in $N(s)$. One drawback of this strategy is that it is memoryless. We may scan the same neighbor several times, instead of considering another neighbor. Moreover, two successively generated neighbors are globally more different than when generated systematically (see below); then it is more difficult to benefit by the previous trials in order to reduce the amortized complexity, which usually involves a great amount of CPU time. Finally, it is difficult to be sure to reach a local optimum. On the other hand, if the number of trials is very low, it can sample the solution-space more efficiently than the strategies below. Similarly, if the size of the neighborhood is very large, it can be the best way of looking for an interesting neighbor.
- The exploration can also be *systematic* or *cyclic* (see [23]): the neighbors are ranked in a certain order and they are all considered in this order once, before being considered for a second time. A better neighbor than the current solution is accepted as soon as it has been discovered; for a descent, this process is applied until a complete cyclic exploration of the neighborhood has been performed without finding any better neighbor: then the final solution is a local minimum with respect to the adopted elementary transformation.
- The third classic exploration, that we can call *exhaustive*, is the one applied for instance in a classic tabu search or in a steepest descent: the whole neighborhood is explored in order to find the best neighbor of the current solution. The main drawback of this strategy is that, when the size of the neighborhood is quite large, it can be very long to explore it entirely between two adopted transformations.

The noising methods are based on elementary transformations. The next section is devoted to the principles and variants of these methods.

12.3 PRINCIPLES OF THE NOISING METHODS

As said above, to minimize the function f , the noising methods do not take the true values of f into account but consider that they are perturbed in some way by *noises* in order to get a “noised” function f_{noised} . It means that, with respect to what happens in a descent, when the variation $\Delta f(s, s')$ is computed in order to know whether s' is accepted instead of s or not, we do not compute $\Delta f(s, s')$ from the true values taken by f , but we consider that these values taken by f are perturbed by *noises* and thus we compute in fact the variation $\Delta f_{noised}(s, s')$. During the run of the algorithm, the range of the perturbing noises decreases (typically to zero, but it often happens that we may stop the process before), so that, at the end, there is no significant noise and the optimization of f_{noised} leads to the same solution as the one provided by a descent applied to f with the same initial solution. Thus it is necessary to specify how to perturb f to get a noising method.

More generally, several questions must be answered in order to design a noising method scheme:

1. How to perturb f in order to get f_{noised} ?
2. How and when to make the range of noise decrease?
3. How to choose the maximum (and initial) and the minimum (and final) values of the range of noise?
4. How to explore the neighborhood?
5. Is it possible to add other ingredients to get interesting variants?

Of course, the answers to these questions depend on the problem to solve and on what the user wants to do: he or she has the possibility to choose his or her own components, and so, by designing his or her own noising scheme, to diversify his or her investigations. Nonetheless, we give below some possible answers to the previous questions.

12.3.1 How to perturb f ?

Though this distinction is a little contrived, we may distinguish three main ways to perturb f : by *noising the data*, by *noising the variations of f* , or by *forgetting a part of the data*. They are described more precisely below.

12.3.1.1 Noising the data. The first possibility to perturb f consists in adding the noises to the data. For instance, if f is a linear function of n variables x_i (with $1 \leq i \leq n$):

$$f(s) = \sum_{i=1}^n a_i x_i,$$

we may perturb the data a_i by adding some noises r_i to them:

$$f_{noised}(s) = \sum_{i=1}^n (a_i + r_i)x_i.$$

We may insert this way of noising the data in the following noising method scheme:

```

Initializations
repeat
    add noises to the data to get  $f_{noised}$ 
    apply a descent to  $f_{noised}$  from the solution computed
        at the previous iteration
    reduce the range of noise
until the range of noise is low enough
apply a final descent to  $f$  from the solution provided by the previous loop.

```

In other words, we first compute an initial solution. Then, for each repeat-loop, noises are added to the data and a descent is applied with respect to these noised data, i.e., to f_{noised} ; the solution found during this application of the descent becomes the initial solution of the next application of the descent. The range of noise decreases between two descents to a fixed value (zero, for example). Notice that it is not always necessary to apply a complete descent to f_{noised} : for instance, the exploration of the neighborhood of the local (with respect to the noised data) minimum found by the descent is systematically useless. Thus it is often better (because it saves CPU time that we can use differently) to apply just the beginning of a descent and to stop it when a given number of (adopted or not) elementary transformations has been performed (for instance, the number of elementary transformations necessary to scan a neighborhood a few times). Anyway, it is better to apply the final descent to f completely.

Usually, the added noises are chosen randomly according to a given probability law. For instance, in the first design of a noising method [10], the noises r_i ($1 \leq i \leq n$) are chosen with a uniform law in an interval $[-r, +r]$, where r is the rate of noise at the considered iteration, and the initial (and maximum) value r_{max} of r depends on the data. But, here also, we may imagine different possibilities to choose the noises. Some examples are:

- a_i becomes $a_i + r_i$;
- a_i becomes $a_i \times (1 + r_i)$;
- the probability law is not necessarily uniform; it can be a Gaussian law or something else; and
- the interval in which r_i is drawn is not necessarily centered at the origin.

This pattern has been applied in particular in the first noising method [10] and, since, in [2]–[4], [6]–[9], [11], [20]–[22], [26]–[32], [34, 35, 38].

12.3.1.2 Noising the variations of f . We tried another possibility (see [12]): it consists in perturbing the variations of f . When a neighbor s' of s is tried, we do not consider the genuine variation $\Delta f(s, s')$ but a noised variation $\Delta f_{noised}(s, s')$ defined by:

$$\Delta f_{noised}(s, s') = \Delta f(s, s') + r(s, s')$$

where $r(s, s')$ denotes the noise (in fact, it depends not only on s and s' , but also on the number of the iteration: if s and s' are considered twice, the noise $r(s, s')$ is not necessarily the same).

As for before, the probability law followed by the noises $r(s, s')$ can be uniform on an interval centered at the origin or not, or Gaussian, or something else (for instance, we tried a distribution involving the hyperbolic tangent of the variation of f in the first version of [12]). This way of perturbing f by adding the noises to its variations has been developed in [12]–[18] and [40] (similar ideas are also considered in [33] to study the convergence of such metaheuristics). We will see latter that an appropriate choice of the probability law (and of the other elements of the noising scheme) leads to the pattern of a classic simulated annealing or of a threshold accepting algorithm.

The pattern of this kind of noising can be the following:

```

Initializations
repeat
    repeat
        consider a neighbor  $s'$  of the current solution  $s$ 
        compute the noised variation of  $f$ :
            
$$\Delta f_{noised}(s, s') = \Delta f(s, s') + r(s, s')$$

            where  $r(s, s')$  denotes the random noise
        if  $\Delta f_{noised}(s, s') < 0$ , then adopt  $s'$  instead of  $s$ 
    until a first criterion is fulfilled
    reduce the range of noise
until a second criterion is fulfilled
apply a final descent to  $f$  from the solution provided by the previous loop.

```

In this pattern, the first criterion (in the inner repeat-loop) gives the number of elementary transformations tried with a given rate of noise. The second criterion gives the number of decreases of the rate of noise.

12.3.1.3 Forgetting a part of the data. The third and most recent way [14] (similar ideas are developed in [9]) consists in perturbing f by forgetting a part of the data. For instance, in [15], we consider the problem consisting in partitioning the vertex-set of a graph of which the edges have negative or positive weights in order to minimize the sum of the weights of the edges with their two extremities in the same set. Then, we designed two ways to “forget” a part of the data:

- In the first one, a part of the vertices is selected and the weight of an edge is taken into account for the computation of f_{noised} only if its two

extremities have been selected; in this process, the rate of noise (less than or equal to one) gives the ratio of “forgotten” vertices.

- In the second way, we do the same with the edges instead of the vertices: a part of the edges is selected and an edge is involved in the computation of f_{noised} if and only if it has been selected (for the problem tackled in [15], it is the same as giving a weight equal to zero to a forgotten edge).

The scheme of such a noising method can be the following:

```

Initializations
repeat
    forget a proportion of data equal to the rate of noise
    apply a descent to the remaining data
        from the solution computed at the previous iteration
    reduce the range of noise
until the range of noise is low enough
apply a final descent to  $f$  from the solution provided by the previous loop.

```

As for the noising of the data, it is not always necessary to apply a complete descent (except the last one).

This way of noising f gave very good results for the functions studied in [9] and in [15], but it is not obvious that it would be the general case: more experiments are still necessary to draw conclusions.

It is worth noticing that, from a theoretical point of view, the first and third ways to noise f are in fact special cases of the second type, since the rule to know whether a neighbor s' is accepted or not instead of the current solution s comes down to know whether a number depending on s and s' (the noised variation of f) is negative or not. For instance, for the traveling salesman problem with the neighborhood induced by the usual 2-opt transformation, noising the data with a uniform noise is the same as noising the variation with a noise following a probability law given by the sum of four uniform laws. Anyway, it is usually more convenient to apply the first or the third ways of noising f than their equivalents with respect to the noising of the variations, because it can be less natural, more tedious, and sometimes very difficult to find the equivalent probability law.

12.3.2 How and when to make the range of noise decrease?

In the first noising method, the rate of noise r decreases arithmetically during the running of the algorithm from a maximum rate of noise value r_{max} to zero (thus the arithmetically decreasing ratio, the maximum rate of noise, and the total number of applied noised descents are related). It is also the case for the majority of the applications reported in this paper, sometimes with a minimum value r_{min} of the rate of noise which is not equal to zero. Then, if N denotes the total number of decreases, the arithmetically decreasing rate is equal to $(r_{max} - r_{min})/N$.

Nevertheless, other kinds of decreases have been studied. More precisely, a geometrical decreasing was tried in [12, 13], associated with a “logarithmic noise” (the noise follows a probability law given by the logarithm of a uniform law) as well as an arithmetical one associated with a uniform noise. Still in [12], we tried another possibility (for a noising scheme inspired by simulated annealing) in which the rate of noise was allowed to increase from time to time. More precisely, let k be a given number; a function α is designed to give the number $\alpha(i)$ of bad elementary transformations that we would like to accept between iterations i and $i + k$; if the number of bad elementary transformations really accepted between iterations i and $i + k$ is less than $\alpha(i)$, then the rate of noise increases, otherwise it decreases. The experiments done in [12] for the traveling salesman problem show that this “adaptive” method can lead to very good results, often better than those provided by the same method but with a decreasing rate of noise.

It seems that the frequency of decreases is not a crucial point, at least of course if its value is not too low (for instance, decreasing the rate of noise from its maximum value r_{max} to zero suddenly is usually not a good choice). For example, the rate of noise may decrease after each elementary transformation, at least theoretically; this could avoid to deal with an extra parameter to tune. Anyway, for practical reasons, it can be more convenient to make the rate of noise decrease less often, for instance when an exploration of the neighborhood is completed (or when a number of elementary transformations equal to the size of the neighborhood has been performed).

12.3.3 How to choose r_{max} and r_{min} ?

12.3.3.1 Choice of r_{max} . The choice of r_{max} seems to be the most important one (of course after the choice of the number of elementary transformations that the user wants to apply to solve his or her problem. This value is directly related to the CPU time that he or she accepts to spend). So, in general, it is a parameter that the user must tune.

Anyway, some experiments done with the traveling salesman problem (see [16]) show that the sensitiveness of this parameter is not very sharp: a value of more or less 10 % from the “best” value of r_{max} gives almost the same final result and even a value of more or less 20 % gives very good results. One consequence is that it is not always necessary to find a sharp tuning for r_{max} .

Moreover, it is sometimes possible to tune r_{max} automatically, as in [18] (see also [15] and [17]). In this paper, we designed a generic automatic noising scheme which can be applied with various types of noise or to various problems by changing only the type of noise or the features defining the problem; no parameter of this generic method is changed from one noising scheme to another or from one problem to another. The only parameter given by the user is the CPU time that he or she wishes to spend to solve his or her problem. Broadly speaking, the principle of this automatically tuned variant consists in applying the considered noising method several times, in the sense that the rate of noise decreases several times from a maximum value to zero. The duration of each

run is twice that of the run performed just before; the first run is very short. This repetition of the noising method succeeded, for the problems in [15, 17, 18], in computing a suitable value for r_{max} which is improved during the process (see [18] for details).

12.3.3.2 Choice of r_{min} . In general, r_{min} is a parameter to tune. It must be chosen so that minimizing f_{noised} with a rate of noise less than r_{min} would lead to the same result as minimizing f itself. If so, it is clearly useless to try such values for the rate of noise: we can only waste CPU time without improving the current solution. So, choosing a value not equal to zero for r_{min} may save CPU time. Anyway, if the user does not want to tune an extra parameter, it is always possible to choose $r_{min} = 0$, even if it consumes a little more CPU time. When it is not easy to have a broad idea of a good value for r_{min} , it can even be a good deal to do so, because tuning a parameter is not always obvious, especially if the user is not an expert. Moreover, a too high value of r_{min} may damage the efficiency of the method very much. So, in the automatically tuned noising method of [18], we preferred to fix $r_{min} = 0$ and to increase the CPU time a little bit, rather than having to tune r_{min} .

12.3.4 Exploration of the neighborhood

In the majority of the applications of the noising methods reported here, the exploration of the neighborhood is systematic (or cyclic; see Section 12.2). But it happens that the exploration may be random (especially when noise perturbs the data, as described in Section 12.3.1.1), or even partly systematic and partly random. For instance, if the elementary transformation involves two parameters (as for example the 2-opt transformation usually applied to the traveling salesman problem), the exploration can be done randomly on the first parameter and, for each value of the first parameter, the exploration can be systematic.

As said above, the main advantages of a systematic exploration are that we may sometimes benefit by the scanning of the previous neighbors to reduce the amortized complexity and to increase the diversity of the exploration by avoiding to scan twice the same neighbor before scanning a neighbor not yet considered. Another advantage of such a strategy is that we can save CPU time with respect to an exhaustive exploration, but also with respect to a random one because drawing random numbers is a procedure which can consume a rather great amount of CPU time.

12.3.5 Other ingredients

Independently of the different schemes that we can get by combining the possibilities described above, it is possible to design some other variants. The two that we detail below can be applied to other metaheuristics. In our experiments with noising methods, they often appear fruitful.

12.3.5.1 Alternation of noised and unnoised phases. The first variant consists in alternating noised phases with unnoised descents. More precisely, in order to stay closer to the original function f , we may apply a given number ν of elementary transformations with respect to the noised function f_{noised} , then a descent with respect to f until a local minimum is reached, then again ν noised trials, then a descent with respect to f , and so on. This variant usually allows to check a fair number of local minima (with respect to the original data) which could provide good solutions.

For the first type of noising methods (noises are added to the data), we may alternate a noised descent (that is, a descent with respect to f_{noised} ; of course, new noises are computed before applying each noised descent) and an unnoised one (that is, a descent with respect to f). For the second type (noises are added to the variations of f), it seems to be a good choice to give to the noised phase a number of elementary transformations which is about the same as the number of elementary transformations performed by a descent. From our experiments, it appears that a descent performs an average of about 4σ elementary transformations, where σ is the size of the neighborhood. Then we can perform successively $\nu = 4\sigma$ noised elementary transformations, an unnoised descent, ν noised elementary transformations, an unnoised descent, and so on.

12.3.5.2 Periodic restarts from the best computed solution. The second variant consists in coming back to the best computed solution periodically. Indeed, because of the bad transformations sometimes accepted, it may happen that we leave an interesting part of the space of solutions for a less interesting one. So one possible strategy is to periodically restart the current solution with the best solution found since the beginning. Of course, it is useless (and even harmful) to restart the current solution too often. In order not to introduce a new parameter, the restart period was fixed as follows in [12]. Let $\gamma^2\sigma$ be the total number of elementary transformations performed by the method (where σ is the size of the neighborhood); then the current solution is restarted with the best computed solution after every cluster of about $\gamma\sigma$ elementary transformations (in other words, there are about γ restarts), what seemed to give a good frequency.

12.4 THE NOISING METHODS AS GENERALIZATIONS OF OTHER METAHEURISTICS

By combining all the ingredients detailed above, we may get many different noising methods, including the simulated annealing method or the threshold accepting algorithm. It is what we show now (the possibility of such a generalization is also noticed in [33]).

12.4.1 Links between the noising methods and simulated annealing

Indeed, we may consider that the second type of noising (noises are added directly to the variations of f) is a generalization of simulated annealing if we choose properly the parameters, especially the probability distribution.

In simulated annealing, the current solution s is replaced by one of its neighbors s' with a probability equal to $\min\{1, \exp(-\Delta f(s, s')/\theta)\}$, where θ is the decreasing parameter called *temperature*; then a bad transformation ($\Delta f(s, s') > 0$) is accepted if we have:

$$\exp(-\Delta f(s, s')/\theta) > p,$$

where p is a random number uniformly drawn into $]0, 1[$ or, equivalently, if the following condition is fulfilled:

$$\Delta f(s, s') + \theta \ln p < 0.$$

Thus, it gives the same result as adding to the variation of f a noise equal to $\theta \ln p$. Here the probability law is given by the logarithm of a uniform random variable drawn into $]0, 1[$, and θ gives the value of the rate of noise. Then it is easy to choose the other ingredients of the noising method to get a classic simulated annealing (in particular, the decreases of the rate of noise θ will be geometric in this scheme; the exploration of the neighborhood is usually a random one for simulated annealing, though a systematic one is sometimes applied, see [23]). Variants of this “logarithmic scheme” have been studied in [12, 13, 15], as well as a uniform distribution.

12.4.2 Links between the noising methods and threshold accepting algorithms

Similarly, we may consider that the noising methods are a generalization of the threshold accepting algorithms designed by Dueck, Scheurer and Wirsching [24, 25].

In such a method, the current solution s is replaced by one of its neighbors s' if the variation $\Delta f(s, s')$ does not get bigger than a given threshold. This threshold depends on the iteration and decreases to zero during the process. So, with respect to simulated annealing, the main difference relies in the fact that the acceptance criterion is no longer the exponential Metropolis criterion. More precisely, if k denotes the current iteration, s' is accepted instead of s if we have

$$f(s') - f(s) < \theta_k,$$

where θ_k is the threshold of the current iteration, with $\theta_k \geq 0$, $\theta_{k+1} \leq \theta_k$ and $\theta_\nu = 0$ if ν denotes the total number of elementary transformations performed.

This criterion avoids the computation of an exponential and the call to a random number generator, which usually saves CPU time. One of the main difficulties of this method is the determination of the appropriate values for the thresholds θ_k , though Althöfer and Koschnick [5] have related some convergence

properties of threshold accepting methods to those of simulated annealing (but their proofs are not constructive).

It is quite easy to see that these thresholds θ_k can be seen as noises subtracted from the variation $\Delta f(s, s')$ of f . Thus, threshold accepting algorithms can be considered as noising methods with the second way of perturbing f described above, and with a noise equal to $-\theta_k$ when iteration k is performed.

12.5 APPLICATIONS OF THE NOISING METHODS

The first paper presenting a noising method [10] dealt with the following *clique partitioning problem*: given a complete non-oriented graph $G = (X, E)$ of which the edges are weighted by (negative or positive) integers, find a partition of X into $p(G)$ subsets so that the sum of the weights of the edges with their two extremities in the same subset is minimum. As explained above, in this noising method, the noises were added to the weights of the edges before applying a descent. This can be done for any problem represented by a weighted graph and, more generally, for problems with numerical data.

But sometimes the problem can be more “structural” than “numerical”. It is one of the reasons that led us to design noising methods in which noises are added to the variations of f and not to the data. Another possibility to deal with such problems is that adopted by Bogdanova [7] for the following NP-hard problem in coding theory. Let n and d be two positive integers; let Z be the set $\{0, 1, 2, 3\}$ and let Z^n be the set of all n -tuples defined over Z . A set $C \subseteq Z^n$ is called a 4-ary M -code if $|C| = M$, and a 4-ary (n, M, d) -code if $|C| = M$ and if the Hamming distance δ between two distinct elements of C is at least d . The largest value of M such that a 4-ary (n, M, d) -code exists is denoted by $A_4(n, d)$. To determine lower bounds of $A_4(n, d)$, Bogdanova defines the function f^M over the set of 4-ary M -codes as follows:

$$f^M(C) = |\{(x, y) \in C^2 \text{ such that } \delta(x, y) < d\}|$$

and, for a given value M , she tries to minimize f^M over the set of 4-ary M -codes C . If such a M -code C with $f^M(C) = 0$ is found, then $A_4(n, d) \geq M$. In this case, the same process is applied to f^{M+1} . In order to minimize f^M for any given value of M by the help of the noising method, Bogdanova gives a random value $k(v) \in [1 - r, 1 + r]$ to each element v of Z^n , where r is the arithmetically decreasing rate of noise. Then she defines the noised function f_{noised}^M over the set of 4-ary M -codes by:

$$f_{noised}^M(C) = \sum_{(x,y) \in C^2 \text{ and } \delta(x,y) < d} \frac{k(x) + k(y)}{2}.$$

Notice that, when the rate of noise r is equal to 0, $f_{noised}^M = f^M$. By means of this noising method, she succeeded to break several lower bounds of $A_4(n, d)$.

Anyway, even if the applications of the noising methods are not yet very numerous, it is not possible to detail all of them here. So we just summarize the field and references for the applications that we know:

- partitioning a weighted graph into cliques [10, 14, 15, 18, 19, 27, 28, 38];
- traveling salesman problem [2, 12, 16, 18, 30, 31];
- aggregation of linear orders into a median order (linear ordering problem) [13, 17]–[19];
- scheduling problems [4, 34, 35];
- covering and packing problems in coding theory [7, 20, 21];
- 0-1 multidimensional knapsack problem [8, 26, 29];
- the multi-resource generalized assignment problem [3, 6];
- the prize-collecting Steiner tree problem [9];
- multi-criteria decision aid [11];
- the task allocation problem [22];
- the design of discrete manufacturing processes [32]; and
- the alignment of graphemes and phonemes in linguistics [40].

The noising methods involved in these applications may follow the basic noising method scheme developed in [10] as well as the two other types of noising f . Their results usually (but not always) show that they can compute good solutions, often better than those found by other metaheuristics as simulated annealing.

12.6 CONCLUSION

It would be unwise to conclude that the noising methods may solve any NP-hard problem efficiently. For instance, we tried to apply them to the computation of some Ramsey numbers, but we did not succeed to improve the currently best results.

Thus, the aim of this paper is only to show, through the principles and the applications of the noising methods, that this family of metaheuristics deserves interest because they are simple to implement and because they may provide good solutions within reasonable CPU times for quite different problems. There remain the problem of choosing the noising method and, then, that of tuning the parameters of the method. Some studies already mentioned [15, 18] show that it is possible to tune them automatically so that the user has just to give the desired CPU time before applying these methods.

Another subject which could be investigated in the future deals with the convergence of the noising methods. If we consider them as generalizations of simulated annealing or of threshold accepting methods, we benefit from the convergence results established for these metaheuristics (see for instance [1] for references about simulated annealing and [5] for threshold accepting methods):

they show that there exist some noising schemes (with an infinite number of iterations) which surely converge towards an optimal solution. Notice also that Johnson and Jacobson studied in [33] the convergence of the first noising type (noises added to the data): they give sufficient conditions to converge (still with an infinite number of iterations) towards an optimal solution.

These aspects of the noising methods (convergence and parameter tuning) as well as others such as their application to other combinatorial optimization problems or their hybridization with other metaheuristics (especially with genetic algorithms, as in [13]) will surely be the topics of our future research efforts devoted to this field.

References

- [1] E.H.L. Aarts and J.K. Lenstra (editors). *Local Search in Combinatorial Optimization*. Wiley, 1997.
- [2] B. Alidaee. Experimental Design for Combinatorial Optimization Problems. Presented at *INFORMS New Orleans 1995*, New Orleans, 1995.
- [3] B. Alidaee, M. Amini, and G.A. Kochenberger. Hybrid Metaheuristic Approaches to the Multi-Resource Generalized Assignment Problem: a Case Study of Tabu Search, Space Smoothing, and Noising Methods. *2nd Metaheuristics International Conference (MIC-97)*, page 131, Sophia-Antipolis, 1997.
- [4] B. Alidaee, J.T. Naidu, and E.L. Gillenwater. Heuristic Algorithms for Multiple Machine Scheduling with the Objective of Minimizing Total Weighted and Unweighted Tardiness. Presented at *INFORMS Dallas 1997*, Dallas, 1997.
- [5] I. Althöfer and K.-U. Koschnick. On the Convergence of “Threshold Accepting”. *Applied Mathematics and Optimization*, 24:183–195, 1991.
- [6] M.M. Amini, B. Alidaee, and M.J. Racer. Alternative Metaheuristics Approaches to the Multi-Resource Generalized Assignment Problem. Presented at *INFORMS Dallas 1997*, Dallas, 1997.
- [7] G. Bogdanova. Optimal Codes Over an Alphabet of 4 Elements. *Proceedings of the Fifth International Workshop on Algebraic and Combinatorial Coding Theory*, pages 46–53, Sozopol, 1996.
- [8] P. Boucher and G. Plateau. Étude des méthodes de bruitage appliquées au problème du sac à dos à plusieurs contraintes en variables 0-1. *Actes des 5es Journées nationales sur la résolution pratique de problèmes NP-complets (JNPC'99)*, pages 151–162, Lyon, 1999.
- [9] S.A. Canuto, C.C. Ribeiro, and M.G.C. Resende. Local Search with Perturbations for the Prize-Collecting Steiner Tree Problem. To appear in: *Networks*.

- [10] I. Charon and O. Hudry. The Noising Method: A New Combinatorial Optimization Method. *Operations Research Letters*, 14:133–137, 1993.
- [11] I. Charon and O. Hudry. An Application of the Noising Method to MCDA. *Abstracts of the 14th European Conference of Operations Research*, page 15, Jerusalem, 1995.
- [12] I. Charon and O. Hudry. Mixing Different Components of Metaheuristics. In: *Metaheuristics: Theory and Applications*, I.H. Osman and J.P. Kelly, editors, pages 589–603, Kluwer, 1996.
- [13] I. Charon and O. Hudry. Lamarckian Genetic Algorithms Applied to the Aggregation of Preferences. *Annals of Operations Research*, 80:281–297, 1998.
- [14] I. Charon and O. Hudry. Variations on the Noising Schemes for a Clustering Problem. *Extended Abstracts of the Third Metaheuristics International Conference*, pages 147–150, Angra dos Reis, 1999.
- [15] I. Charon and O. Hudry. Noising Methods for a Clique Partitioning Problem. Submitted for publication.
- [16] I. Charon and O. Hudry. Application of the Noising Method to the Travelling Salesman Problem. *European Journal of Operational Research*, 125:266–277, 2000.
- [17] I. Charon and O. Hudry. A Branch and Bound Algorithm to Solve the Linear Ordering Problem for Weighted Tournaments. To appear in: *Discrete Applied Mathematics*.
- [18] I. Charon and O. Hudry. Automatic Tuning of the Noising Methods. Submitted for publication.
- [19] I. Charon and O. Hudry. Descent with Mutations for the Aggregation of Relations. Submitted for publication.
- [20] I. Charon, O. Hudry, and A. Lobstein. A New Method for Constructing Codes. *Proceedings of the 4th International Workshop on Algebraic and Combinatorial Coding Theory*, pages 62–65, Novgorod, 1994.
- [21] I. Charon, O. Hudry, and A. Lobstein. Application of the Noising Methods to Coding Theory. *Abstracts of the 14th Triennial Conference of the International Federation of Operational Research Societies*, page 174, Vancouver, 1996.
- [22] W.-H. Chen and C.-S. Lin. A Hybrid Heuristic to Solve a Task Allocation Problem. *Computers and Operations Research*, 27:287–303, 2000.
- [23] K.A. Dowsland. Simulated Annealing. In: *Modern Heuristic Techniques for Combinatorial Problems*, C. Reeves, editor, pages 20–69, McGraw-Hill, 1995.

- [24] G. Dueck and T. Scheurer. Threshold Accepting: A General Purpose Optimization Algorithm Appearing Superior to Simulated Annealing. *Journal of Computational Physics*, 90:161–175, 1990.
- [25] G. Dueck and J. Wirsching. Threshold Accepting Algorithms for Multi-Constraint 0-1 Knapsack Problems. Technical Paper TR 89 10 016, IBM Heidelberg Scientific Center, 1989.
- [26] A. Fréville, S. Hanafi, and A. El Abdellaoui. Noising Method for the 0-1 Multidimensional Knapsack Problem. *Premières rencontres francophones de recherche opérationnelle*, page 53, Mons, 1995.
- [27] D. Guillaume and F. Murtagh. An Application of XML and XLink using a Graph-Partitioning Method and a Density Map for Information Retrieval and Knowledge Discovery. In: *Astronomical Data Analysis Software and Systems VIII*, D.M. Mehringer, R.L. Plante, and D.A. Roberts, editors, pages 278–282, 1999.
- [28] D. Guillaume and F. Murtagh. Clustering of XML Documents. To appear in: *Computer Physics Communications*.
- [29] S. Hanafi, A. Fréville, and A. El Abdellaoui. Comparison of Heuristics for the 0-1 Multidimensional Knapsack Problem. In: *Metaheuristics: Theory and Applications*, I.H. Osman and J.P. Kelly, editors, pages 449–465, Kluwer, 1996.
- [30] C.-P. Hwang. *Global and Local Search Heuristics for the Symmetric Travelling Salesman Problems*. PhD Thesis, University of Mississippi, 1996.
- [31] C.-P. Hwang, B. Alidaee, and J.D. Johnson. A Tour Construction Heuristic for the Travelling Salesman Problem. *Journal of the Operational Research Society*, 50:797–809, 1999.
- [32] S.H. Jacobson, K.A. Sullivan, and A.W. Johnson. Discrete Manufacturing Process Design Optimization using Computer Simulation and Generalized Hill Climbing Algorithms. *Engineering Optimization*, 31:247–260, 1998.
- [33] A.W. Johnson and S.H. Jacobson. On the Convergence of Generalized Hill Climbing Algorithms. Submitted for publication.
- [34] R.B. Kethley. *Single and Multiple Machine Scheduling to Minimize Total Weighted Late Work: An Empirical Comparison of Scheduling Rules, Algorithms and Meta-Heuristics using Taguchi Loss Functions*. PhD Thesis, University of Mississippi, 1997.
- [35] J. Naidu, B. Alidaee, and E. Gillenwater. Heuristic Approaches to Minimize Tardiness Problems on Single Machine with Sequence Dependent Set-Up Times. Presented at *INFORMS Dallas 1997*, Dallas, 1997.

- [36] I.H. Osman and J.P. Kelly (editors). *Metaheuristics: Theory and Applications*. Kluwer, 1996.
- [37] C. Reeves (editor). *Modern Heuristic Techniques for Combinatorial Problems*. McGraw-Hill, 1995.
- [38] V. Sudhakar and C. Siva Ram Murthy. A Modified Algorithm for the Graph Partitioning Problem. *Integration, the VLSI Journal*, 22:101–113, 1997.
- [39] S. Voss, S. Martello, I.H. Osman, and C. Roucairol (editors). *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Kluwer, 1998.
- [40] F. Yvon. *Prononcer par analogie: motivation, formalisation et évaluation*. Ph.D. Thesis, ENST, Paris, 1996.

13 STRATEGIES FOR THE PARALLEL IMPLEMENTATION OF METAHEURISTICS

Van-Dat Cung¹, Simone L. Martins²,
Celso C. Ribeiro², and Catherine Roucairol¹

¹Laboratoire PRISM-CNRS

Université de Versailles

45 avenue des Etats Unis, 78035 Versailles Cedex, France

{Van-Dat.Cung,Catherine.Roucairol}@prism.uvsq.fr

²Department of Computer Science

Catholic University of Rio de Janeiro

R. Marquês de São Vicente 225

Rio de Janeiro 22453-900, Brazil

{simone,celso}@inf.puc-rio.br

Abstract: Parallel implementations of metaheuristics appear quite naturally as an effective alternative to speed up the search for approximate solutions of combinatorial optimization problems. They not only allow solving larger problems or finding improved solutions with respect to their sequential counterparts, but also lead to more robust algorithms. We review some trends in parallel computing and report recent results about linear speedups that can be obtained with parallel implementations using multiple independent processors. Parallel implementations of tabu search, GRASP, genetic algorithms, simulated annealing, and ant colonies are reviewed and discussed to illustrate the main strategies used in the parallelization of different metaheuristics and their hybrids.

13.1 INTRODUCTION

Although metaheuristics provide quite effective strategies for finding approximate solutions to combinatorial optimization problems, the computation times associated with the exploration of the solution space may be very large. With

the proliferation of parallel computers, powerful workstations, and fast communication networks, parallel implementations of metaheuristics appear quite naturally as an alternative to speedup the search for approximate solutions. Several strategies have been proposed and applied to different problems. Moreover, parallel implementations also allow solving larger problems or finding improved solutions, with respect to their sequential counterparts, due to the partitioning of the search space and to more possibilities for search intensification and diversification.

Therefore, parallelism is a possible way not only to reduce the running time of local search algorithms and metaheuristics, but also to improve their effectiveness and robustness. The latter are likely to be the most important contribution of parallelism to metaheuristics. Robust implementations can be obtained by the use of different combinations of strategies and parameter settings at each processor, leading to high quality solutions for different classes of instances of the same problem, without too much effort in parameter tuning. We present in the next section some current trends in parallel computing, concerning in particular architectural developments, parallel programming environments, and types of parallelism and parallel programming models. In Section 13.3, we draw some considerations about speedup and efficiency in parallel systems. We discuss some recent results showing that it is possible to achieve linear speedups in the time to find a solution within some target value, using parallel implementations of different metaheuristics based on multiple independent processors, such as simulated annealing, iterated local search, tabu search (under some conditions), and GRASP. Next, we present in Section 13.4 a classification of parallel implementations of metaheuristics, considering the underlying parallel local search strategy and using the notion of search threads. Different parallelization strategies are discussed and compared. Parallel implementations of tabu search, GRASP, genetic algorithms, simulated annealing, and ant colonies are reviewed in Section 13.5. Some concluding remarks are drawn in the last section.

13.2 CURRENT TRENDS IN PARALLEL COMPUTING

In this section, we review some trends and fundamental issues in parallel computing, which interfere in the design and in the efficiency of parallel algorithms.

13.2.1 Architectures

The architectures of parallel machines have considerably diversified in the nineties [18, 34, 64]. In *shared-memory* machines, all processors are able to address the whole memory space and communication between tasks is achieved through read and write operations on the shared memory. *Distributed-memory* machines have their memory physically distributed among the processors. Each processor can only address its own memory and communication among processes executing on different processors is performed by messages passed through the communication network. The MIMD family (*Multiple Instruction*

Stream, Multiple Data Stream machines, according with Flynn's taxonomy [84]) evolved from shared-memory machines with a few processors (Sequent Balance, Encore Multimax) to distributed-memory machines with hundreds or even thousands of processors interconnected by networks with different topologies, such as the hierarchical ring for the KSR, the two-dimensional grid for the Intel Paragon, the three-dimensional toric for the CRAY T3D/E, and multi-stage switches for the IBM-SP systems, among many others. Synchronous SIMD (*Single Instruction Stream, Multiple Data Stream*) massively parallel machines with up to 65536 4- or 8-bit processors such as MasPar MP-1 and MP-2, and the popular Connection Machines CM-1 and CM-2 have also been developed. Their processors had a small memory capacity (e.g. 16 Kbytes on the MasPar MP-1) and were interconnected by specific networks such as the two-dimensional toric grid for the MasPar machines and the hypercube for CM-1 and CM-2.

Since development costs and commercial prices turned out to be extremely high, only large and well established companies such as IBM, SGI-Cray, and Intel still propose parallel machines with the same fundamental characteristics of the above. Several companies have even disappeared, as it was the case for Thinking Machines Corporation which developed and commercialized CM-1 and CM-2.

The end of the nineties has witnessed the resurgence of shared memory multiprocessor machines (Symmetric MP or, simply, SMP), ranging from two to a few hundred processors (SGI Origin). However, the most important innovation in recent years is the connection of SMP machines through fast local networks (Myrinet, SCI, ATM, GigaEthernet) and the connection of general purpose parallel machines through national or international very high speed networks. Besides allowing for scalability and fault-tolerance, networks of computers present good cost/performance ratios when compared to other parallel machines. This has led to an explosion in the use of clusters of computers, which account for one of the leading tendencies among the current trends in parallel processing [34, 158]. Basically, a cluster is a collection of PCs or workstations connected via a network and using off-the-shelf components. The significant improvements in performance and reliability of workstations and high-speed networks strongly contributed to turn the use of clusters into an appealing, low-cost alternative for parallel applications. Clusters can be put together very easily, using operating systems such as public domain Linux, networks such as Ethernet, SCI, or Myrinet, and software environments such as MPI or PVM. Some cluster systems are currently among those with the best cost/performance ratios, ranking among the fastest and most powerful machines.

13.2.2 Parallel programming environments

The architecture of a parallel machine has a strong influence in the degree of parallelism or *granularity* of the applications it can run, i.e., the ratio between computation time and communication time. The granularity may also be seen as the amount of computation performed between two communication steps. If

the architecture of the target machine allows for fast communication (high speed network or shared memory), then fine-grained applications are more suitable to be parallelized; otherwise medium- or coarse-grained applications should be envisioned for parallel implementation.

Parallelism leads to the need for new algorithms designed to exploit concurrency, because many times the best parallel strategy cannot be achieved by just adapting a sequential algorithm [85, 121]. In contrast to sequential programs, parallel algorithms are strongly dependent on the computer architecture for which they are designed. Programs implementing these algorithms are developed with parallel programming tools, which typically provide support for developing programs composed by several processes that exchange information during their execution. Many programming tools are available for the implementation of parallel programs, each of them being more suitable for some specific problem type or machine architecture. The choice of the programming tool to be used depends on the characteristics of the problem to be solved and should match the programming model underlying the parallel algorithm to be implemented. For instance, some tools might be more suitable for numerical algorithms based on regular domain decomposition, while others are more appropriate for applications that need dynamic spawning of tasks or that make use of irregular data structures.

There are basically three strategies for the development of parallel programs. The first one consists in using *parallel programming languages*, which are essentially sequential languages augmented by a set of special system calls. These calls provide low-level primitives for message passing, process synchronization, process creation, mutual exclusion, and other functions. Among them, we may cite HPF (High Performance Fortran) [105, 117, 118, 139, 140] and OpenMP [66], which explore inherent data parallelism (the same instruction set is applied to multiple items of a data structure) appearing for instance in do-loops. FORTRAN 90 provides constructs for specifying concurrent execution based on data parallelism. HPF extends FORTRAN 90 with additional parallel constructs and data placement directives. A data parallel program developed using both languages is a sequence of operations that are applied to some or all elements of an array. Besides inserting parallelisation directives, the programmer does not explicitly handle communication primitives. Parallelisation directives are activated along compilation. Such parallel programming languages are well adapted to fine-grained synchronous numerical applications.

In the second and currently most used strategy, tasks communicate by exchanging messages using *communication libraries* such as PVM (Parallel Virtual Machine) [104] or MPI (Message-Passing Interface) [85, 102, 103, 145, 146, 147, 206]. To increase efficiency and avoid context swapping, each processor usually runs one single process (from now on, we do not make any difference between a process and the processor where it runs) in charge of computations and communication. This programming mode is particularly suited to coarse-grained applications running on clusters or networks of workstations. PVM is a pioneer, widely used message passing library, created to support the develop-

ment of distributed and parallel programs executed on a set of interconnected heterogeneous machines. A PVM program consists of a set of tasks that communicate within a parallel virtual machine by exchanging messages. A configuration file created by the user defines the physical machines that comprise the virtual machine. The application programmer writes a parallel program by embedding these routines into a C, C++, or FORTRAN code. Several parallel implementations of metaheuristics have been developed in PVM, see e.g. [6, 9, 15, 20, 36, 58, 68, 81, 89, 151, 161, 162, 163, 164, 196, 197, 199, 200]. MPI is a proposal for the standardization of a message passing interface for distributed memory parallel machines, with the aim of enabling program portability among different parallel machines. It just defines a message passing programming interface, not a complete parallel programming environment. It does not provide any definition for fault tolerance support and assumes that the computing environment is reliable. Many implementations of the MPI standard are available, all based on a parallel virtual machine composed by several connected heterogeneous computers. Each of these computers executes a process used to control the exchange of messages among them. Again, several parallel implementations of metaheuristics have been developed using MPI, see e.g. [7, 23, 41, 76, 116, 132, 134, 136, 178, 183].

Finally, the increasing use of SMP clusters is leading to the generalized use of *programming with lightweight processes* such as POSIX threads [33, 110] or Java threads [150]. Threads are a general operating systems concept, not specifically related to parallel programming. However, due to their importance in providing support for concurrent programming, and to their extensive use in many of the tools discussed in other sections, their understanding is essential to a parallel programmer. Threads communicate using the global memory allocated to the associated processes. Programming with threads is specially useful in shared-memory machines, since in this case the global variables model the shared memory to which all processors have access. Smaller creation times and context swapping times lead to an improved superposition of communication and computation tasks along the execution time. The use of threads is particularly suited to medium- and coarse-grained parallel applications. Recently, the concept of threads has been extended to distributed-memory machines with programming tools such as Cilk [167, 190], Charm++/Converse [30, 111, 112, 113], Athapascan [49], PM2 [159], and Java threads [128], among others.

13.2.3 Types of parallelism and parallel programming models

There are two basic types of parallelism that can be explored in the development of parallel programs: data parallelism and functional parallelism [85, 121, 135, 144]. In the case of *data parallelism*, the same instruction set is applied to multiple items of a data structure. This type of parallelism can be easily implemented in shared memory machines. Data locality is a major issue to be considered, regarding the efficiency of implementations in distributed memory machines. In the case of *functional parallelism*, the program is partitioned into cooperative tasks. Each task can execute a different code and all tasks can run

asynchronously. Data locality and the amount of processing within each task are important concerns for efficient implementations.

Independently from machine architecture and programming environment considerations, we may distinguish two main parallel programming models: centralized and distributed. In the *centralized model*, also called master-slave or client-server, data are either handled by a specialized processor (the master) or stored in a shared memory. In the first case, the slaves communicate with the master to get work to be done and to send results. The master is in charge of load balancing and may also eventually perform computations. In the second case, the processors obtain the work to be done from the shared memory, where they also store the results they obtain. The *distributed model* is characterized by the absence of global data, either shared or centralized. All data is local to each processor. Information is shared or made global by the exchange of messages among the processors. SPMD (*Single Program, Multiple Data*) is a model used for parallel programming on MIMD machines, based on the distributed programming model, where the same code is executed on different processors over distinct data. It has been widely used due to the ease of designing a program that consists of a single code running on different processors [137, 160, 166].

The choice of the appropriate programming model strongly depends on the architecture of the target parallel machine where the implementation will run. With the advent of clusters of SMP machines, we see an increase in the number of implementations based on a hybrid model, i.e., implementations using a centralized model within each SMP machine, running under a distributed model with respect to the machines in the cluster. We will see in the forthcoming sections that parallel implementations of metaheuristics are usually based on either one of these models.

13.3 SPEEDUP AND EFFICIENCY

Given some problem \mathcal{P} , a parallel algorithm \mathcal{A} , and a parallel machine \mathcal{M} with q identical processors, we denote by $T_{\mathcal{A}, \mathcal{M}}(p)$ the elapsed time taken by algorithm \mathcal{A} to solve problem \mathcal{P} on machine \mathcal{M} using $p \leq q$ processors. We denote by $T_{\mathcal{A}_s}$ the time taken by the best (i.e., the fastest) known sequential algorithm \mathcal{A}_s to solve the same problem \mathcal{P} on a sequential machine whose processor is equivalent to those in the parallel machine \mathcal{M} . We define the *speedup* of the parallel system defined by algorithm \mathcal{A} and machine \mathcal{M} when p processors are used as

$$s_{\mathcal{A}, \mathcal{M}}(p) = \frac{T_{\mathcal{A}_s}}{T_{\mathcal{A}, \mathcal{M}}(p)}.$$

We note that if the above definition is to be precise, then both the sequential algorithm \mathcal{A}_s and the parallel algorithm \mathcal{A} are supposed to always find exactly the same solution to problem \mathcal{P} . Although this will not necessarily be the case for most parallel implementations of metaheuristics, as discussed later in Section 13.4, this definition will also be used in this context on a less formal basis. We also notice that due to the hardness of establishing the best sequential

algorithm for general instances of some problem, the value of $T_{\mathcal{A}}$, in the above formula is often replaced and approximated by $T_{\mathcal{A},\mathcal{M}}(1)$, i.e., the time taken by the parallel algorithm \mathcal{A} using only one processor of the parallel machine \mathcal{M} .

The speedup measures the acceleration observed for the parallel algorithm running on p processors, with respect to the best sequential algorithm. The efficiency $\eta_{\mathcal{A},\mathcal{M}}(p)$ of the above parallel system is given by

$$\eta_{\mathcal{A},\mathcal{M}}(p) = \frac{s_{\mathcal{A},\mathcal{M}}(p)}{p}.$$

It measures the average fraction of the time along which each processor is effectively used. Ideal efficiency values are as much close to one as possible, i.e., each processor should be used for as much time as possible in effective computations. However, some anomalies characterized by efficiency values larger than one are observed in some problem instances, usually due to wrong decisions taken by the sequential algorithm (see Porto and Ribeiro [161] for some illustrative results in the case of a single-walk parallel tabu search implementation, as well as e.g. [123, 124, 130, 131] for results and discussions in the context of parallel branch-and-bound algorithms).

Several authors have recently addressed the efficiency of parallel implementations of metaheuristics based on running multiple copies of the same sequential algorithm, classified as independent multi-thread strategies in Section 13.4.2.1. A given target value τ for the objective function is broadcasted to all processors which independently execute the sequential algorithm. All processors halt immediately after one of them finds a solution with value at least as good as τ . The speedup is given by the ratio between the times needed to find a solution with value at least as good as τ , using respectively the sequential algorithm and the parallel implementation with p processors. Some care is needed to ensure that no two iterations start with identical random number generator seeds. These speedups are linear for many parallel implementations of metaheuristics reported in the literature, i.e., they are proportional to the number of processors. A typical example is described in [155], for a PVM implementation of a parallel GRASP for the MAX-SAT problem. This observation can be explained if the random variable *time to find a solution within some target value* is exponentially distributed, as indicated by the following proposition [203]:

Proposition 1: *Let $P_\rho(t)$ be the probability of not having found a given target solution value in t time units with ρ independent processes. If $P_1(t) = e^{-t/\lambda}$ with $\lambda \in \mathbb{R}^+$, corresponding to an exponential distribution, then $P_\rho(t) = e^{-\rho t/\lambda}$.*

This proposition follows from the definition of the exponential distribution. It implies that the probability $1 - e^{-\rho t/\lambda}$ of finding a solution within a given target value in time ρt with a sequential algorithm is equal to the probability of finding a solution at least as good as that in time t using ρ independent parallel processors. Hence, it is possible to achieve linear speedups in the time to find a solution within a target value by multiple independent processors. An analogous proposition can be stated for a two parameter (shifted) exponential distribution:

Proposition 2: Let $P_\rho(t)$ be the probability of not having found a given target solution value in t time units with ρ independent processors. If $P_1(t) = e^{-(t-\mu)/\lambda}$ with $\lambda \in \mathbb{R}^+$ and $\mu \in \mathbb{R}^+$, corresponding to a two parameter exponential distribution, then $P_\rho(t) = e^{-\rho(t-\mu)/\lambda}$.

Analogously, this proposition follows from the definition of the two-parameter exponential distribution. It implies that the probability of finding a solution within a given target value in time ρt with a sequential algorithm is equal to $1 - e^{-(\rho t - \mu)/\lambda}$, while the probability of finding a solution at least as good as that in time t using ρ independent parallel processors is $1 - e^{-\rho(t-\mu)/\lambda}$. If $\mu = 0$, then both probabilities are equal and correspond to the non-shifted exponential distribution. Furthermore, if $\rho\mu \ll \lambda$, then the two probabilities are approximately equal and it is possible to approximately achieve linear speedups in the time to find a solution within a target value using multiple independent processors [8].

This behavior has been noticed in a number of metaheuristics. These include simulated annealing [70, 152]; iterated local search algorithms for the traveling salesman problem [75], where it is shown that the probability of finding a sub-optimal solution is given by a shifted exponential distribution, allowing for the time to find the first local optimum; tabu search, provided that the search starts from a local optimum [24, 191]; and WalkSAT [186] on hard random 3-SAT problems [108]. Recently, Aiex et al. [8] have shown experimentally that the solution times for GRASP also have this property, showing that they fit a two-parameter exponential distribution. Figure 13.1 illustrates this result, depicting the superimposed empirical and theoretical distributions observed for one of the cases studied along the computational experiments reported by the authors, which involved GRASP procedures applied to 2400 instances of five different problems: maximum independent set [80, 171], quadratic assignment [129, 172], graph planarization [175, 177], maximum weighted satisfiability [173, 174], and maximum covering [170]. The same result still holds when GRASP is implemented in conjunction with a post-optimization path-relinking procedure [7].

13.4 PARALLELIZATION STRATEGIES

Even though parallelism is not yet systematically used to speed up or to improve the effectiveness of metaheuristics, parallel implementations abound in the literature. Parallelization strategies considerably differ from one technique to another. Although much of the initial efforts have been concentrated on the parallelization of simulated annealing [2, 19, 100], they were quickly followed by parallel implementations of genetic algorithms and tabu search. The first classification of parallel tabu search algorithms was proposed by Voss [205], based on the use of different search strategies and initial solutions. It was later generalized by Crainic et al. [60, 63], by taking into account additional aspects such as communication organization and information handling. We describe an architecture-independent classification considering the underlying parallel local

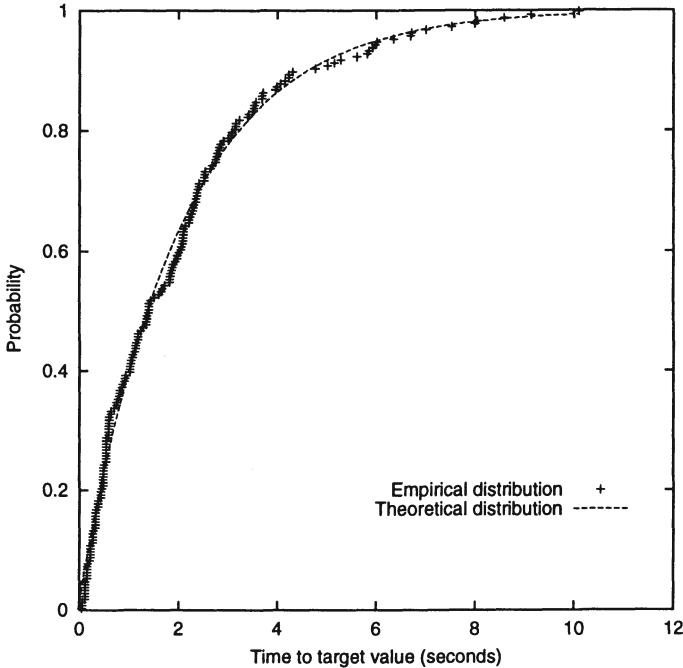


Figure 13.1 Superimposed empirical and theoretical distributions (times to target values measured in seconds on a SGI Challenge computer with 28 processors).

search strategy, inspired from that proposed in [203]. We also discuss in details some applications of these strategies. Due to the high number of applications, we have chosen a few examples for each strategy, among those we are more familiar with.

Metaheuristics based on local search may be seen as the exploration of the *neighborhood graph* (or state space graph) associated with a problem instance, in which nodes correspond to solutions and edges connect neighbor solutions. Each iteration consists basically in the evaluation of the solutions in the neighborhood of the current solution, followed by a move towards one of them, avoiding as much as possible to prematurely stop in a local optimum and until no further improvement of the best known solution can be achieved. The solutions visited along the search define a *walk* (or a *trajectory*) in the neighborhood graph. Parallel implementations of metaheuristics use several processors to concurrently generate or explore the neighborhood graph. Since this graph is not known beforehand, parallelizations of metaheuristics are *irregular applications*, whose efficiency strongly depends on the appropriate choice of the granularity of the algorithm and on the use of load balancing techniques.

We distinguish between two approaches for the parallelization of the local search, according with the number of trajectories investigated in the neighborhood graph:

- (1) Single walk: fine- to medium-grained tasks
- (2) Multiple walks: medium- to coarse-grained tasks
 - a. independent search threads
 - b. cooperative search threads

In the case of a *single-walk parallelization*, one unique trajectory is traversed in the neighborhood graph. The search for the best neighbor at each iteration is performed in parallel, either by the parallelization of the cost function evaluation or by domain decomposition (the neighborhood evaluation or the problem instance itself are decomposed and distributed over different processors). A *multiple-walk parallelization* is characterized by the investigation in parallel of multiple trajectories, each of which by a different processor. We call by a *search thread* the process running in each processor traversing a walk in the neighborhood graph. These threads can be either *independent* (the search threads do not exchange any information they collect) or *cooperative* (the information collected along each trajectory is disseminated and used by the other threads).

13.4.1 Single-walk parallelizations

The goal of this strategy is basically to speed up the sequential traversal of the neighborhood graph. The task whose execution is distributed among the processors may be the evaluation of the cost function at each neighbor of the current solution or the construction of the neighborhood itself. In the first case, speedups may be obtained without any modification in the trajectory followed by the sequential implementation. In the second case, the neighborhood decomposition and its distribution over several processors often allow more deeply examining a larger portion of the neighborhood than that examined by a sequential implementation, which often uses neighborhood reduction techniques. Thus, the trajectory followed in the neighborhood graph may be better guided in parallel than in sequential mode, possibly leading to improved solutions. This approach may also be useful whenever one wants to explore extended neighborhoods obtained by move composition techniques such as ejection chains [93, 98].

The idea of *distributed neighborhoods* may be used to formalize parallelization strategies based on domain decomposition. A strategy based on distributed neighborhoods consists of two parts. Each iteration of the local search starts by broadcasting the current solution to all processors, together with a domain decomposition which defines the local neighborhoods (i.e., the portions of the complete neighborhood which will be investigated by each processor). This decomposition is temporary and may change from an iteration to another. In the second part, each processor finds and proposes a move within its local neighborhood. These moves are combined or the best move found within the local neighborhoods is selected, so that a new feasible solution is generated. This process is repeated until the current solution cannot be further improved.

Single-walk parallelizations have small or medium granularity and need frequent synchronizations. They are extremely dependent on the application, in terms of neighborhood structure and cost function definition. Their first applications appeared in the context of simulated annealing and genetic algorithms. They preserve convergence properties and are used in most parallel implementations of simulated annealing [2, 19]. Although domain decomposition strategies do not always provide significant reductions in computation times, they are often used due to the need to investigate large neighborhoods.

A single-walk parallelization based on neighborhood decomposition of a tabu search algorithm for solving a task scheduling problem on heterogeneous processors was proposed and discussed by Porto and Ribeiro [161, 162, 163]. The objective function to be minimized is the makespan of the parallel application, i.e., the time needed for the last processor to halt. Each solution has $O(n^2)$ neighbors, which can be obtained by reassigning each of its tasks to any other processor. Due to the precedence constraints, the evaluation of the makespan of each solution has time complexity $O(n^2)$, where n is the number of tasks. As mentioned by Fiechter [83], synchronous parallelizations usually require extensive communication and therefore are only worth applying to problems in which the computations performed at each iteration are complex and time consuming. This is exactly the case for this problem, in which the search for the best move at each iteration is a computationally intensive task running in time $O(n^4)$. Thus, from this point of view, this sequential tabu search algorithm is suitable for parallelization. Supposing p processors are available, the neighborhood is partitioned by distributing to each processor $k = 1, \dots, p - 1$ the moves defined by the transfer of the tasks numbered from $(k - 1)[n/p] + 1$ to $(k - 1)[n/p] + [n/p]$, and to processor p the remaining ones. Some computational results obtained on an IBM SP-1 system for problems with up to 400 tasks are reported in [161]. Different PVM implementations, using either the master-slave or the SPMD parallel programming models, showed that efficiencies close to the peak unit value (i.e., speedups of the same order of the number of processors used in the parallel implementation) can be obtained for large problems with up to $p = 16$ processors. Additional results reported in [164] for larger task graphs modeling typical program structures (such as diamonds, divide-and-conquer, fast Fourier transform, Gauss decomposition, and partial differential equations) further illustrated the robustness of the best parallel strategy, which uses the master-slave model with a dynamic load balancing strategy.

Parallelizations of tabu search based on neighborhood decomposition and exhaustively searching for the best improving neighbor always find the same solution obtained by the sequential algorithm. They allow for efficient load balancing strategies, since the work to be done at each iteration is known beforehand and may be easily and evenly distributed among the processors, leading to efficient parallel implementations as that described in the above paragraph. However, if the parallelization involves a local search algorithm in which we seek the first improving move at each iteration, then load imbalance is more

likely to deteriorate the implementation (because in this case the overall computational effort associated with the investigation of the neighborhood is not known beforehand). On the other hand, better solutions than those obtained by the sequential algorithm can be found, since in this case the application of several local searches based on finding the best solution over subsets of the neighborhood may lead to a better solution than a single local search applied to the whole neighborhood.

In the case of vehicle routing problems, parallel implementations of heuristics based on domain decomposition, such as that proposed by Fiechter [83] for the traveling salesman problem, start by partitioning the clients into subsets or clusters (according with some criteria such as proximity measures or polar coordinates), under the rationale that they should be part of the same route. Different processors acting in parallel may construct sets of different routes appropriate for each subset of clients, using constructive algorithms or metaheuristics. After synchronization, these routes are combined to yield a full solution to the routing problem. Since the combination requirements strongly limit the search for partial solutions and the quality of the final solution, the latter can be further ameliorated by a diversification procedure running at a higher level.

Aarts and Verhoeven [4, 203] make the distinction between single-step and multiple-step parallelism within this class. In the case of *single-step* implementations, neighbors are searched and evaluated in parallel after neighborhood partitioning. The algorithm subsequently selects and performs one single move, as for the above described parallelization of tabu search for task scheduling. In *multiple-step* parallelizations, a sequence of consecutive moves in the neighborhood graph is made simultaneously.

13.4.2 Multiple-walk parallelizations

Most parallel implementations of metaheuristics other than simulated annealing follow multiple-walk strategies. Tasks executed in parallel have a larger grain. Besides the search for speedups, improvements in solution quality are also sought. The search threads can be independent or cooperative.

13.4.2.1 Independent search threads.

We distinguish between two basic approaches:

- Exploration in parallel of multiple trajectories originating from different nodes of the neighborhood graph: each walk starts from a different solution (or with a different population, in the case of population methods). The search threads may use the same local search algorithm or different ones, with the same parameter values or not. The trajectories may intersect at one or more nodes of the neighborhood graph. If p processors are used, this strategy corresponds to the successive execution of p sequential independent searches.

- Exploration in parallel of subgraphs of the neighborhood graph, obtained by problem decomposition (e.g. by variable fixation): several subgraphs of the neighborhood graph are explored in parallel, without intersection of the corresponding trajectories. We have a total and permanent decomposition of the neighborhood graph formed by complete solutions.

The first parallel implementations of tabu search based on this type of strategy seem to concern the quadratic assignment problem and job shop scheduling [191, 192]. A parallel implementation of reactive tabu search [25] applied to the Steiner problem in graphs appears in this book [23]. Each slave processor independently performs the reactive tabu search algorithm, with a different parameter setting and from a different initial solution generated by a randomized version of the shortest-path heuristic of Takahashi and Matsuyama [195]. The master processor also handles a pool of elite solutions, which is used by a post-optimization intensification procedure based on path-relinking [94, 96] once all processors have finished their searches. Computational experiments have been performed on a cluster of 32 Pentium II-400 processors, running under Linux and using the LAM implementation of MPI. The parallel implementation found optimal solutions for all OR-Library instances [26] in series C, and for respectively 18 and 14 out of the 20 instances in each of series D and E. The maximum and the average relative errors with respect to the optimum among all instances of this type were respectively 0.14% and 0.06%. For the incidence problems [72], this strategy found optimal solutions for all instances with 80 nodes, and for respectively 96 and 82 out of each set of 100 instances with 160 and 320 nodes. The maximum and the average relative errors with respect to the optimum among all instances of this type were respectively 3.91% and 0.07%. The comparison of the results obtained by the parallel implementation with those reported for the original sequential algorithm shows that the former found improved results for all problem series and sizes, in particular for the largest instances. The main advantage of the parallel implementation is due not only to solution improvements, but mainly to its robustness, since high quality solutions are obtained without almost any effort in parameter tuning.

Rego and Roucairol [168] developed a parallel implementation of an ejection chain procedure for the vehicle routing problem with bounds on vehicle capacities and route lengths. To build an ejection chain with k levels, we first choose k clients not consecutively visited in the current solution. The first client takes the place of the second, the second that of the third, and so on. The k -th client takes either the place of the first (circular permutation) or is inserted into the best possible position between two clients. The chain is initialized by the choice of the client in the first level. Next, one computes iteratively for k ranging from one to a maximum level k_{\max} the cost variation associated with the k -th level ejection chain and the best one is selected. This move is performed in the context of a tabu search procedure, which visits infeasible solutions using different strategic oscillation [95, 96] schemes. The authors implemented an independent-thread strategy within a centralized algorithm on a network of four Sun SPARC IPC workstations. Starting from the same initial solution

communicated by the master, each slave runs a tabu search algorithm with different parameters. This strategy is combined with two single-walk parallelizations. The first one is based on neighborhood decomposition. All possible closing moves of the ejection chain are evaluated in parallel by different processors, which search for the best position where the last client ejected from the chain should be inserted. The second one is a natural problem decomposition strategy based on a post-optimization procedure, which is activated whenever tabu search cannot improve the current solution after some number of iterations. Both single-walk parallelizations of the neighborhood search are quite effective and allow for approximately halving the computation time of the multiple independent-thread parallel algorithm. The latter, combining these different parallelization strategies, improved the best known solutions by 0.22% in the average, over 15 classical examples with up to 199 clients.

Typical parallelizations of GRASP make use of multiple independent threads. Basically, they consist in distributing over the p processors acting in parallel the total number of iterations to be performed. Since the GRASP iterations are completely independent and very few 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 with their demands, to improve load balancing. This strategy was applied first to the quadratic assignment problem [154] on a shared virtual memory KSR-1 machine. Parallel GRASP implementations of independent-thread strategies for the Steiner problem in graphs [134, 136] and for a matrix decomposition problem arising in the context of traffic assignment in satellite systems [13, 14, 165, 176] are reported farther in Section 13.5.2.

Parallelizations of genetic algorithms using multiple independent search threads correspond to the so called island model, without exchanges among the subpopulations in each island [31, 119]. This approach was also applied in the parallelization of a scatter search procedure for the quadratic assignment problem [65] on a small cluster of PCs. Different scatter search algorithms using different parameter settings are launched on the processors. Although some speedup has been observed, there was no improvement in the solutions obtained by the sequential algorithm. This is explained by the fact that small subpopulations within each processor freeze very soon, due to the absence of communication among the processors.

Multiple-walk strategies based on independent search threads can be very easily implemented. They lead to good speedups, provided the problem to be solved satisfies the conditions of Propositions 1 and 2 in Section 13.3. Very robust implementations can be obtained by using different parameter settings at each processor. Moreover, an appropriate partitioning of the search space allows for its good coverage and avoids redundant work. On the other hand, this model is quite poor and can be very easily simulated in sequential mode, by several successive executions with different initializations, as a multistart algorithm. The lack of cooperation between the search threads does not allow for the use of the information collected by different processors along their trajectories. Since

the trajectories may be quite long, load imbalance problems are very likely to occur. Redundant work may be done if the search space is not appropriately partitioned.

13.4.2.2 Cooperative search threads. This is the most general and also the most promising type of strategy, demanding more programming efforts and implementation skills. The search threads exchange and share information collected along the trajectories they investigate. This shared information is implemented either as global variables stored in a shared memory, or as a pool in the local memory of a dedicated central processor which can be accessed by all other processors. In this model, in which the search threads cooperate and the information collected along each trajectory is used to improve the other trajectories, one expects not only to speed up the convergence to the best solution but, also, to find better solutions than independent-thread strategies within the same computation time. 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. We may cite elite solutions and their costs, best solutions found, move frequencies, tabu lists, and population sizes, among others. This data can give a broader view of the search space and may be used to drive diversification and intensification procedures.

The multicommodity location-allocation problem with balancing requirements [61] was one of the first applications addressed by cooperative-thread implementations of tabu search. The algorithm is centralized, as in most parallelizations of this type. The trajectories start from different initial solutions and communicate through a central pool of elite solutions. In a similar domain, another application to the design of capacitated multi-resource networks is described in [59]. Improved solutions and smaller computation times are obtained for both applications, with respect to the sequential algorithms.

The same approach was used by Aiex et al. [6] in the parallelization of a tabu search algorithm for circuit partitioning, in the context of the pseudo-exhaustive logical test of VLSI combinational circuits. Each search thread runs a variant of the original sequential tabu search algorithm developed for this problem [16], using different initial solutions and move attributes. The search threads communicate through a central pool of elite solutions, implemented on a dedicated master processor which handles pool management. An elite solution is defined as a local optimum which improves the best solution already visited by the search trajectory in the same processor. Each processor performing the search sends to the pool the elite solutions it has found. A newly received solution is inserted into the pool or not, according with some criteria based on the quality of the solutions already in the pool. A processor requests a solution from the pool to restart its search whenever it is not able to improve its best solution after some number of iterations. Numerical results using nine search processors on an IBM SP-2 machine are reported for benchmark circuits with up to 207 inputs, 3405 gates, 140 outputs, and 7552 links. Significant

reductions of up to 30% in the number of circuits in the partition illustrate the effectiveness of the parallel tabu search procedure. Comparative results illustrating the efficiency of implementations in PVM and Linda [43, 50, 185] are also discussed.

Parallelizations of GRASP using multiple cooperative search threads may be implemented with a similar use of a pool of elite solutions and are described in Section 13.5.2. The search threads cooperate through a path-relinking procedure [94, 96, 97], combining elite solutions from the pool with the local optima found at the end of each iteration (see also [7, 41, 42] for details).

Software packages such as EBSA [86] and ASA [109] with implementations of simulated annealing may run on parallel machines. ParSA is a library developed in C++ [116], containing multiple-walk strategies that allow the exchange of some statistics. Some numerical results obtained for a scheduling problem in aerial transportation with 4,000 flight segments and 130 aircrafts of ten different types lead to improved solutions and speedups of 5.4 on eight nodes of an ATM cluster and of 17.5 on a GC/PowerPlus machine with 32 nodes.

Cooperative strategies are also the most successful ones for the parallel implementation of genetic algorithms, in terms of both the number of applications and the improvements in solution quality. Several libraries are also available, such as POOGAL [31] and PGAPack [17]. This type of parallelization is defined by an additional operator of *migration*, which establishes the policy for solution exchanges: processors involved in the exchanges, frequency of exchanges, and solutions exchanged, among other information. Most cooperative parallel implementations of genetic algorithms are based on the island model with communication. Each processor has its own subpopulation and solutions are exchanged with neighbor processors according with its migration policy [31, 60]. Another variant consists for each solution to select a mate in another processor within a specified neighborhood to perform a crossover. The selection strategy may even involve another heuristic in the determination of the best possible mate. Bubak and Sowa [31] used the island model in the implementation of a parallel procedure for the traveling salesman problem on an HP/Convex Exemplar SPP1600 machine with 16 processors and on a heterogeneous cluster formed by Hewlett Packard (D-370/2 and 712/60) and IBM (RS6000/520 and RS6000/320) machines.

Since ant system optimization can be viewed as the application of an iterated greedy randomized algorithm, parallelization strategies are similar to those of GRASP. Cooperation is guided by the pheromone trails. Each thread handles one ant and the ants communicate through the pheromone trails. Typically, the ants are evenly distributed over the processors. Since each ant acts independently of the others, linear speedups can be obtained. In practice, however, the communication incurred by the management of the pheromone trails as global information is an important overhead. Since all ants use and update the pheromone trails, access to the latter is clearly the key point for efficient parallel implementations. Taillard [193] dedicated the management of the pheromone trails to a *queen* process, which is a very convenient way to

implement a parallel ant system following a master-slave scheme [199, 200]. The bottleneck at the master can be dealt with by a hierarchical approach (i.e., a hierarchy of masters) or by a colony approach similar to the island model in genetic algorithms. Each colony is handled by a master processor, who manages its pheromone trails. The information exchanged between the colonies can be either ants finding good solutions or parts of the local pheromone trails to influence the searches performed by the other colonies.

13.5 APPLICATIONS

We review in this section parallel implementations of algorithms derived from tabu search, GRASP, genetic algorithms, simulated annealing, and ant colonies. This review is not intended to be either exhaustive, or comprehensive, due to the enormous amount of work and publications in the area in recent times. Instead, we try to give a broad view of applications and implementation studies of parallel metaheuristics, referring the reader to the original references for additional details and publications.

13.5.1 Tabu search

Tabu search is an adaptive procedure for solving combinatorial optimization problems, which guides an improvement heuristic to continue exploration without being confounded by an absence of improving moves (see e.g. [91, 92, 96]). Tabu search goes beyond local search by employing a memory-based strategy for modifying the neighborhood of the current solution as the search progresses. Depending on the search stage, this modified neighborhood may be an extension or a restriction of the regular one. As a consequence, and contrary to the basic local search scheme, moves towards solutions which deteriorate the cost function are permitted under certain conditions.

Single-walk parallelizations of tabu search based on neighborhood decomposition are reported in [51, 52, 62, 87, 161, 162, 163, 164, 191]. An interesting application in chemistry to determine the best energy minimum for oligopeptides is reported in [143]. Different systems such as the Connection Machine CM-2, a network of transputers, an IBM SP-1 machine, a SGI Origin 2000, a Meiko computer with 16 T-800 transputers, and a network of Sun SPARC workstations have been used for these implementations. Computation times are smaller than those of their sequential counterparts and almost-linear speedups are often obtained, but only minor improvements in solution quality are observed for some instances. A parallel tabu search algorithm for the traveling salesman problem based on domain decomposition [83] was already discussed in Section 13.4.1.

Badeau et al. [20] implemented a parallel algorithm for vehicle routing with time windows, combining single- and multiple-walk strategies. First, the master processor distributes the problem data to slaves running initialization processes. Each of these slave processors builds a different initial solution and returns it to the master. The master combines the solutions (i.e., the routes) it received,

so as to generate a unique initial solution (formed by a set of routes) which is sent to slaves running decomposition processes. Each of the latter decomposes its current solution into several subproblems, which are then solved by variants of the same sequential tabu search algorithm [194]. The solutions to the subproblems are merged into a complete solution and sent to the master, which combines the new routes to obtain an improved solution. A new iteration of this procedure resumes, until some stopping criterion is attained. Computational results for an implementation on a network of Sun SPARC 5 workstations are reported. Gendreau et al. [89] implemented a parallel tabu search algorithm for real-time vehicle routing and dispatching following a similar master-slave scheme. The master processor manages the adaptive memory, decomposes the original problem into subproblems, and creates new starting solutions for the slaves. The latter apply tabu search to improve their initial solutions. Although the threads run independently, they communicate implicitly through the adaptive memory, which stores good solutions and distributes them as new starting solutions. Computational experiments have been performed on a network of 17 Sun UltraSPARC workstations, with communications between the processes being handled by PVM. As for the first application described in this paragraph, the authors report that the parallel tabu search implementation is competitive with the original sequential algorithm, in terms of solution quality for the same amount of computational work, however with much smaller elapsed times. A multiple-walk tabu search implementation for job-shop scheduling using independent search threads is reported in [192]. Talbi et al. [196] described an independent-thread multiple-walk parallel implementation of tabu search applied to the solution of the quadratic assignment problem on a network of heterogeneous workstations, based on the idea of adaptive parallelism to take into account and explore variations in machine load and availability.

A multiple-walk parallel implementation of an ejection chain procedure [168] for a vehicle routing problem with bounds on vehicle capacities and route lengths was already described in Section 13.4.2.1. Another parallel implementation of a tabu search algorithm for vehicle routing is reported in [184]. The neighborhood structure is based on simple customer shifts. All routes generated by independent search threads running a tabu search heuristic are collected in a distributed pool. New initial solutions are periodically obtained by a fast set covering heuristic applied to the routes in the pool. Computational results on a Parsytec CC system with eight Motorola PowerPC 604 nodes are reported. We have already reported in Setion 13.4.2.1 the results obtained by Bastos and Ribeiro [23] with an independent-thread multiple-walk parallel reactive tabu search strategy, which is among the most effective and robust heuristics for the Steiner problem in graphs. In the last two cases, we notice that although the threads are independent, they cooperate either periodically or at a post-optimization phase using a pool of good solutions.

Cooperative multiple-walk implementations are among the most promising tabu search parallelization strategies, due to their effectiveness and robustness. In most implementations, the processors start from different initial solutions

and their searches follow paths generated by different sequential tabu search algorithms. Communication is performed exclusively through a central pool of elite solutions. Crainic et al. [61] implemented several parallelizations of tabu search for the multicommodity location-allocation problem with balancing requirements and discussed the impact on performance and solution quality of several parameters. Crainic and Gendreau [59] have also implemented a cooperative multiple-walk parallel tabu search for the fixed charge, capacitated, multicommodity network design problem, which is shown to outperform both the sequential algorithm and independent-thread multiple-walk strategies. A similar approach was used by Aiex et al. [6] in the parallelization of a tabu search algorithm for circuit partitioning, as described in Section 13.4.2.2. The same strategy was recently applied in the solution of a labor constrained scheduling problem on a Sun SPARC 1000 workstation with eight processors [46], which significantly improved not only the solutions found by different sequential algorithms [47, 48], but also those obtained by a parallel asynchronous team [201] for most of the benchmark instances. Another possibility for cooperation could also consist in applying path-relinking to solutions newly found by the search threads, using those in the pool of elite solutions as targets, as in some parallel implementations of GRASP [7, 41, 42] described in Section 13.5.2.

13.5.2 GRASP

A greedy randomized adaptive search procedure (GRASP) [78, 79, 82] is a multi-start algorithm, in which each iteration consists of two phases. In a construction phase, a feasible solution is iteratively constructed, one element at a time, by a greedy randomized procedure. Next, in a local search phase, a local optimum in the neighborhood of the constructed solution is sought. The best solution over all iterations is kept as the result.

A multiple-step single-walk parallelization of a GRASP for the maximum independent set problem is described in [80], based on the decomposition of the solution space. The problem to be solved is divided into several subproblems, which are distributed over the processors. Each processor performs a sequential GRASP on a different subproblem, defined by conditioning the vertices appearing in the solution. Almost-linear speedups are reported.

Most parallel implementations of GRASP are independent-thread multiple-walk strategies, based on the distribution of the iterations over the processors. In general, each search thread has to perform $\text{Max_Iterations}/p$ iterations, where p and Max_Iterations are, respectively, the number of processors and the total number of iterations. Each processor has a copy of the sequential algorithm and a copy of the problem data. So as to avoid that the processors find the same solutions, each of them must use a different sequence of pseudorandom numbers. One of the processors acts as the master, reading and distributing problem data, generating the seeds which will be used by the pseudorandom number generators at each processor, distributing the iterations, and collecting the best solution found by each processor.

The parallelization of the GRASP developed by Prais and Ribeiro [165] for the problem of traffic assignment in communication satellites is discussed in [13]. The author also described the implementation of another independent multiple-walk strategy, in which each processor uses a different parameter value. Li et al. [129] reported almost-linear speedups of approximately 62 using 64 processors for a parallel GRASP for quadratic assignment. Other parallel implementations are described e.g. in [149, 154, 155].

Martins et al. [134] implemented a parallel GRASP for the Steiner problem in graphs. The construction phase is based on a probabilistic version of the distance network heuristic [138]. A hybrid local search procedure is applied only to new, previously unexplored solutions. First, a local optimum with respect to a neighborhood defined by insertions and eliminations of Steiner nodes is sought. This solution is then submitted to another local search procedure, based on the exchange of key-paths (i.e., paths connecting terminals or Steiner nodes with degree larger than two; see also [133, 136, 204] for additional details). Parallelization is achieved by the distribution of 512 iterations over the processors, with the value of the parameter α randomly chosen in the interval $[0.0, 0.3]$ at each iteration. The algorithm was implemented in C on an IBM SP-2 machine with 32 processors, using the MPI library for communication. The 60 problems from series C, D, and E from the OR-Library [26] have been used for the computational experiments, with respectively 500, 1000, and 2500 nodes. The parallel implementation obtained 45 optimal solutions over the 60 test instances and the relative deviation with respect to the optimal value was never larger than 4%. Average elapsed times in seconds (measured in exclusive mode, i.e., with no other application simultaneously running in the same machine) for both the sequential and parallel versions are reported in Table 13.1, while the speedups obtained with 2, 4, 8, and 16 processors are illustrated in Figure 13.2.

Series	Processors (times in seconds)				
	1	2	4	8	16
C	32.77	17.77	10.12	6.10	4.08
D	154.12	85.19	50.36	31.68	22.08
E	1379.90	828.19	485.97	307.44	216.86

Table 13.1 Elapsed times for 2, 4, 8, and 16 processors (512 iterations).

Alvim and Ribeiro [13, 14] have shown that multiple-walk independent-thread approaches for parallelization may benefit a lot from load balance 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 p processors in $q \geq p$ packets. Each processor starts performing one packet of $\lceil \text{Max_Iterations}/q \rceil$ iterations. Each slave processor informs the master when it

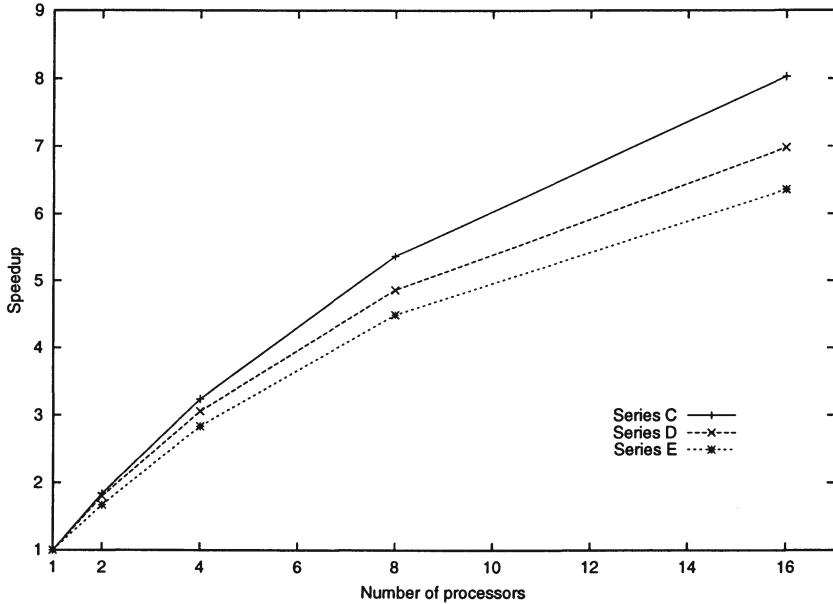


Figure 13.2 Average speedups on 2, 4, 8, and 16 processors.

finishes its packet of iterations. The master stops the execution of each slave 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 implemented for the problem of traffic assignment, this dynamic load balancing strategy allowed reductions in 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. The same approach was used in [136] for a preliminary parallel implementation of the above described GRASP for the Steiner problem in graphs.

Canuto et al. [41, 42] used path-relinking as a post-optimization step to implement a multiple-walk parallel GRASP algorithm for the prize-collecting Steiner tree problem. A similar approach was recently adopted by Aiex et al. [7] for the 3-index assignment problem. In both cases, the first phase of the parallel implementation is an independent multiple-walk strategy. Each processor, upon completing its iterations, applies path-relinking to pairs of elite solutions stored in a pool. In the second case [7], the parallel implementation globally follows an independent-thread strategy, since each processor keeps its own local pool of elite solutions. However, the strategy used in the first case [41] leads to a cooperative procedure, since pairs of elite solutions from a centralized unique central pool are distributed to the processors which perform path-relinking in parallel. Computational results obtained with implementations using MPI and running on a cluster of 32 Pentium II-400 processors [41] and on a SGI

Challenge computer with 28 196-MHz MIPS R10000 processors [7] show linear speedups and illustrate the effectiveness of path-relinking procedures used in conjunction with GRASP to improve the quality of the solutions found by the latter.

13.5.3 Genetic algorithms

A genetic algorithm is a basic search procedure, proposed and developed by Holland [106, 107]. It is based on the process of natural evolution, following the principles of natural selection and survival of the fittest [141, 169]. First, a population of individuals (solutions) is created. Each individual is evaluated according to a fitness function. Individuals with higher evaluations (i.e., more fitted) are selected to generate a new generation of this population. Crossover is a genetic operation which combines parts from pairs of individuals to generate new ones. Mutation is a unary genetic transformation that creates a new individual by small changes applied to an existing one. New individuals created by crossover or mutation replace all or part of the initial population. The process of evaluating fitness and creating a new population generation is repeated until a termination criterion is achieved.

Many different approaches to the parallelization of genetic algorithms can be found in the literature and they differ in many aspects. Basically, there are three types of parallelization strategies [35, 37, 40, 56]: global (single-walk parallelization), diffusion model, and island model (also called migration model by some authors). Fine-grained (diffusion model) and coarse-grained (island model) parallelizations characterize cooperative multiple-walk strategies.

Global (single-walk) parallelization strategies manipulate a single population, but the individuals are distributed among several processors for fitness evaluation. The trajectory followed by the parallel implementation is the same followed by the sequential algorithm. Almost-linear speedups may be obtained whenever the evaluation of the objective function is significantly more computationally expensive than the other steps (i.e., the genetic operators) of the algorithm. This approach was used by Chalermwat et al. [53] in the context of an application to image registration, in which parallelism is exploited for fitness evaluation. The authors reported very accurate registration results for LandSat/Thematic Mapper images and the parallel algorithm seemed to scale quite well on a 50-node Beowulf parallel cluster of Pentium Pro 200 MHz processors with a fat-tree network topology. Abramson and Abela [5] implemented a global master-slave parallelization to solve a school timetabling problem on a shared-memory Encore Max computer with 16 processors, in which the master distributes the individuals to be evaluated by each slave processor. The master also performs crossover and mutation operations to obtain a new generation. They reported limited speedups, due to critical sections of the code not being parallelized. Grefenstette [101] presented a master-slave parallel genetic algorithm for a system that learns strategies. The master maintains a population of competing strategies. At each iteration it generates a new population of strategies that are evaluated by the workers. Each worker evaluates a strategy by

simulating the execution of the assigned strategy in a determined task environment. Observed speedups were closer to linear for tasks requiring larger times for evaluating the strategies. Mühlenbein [148] developed a parallel genetic algorithm where each individual performs hill-climbing and selects a partner for mating within its neighborhood. The author claimed better or comparable solutions for very large instances of both the traveling salesman problem and the graph partitioning problem, with respect to those found by other heuristics such as multi-start hill-climbing and iterated hill-climbing.

Fine-grained parallelizations using the *diffusion model* are developed to take advantage of massively parallel computers. The population is divided into many small subpopulations, usually formed by a single individual. Each subpopulation is mapped onto the processing elements. Selection and mating are only possible within a small, restricted neighborhood defined by the topology of the massively parallel computer. The use of local selection and reproduction rules leads to a continuous diffusion of individuals over the population [56]. Talbi and Muntean [198] proposed and implemented a fine-grained algorithm on a T-800 transputer to solve a task mapping problem. Near-linear speedups have been observed and solution quality improved when the size of population was increased. Hill-climbing and a parallel simulated annealing algorithm were also implemented to be compared with the parallel genetic algorithm. The latter outperformed hill-climbing in terms of solution quality, with similar execution times. Simulated annealing obtained similar results concerning solution quality, but much greater elapsed times than the genetic algorithm. Kohlmorgern et al. [119] presented some results concerning fine-grained parallel versions of both the island and the diffusion models, applied to four problems: traveling salesman, flow shop scheduling, scheduling with resource constraints, and uncapacitated facility location. The initial population was partitioned into 1, 4, 16, 64, 256, and 1024 islands on a MasPar MP-1 parallel machine with 16 K processors organized as a two-dimensional grid, one individual per processor. In the island model implementation, the processing elements were partitioned into arrays forming the islands. Solutions are exchanged with neighbor islands in one, two, three, or four directions, after every 15, 30, or 50 generations. The authors noticed that using a high number of islands leads in general to better solutions, but with the burden of slower convergence. Good speedups in terms of the number of processors have been observed. They have also implemented another procedure, based on considering neighbors at different distances in the eight possible directions of the MasPar MP-1's X-net. The elitist strategy, where the very best neighbor is always selected, found the best solutions.

Coarse-grained or *island model* algorithms are based on dividing the population into a few subpopulations. Each subpopulation is assigned to a different processor, which performs a sequential genetic algorithm starting from this subpopulation. Mating involves only individuals within the same subpopulation. Occasionally, the processors exchange individuals through a migration operator. Some important technical issues influence the implementation of this strategy, such as the size of the subpopulations, the topology of the interprocessor

connection network that defines the exchange of individuals, and the migration rate (i.e., the number of individuals to be exchanged during a migration phase and the frequency in which migration takes place), see e.g. [35, 36, 157]. Some libraries to support the development of parallel genetic algorithms based on the island model were already cited in Section 13.4.2.2. Bubak and Sowa [31] developed an implementation for the traveling salesman problem using the island model, as described in Section 13.4.2.2. Belding [27] extended previous work on distributed genetic algorithms [202], focusing migration rates and intervals. A coarse-grained algorithm was developed and tested using the Royal Road class of fitness functions. For the easiest functions, the sequential algorithm performed better, while for the hardest functions the parallel algorithm was able to find better solutions. Levine [127] implemented a genetic algorithm based on the island model for solving the set partitioning problem. Tests were carried out on 128 nodes of an IBM SP-1 and showed that additional subpopulations, which increase the global population size, contributed to improve solution quality. Andre and Koza [15] described a coarse-grained parallel implementation in C and PVM of a genetic algorithm on a network of transputers. Experiments were made considering the problem of symbolic regression of the Boolean even-5-parity function and using different migration rates. Parallelization delivered super-linear speedups and best results in terms of computation times have been observed with an 8% migration rate. Another coarse-grained parallel implementation of a genetic algorithm is described by Falco et al. [68], concerning transonic airfoil optimization. Solutions are exchanged at the end of each iteration. The algorithm was implemented using C and PVM. Computational experiments were performed on a Convex Meta Series machine. Innovative designs of shockless airfoils were obtained by this algorithm, with the authors reporting convergence within much lower times than the sequential algorithms.

Cantú-Paz [36] described the implementation of two different parallel genetic algorithms. The first uses a master-slave strategy, while the other uses a coarse-grained model. The code was written in C++ and the PVM library was used for communication. The computational experiments were processed on a network of workstations and the tests were performed using several fitness functions with different degrees of hardness of the evaluation function. For the master-slave implementation, the results showed that the efficiency of the parallel implementation increases with the hardness of the evaluation function (as expected, since coarse-grained applications are more suitable to overcome the communication overhead introduced by the master-slave scheme). The same set of tests used for the coarse-grained algorithm showed that there are not significant gains in terms of speedup when the subpopulations are completely isolated. They also showed that, when each process is fully connected and migrations of individuals can occur between any pair of processors, the speedup increases with the hardness of the evaluation function. The reason is that the reduction in computation time brought by the use of multiple subpopulations is not enough to overcome the increase in communication time for easy-to-compute functions. The numerical results obtained for the coarse-grained

implementation are quite similar to the theoretical ones developed in [39] for predicting speedups of idealized bounding cases of parallel genetic algorithms.

Some hybrid implementations use combinations of multiple subpopulations with global or fine-grained methods. They form a hierarchy where the lower level consists of a global or fine-grained parallel genetic algorithm that manipulates the subpopulations, while a higher level implements a coarse-grained algorithm that distributes subpopulations to the lower level threads and controls solution migration among them. Bianchini and Brown [28] implemented four parallelization strategies of a genetic algorithm for solving 0-1 integer programming problems: centralized, semi-distributed, distributed, and totally distributed. These implementations were tested on a distributed memory environment consisting of a Transputer with eight 25 MHz T-805 processors, connected by 20 Mbits links. In the centralized method, the master sends some individuals to slave processors, that compute a certain number of generations and send their results back to the master, which then executes the replication algorithm for the whole population. The semi-distributed method consists of clusters of processors working with the centralized method and exchanging solutions among them. The distributed method is the traditional coarse-grained strategy, where each processor has its own subpopulation and exchanges of the best individuals occur from time to time. Finally, the totally distributed implementation consists of the distributed method without any exchange of individuals, each processor executing its own sequential algorithm over a subpopulation without communication. The authors compared these four strategies among them and with a sequential algorithm. All strategies presented similar results in terms of solution quality, with the centralized and semi-distributed ones achieving slightly better results. The speedup achieved by the totally distributed strategy is almost linear, while the centralized and semi-distributed implementations showed smaller speedups, due to bottlenecks created by the master process.

Oussaidène et al. [153] presented a hybrid parallel implementation of a genetic algorithm, in which multiple master-slave instances are associated with the subpopulations. A subpopulation is assigned to each master. The individuals are distributed over the slaves for the computation of their fitness values. The masters can exchange individuals from their subpopulations from time to time. Computational experiments with a parallel genetic algorithm code written in Java are reported in [90] for evolving pharmaceutical drug molecules and digital circuits. This code runs under a batch system called Condor, which manages the resources of a network of workstations and assigns jobs for each workstation according to its load. The initial experiments involved 31 jobs running independently, using the same input parameters but different random seeds. The results for evolving drug molecules were fairly good, but only trivial digital circuits were evolved.

The effect on the performance of different parallelization schemes of genetic algorithms induced by variations of parameters such as the number of slaves, population size, mutation rate, and mutation interval has been addressed by several authors, see e.g. [10, 11, 37, 40, 187] among others.

13.5.4 Simulated annealing

Simulated annealing in the context of optimization methods was introduced by Kirkpatrick et al. [115], see e.g. [3]. As the first metaheuristic to be parallelized, several papers describing parallelization efforts appeared subsequently [1, 21, 22, 44, 45, 67, 77, 182, 208], most of them focusing on cell placement problems in VLSI layout and on the convergence property proved by Geman and Geman [88]. Tools such as ProperPLACE [114, 156] have been designed to solve these problems with parallel implementations of simulated annealing.

Most of the first parallelizations followed the single-walk strategy and were based on the ideas of *move acceleration* and *parallel moves*. In the first case, each move is evaluated by breaking it up into several subtasks executed in parallel: selection of a feasible move, evaluation of the associated cost change, acceptance or rejection of the move, and global information update. Kravitz and Rutenbar [120, 182] described one of the first examples of this type of parallelization of the simulated annealing metaheuristic. In the case of parallel moves, each processor generates, evaluates, and accepts (or not) a different move. Synchronization could be done after only one or after several steps. However, due to the moves made by other processors, the cost function evaluation is not error-free regarding the sequential execution. To overcome the risk of inconsistencies, two approaches are proposed. The first considers only noninteracting moves (see also [21, 22, 120, 182]), for example by using a master-slave scheme in which the master monitors the cooling schedule by synchronizing the slaves at each step and chooses the accepted move [180]. In the second case, interacting moves are accepted and some errors in the evaluation of the cost function are allowed [44, 45, 179]. This approach is generally used in the framework of domain decomposition. In the case of the cell placement problem, this corresponds to partitioning the cells into subsets, handling each subset by one processor, and restricting moves in the same subset. This decomposition framework is also used for the traveling salesman problem in [12, 77]. Other authors have addressed the impact of errors on the convergence properties, see e.g. [19, 73, 74, 100]. This second approach can be viewed as a strong diversification process with respect to the sequential execution, which could explain the better quality of the results so obtained. Since the synchronization strategy can be performed more loosely between the processors, the reported speedups are better than those attained with the first, error-free synchronous approach. A variant of the single-walk strategy was proposed in [207] for the task assignment problem. The basic idea consists in doing speculative computations by evaluating in parallel by two threads both possibilities, acceptance or rejection, before the decision on the previous move has been completed. This approach does not seem to yield very promising results.

Other coarse-grained parallelizations of simulated annealing are based on the use of multiple Markov chains, which is equivalent to the multiple-walk strategy. Unlike the previously described parallelizations, each chain is allowed to perform cell moves on the whole set of cells, and not only within subsets. The synchronous approach was first introduced by Aarts et al. [1]. Let I denotes

the number of iterations (i.e., the length of the Markov chain) executed by a simulated annealing process before reaching equilibrium at temperature T . The parallel strategy consists in executing I/p iterations at temperature T by each of the p threads, followed by the synchronization of all threads to choose the best solution for the next temperature. However, this parallel process presents two main drawbacks: the communication overhead to exchange global information at each synchronization point and the lack of convergence properties, since the length of the Markov chain is arbitrarily reduced. Lee and Lee [122, 125, 126] presented some refinements to this approach, by introducing asynchronous strategies for the graph partitioning problem. The interaction between threads executing the traversal of the multiple walks is clearly the key point to efficiency in this approach.

Another implementation of the above approach, based on the execution of several cooperative threads running simulated annealing at different temperatures of the sequential cooling schedule, was proposed by Miki et al. [142]. Solutions are randomly exchanged between two consecutive temperatures of the cooling schedule. Applied to the minimization of a standard test function from a continuous optimization problem, the results reported are better than those obtained by the sequential algorithm in terms of both solution quality and computation time (speedup of approximately four in an 8-node PC cluster). However, this approach does not seem to outperform by too much the previous multiple-walks approaches. An empirical comparison of all these approaches appears in [54, 55]. The readers are also referred to [19, 60, 100] for other surveys about parallel simulated annealing and its hybrids.

13.5.5 Ant colonies

The ant system introduced by Colorni et al. [57] is presented as yet another evolutionary method inspired from nature. However, an ant system applied to some combinatorial optimization problem can be viewed as an iterated greedy randomized algorithm, guided by the pheromone trails. Indeed, each greedy process corresponds to an ant. The pheromone trails are used by the ants to bias the choice at each step of the greedy randomized algorithm. Ant systems differ from each other by the different strategies they use to handle the pheromone trails (updates and evaporation) [57, 71].

Bolondi and Bondanza [29] and Bullnheimer et al. [32] proposed two very similar and strongly synchronized parallel ant systems applied to the traveling salesman problem. At each parallel iteration, a set of ants is generated by the queen process and each ant explores a tour in parallel. Once every ant has finished its exploration, they are synchronized and the pheromone trails are updated by the queen process with the best tour they have found. Since the amount of computations performed by each ant is quite small, the ants spent most of their time in synchronizations with the queen process. For this reason, the authors also proposed in [32] a less synchronized parallel ant system to reduce the synchronization time. The underlying idea consists in implementing a 2-level hierarchical approach. Each of many first-level masters handles in

parallel but locally its own ant colony and pheromone trails, using a sequential algorithm. After a given number of sequential iterations, the first-level masters are synchronized by a second-level master process to globally update the pheromone trails. Since synchronizations are mainly done at the second-level master, this strategy showed less synchronization overhead than the previous one. The local/global iteration ratio is crucial for this strategy, since load imbalance is very likely to occur between the different first-level masters if this ratio is not correctly tuned.

Stützle [188] proposed a very simple parallel implementation, based on parallel independent runs of the MAX-MIN ant system. The main interest of this work is to check the property of Proposition 1, announced in Section 13.3. Since the execution time taken by the sequential algorithm for solving the traveling salesman problem seems to follow an exponential distribution, the relatively good empirical results obtained in terms of speedups and average solution quality seem to confirm the prediction of the proposition.

The quadratic assignment problem is yet another application of parallel ant systems. Talbi et al. [199, 200] applied the master-slave approach to their ANTabu system. The master handles the pheromone matrix and the best solution found. At each iteration, the master broadcasts the pheromone matrix to all the slave ants. Each slave constructs a complete solution, improves it using tabu search, and sends the improved solution back to the master. Computational experiments on a network of ten SGI Indy workstations running PVM are reported.

In addition to these applications to the more common traveling salesman and quadratic assignment problems, we mention that ant systems have also been adapted to the distributed dynamic routing problem [69] and, more recently, to automatic programming [181].

13.6 CONCLUDING REMARKS

The development of portable, efficient, and easy-to-use software tools, together with the availability of a wide range of faster parallel machines with increasing reliability and decreasing costs, has brought parallel processing into reality as an effective strategy for the implementation of computationally efficient techniques for the solution of large, difficult optimization problems.

Parallel implementations of metaheuristics are a quite effective alternative for the approximate solution of combinatorial optimization problems. They are usually applied in the context of complex applications, often allowing reductions in computation times. Good speedups can be obtained with parallel non-cooperative strategies. However, parallel metaheuristics not only allow solving larger problems or finding improved solutions with respect to their sequential counterparts, but parallelizations based on cooperative multiple search threads also lead to more robust implementations, which are likely to be the most important contribution of parallelism to metaheuristics. The use of multiple processors involved in a broader search of the solution space, using different strategies and parameter values, allows for larger diversity and deeper

investigation of the solution space. As a consequence, improved solutions may be found, very often even faster than in sequential mode. To summarize, parallelism often leads to speedups for complex applications and to more robust algorithms, due to improved mechanisms for search intensification and diversification.

We reviewed several applications of parallel implementations of metaheuristics such as tabu search, GRASP, genetic algorithms, simulated annealing, and ant colonies to a wide variety of hard combinatorial problems. Successful implementations of metaheuristics are often based on the appropriate combination of components of different methods. In the same vein, the hybridization of different metaheuristics in the framework of a cooperative parallel implementation (such as a genetic algorithm and several tabu search threads cooperating by means of a pool of elite solutions generated by the latter and improved by the former, or the construction phase of a GRASP procedure being used to create initial subpopulations for a parallel implementation of a genetic algorithm based on an island model, among many other possibilities) is a very promising research avenue.

The judicious choice of a programming environment is essential to the implementation of a parallel metaheuristic. Implementation issues may lead to programs that perform very poorly, independently of the quality of the underlying parallel algorithm. One possible source of problems is a mismatch between the proposed algorithm and the programming model supported by a parallel machine or a development tool. It is also common for the program, once implemented, to have a different behavior from what was expected due to unforeseen communication and processing bottlenecks. Performance evaluation tools have a fundamental role in solving this kind of problem, since they can help the programmer in identifying the origin of such bottlenecks.

Acknowledgments: The work of S.L. Martins was sponsored by FAPERJ grant 151921/99. The work of C.C. Ribeiro was sponsored by FAPERJ grant 150966/99 and CNPq research grants 302281/85-1, 202005/89-5, and 910062/99-4.

References

- [1] E.H.L. Aarts, F.M.J. de Bont, J.H.A. Habers, and P.J.M. van Laarhoven. Parallel Implementations of the Statistical Cooling Algorithm. *Integration*, 4:209–238, 1986.
- [2] E.H.L. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines*. Wiley, 1989.
- [3] E.H.L. Aarts and J. Korst. Selected Topics in Simulated Annealing. In: *Essays and Surveys in Metaheuristics*, C.C. Ribeiro and P. Hansen, editors, Kluwer, 2001 (this volume).

- [4] E.H.L. Aarts and M. Verhoeven. Local Search. In: *Annotated Bibliographies in Combinatorial Optimization*, M. Dell'Amico, F. Maffioli, and S. Martello, editors, pages 163–180, Wiley, 1997.
- [5] D. Abramson and J. Abela. A Parallel Genetic Algorithm for Solving the School Timetabling Problem. In: *Proceedings of the 15th Australian Computer Science Conference*, pages 1–11, 1992.
- [6] R.M. Aiex, S.L. Martins, C.C. Ribeiro, and N.R. Rodriguez. Cooperative Multi-Thread Parallel Tabu Search with an Application to Circuit Partitioning. *Lecture Notes in Computer Science*, 1457:310–331, 1998.
- [7] R.M. Aiex, M.G.C. Resende, P.M. Pardalos, and G. Toraldo. GRASP with Path-Relinking for the Three-Index Assignment Problem. Submitted for publication, 2000.
- [8] R.M. Aiex, M.G.C. Resende, and C.C. Ribeiro. Probability Distribution of Solution Time in GRASP: An Experimental Investigation. To appear in: *Journal of Heuristics*.
- [9] S.M. Alaoui, O. Frieder, and T. El-Ghazawi. A Parallel Genetic Algorithm for Task Mapping on Parallel Machines. *Lecture Notes in Computer Science*, 1586:201–209, 1999.
- [10] E. Alba and J.M. Troya. Influence of the Migration Policy in Parallel Distributed GAs with Structured and Panmictic Populations. *Applied Intelligence*, 12:163–181, 2000.
- [11] E. Alba and J.M. Troya. Analyzing Synchronous and Asynchronous Parallel Distributed Genetic Algorithms. *Future Generation Computer Systems*, 17:451–465, 2001.
- [12] J.R. Allwright and D.B. Carpenter. A Distributed Implementation of Simulated Annealing for the Travelling Salesman Problem. *Parallel Computing*, 10:335–338, 1989.
- [13] A.C. Alvim. *Parallelization Strategies of the GRASP Metaheuristic* (in Portuguese). M.Sc. Dissertation, Department of Computer Science, Catholic University of Rio de Janeiro, 1998.
- [14] A.C. Alvim and C.C. Ribeiro. Load Balancing for the Parallelization of the GRASP Metaheuristic (in Portuguese). In: *Proceedings of the X Brazilian Symposium on Computer Architecture*, pages 279–282, Búzios, 1998.
- [15] D. Andre and J.R. Koza. Parallel Genetic Programming: A Scalable Implementation using the Transputer Network Architecture. In: *Advances in Genetic Programming 2*, P.J. Angeline and K.E. Kinnear, Jr., editors, pages 317–338, MIT Press, 1996.

- [16] A.A. Andreatta and C.C. Ribeiro. A Graph Partitioning Heuristic for the Parallel Pseudo-Exhaustive Logical Test of VLSI Combinational Circuits. *Annals of Operations Research*, 50:1–36, 1994.
- [17] Argonne National Laboratory. PGAPack Parallel Genetic Algorithm Library. Online document, http://www-fp.mcs.anl.gov/CCST/research/reports_pre1998/comp_bio/stalk/pgapack.html, last visited on February 11, 2001.
- [18] G. Authié, A. Ferreira, J.-L. Roch, G. Villard, J. Roman, C. Roucairol, and B. Virot (editors). *Algorithmique Parallèle: Analyse et Conception*. Hermès, 1994.
- [19] R. Azencott. *Simulated Annealing: Parallelization Techniques*. Wiley, 1992.
- [20] P. Badeau, F. Guertin, M. Gendreau, J.-Y. Potvin, and E. Taillard. A Parallel Tabu Search Heuristic for the Vehicle Routing Problem with Time Windows. *Transportation Research-C*, 5:109–122, 1997.
- [21] P. Banerjee and M. Jones. A Parallel Simulated Annealing Algorithm for Standard Cell Placement on a Hypercube Computer. In: *Proceedings of the 1986 International Conference on Computer-Aided Design*, pages 34–37, Santa Clara, 1986.
- [22] P. Banerjee, M. Jones, and J. Sargent. Parallel Simulated Annealing Algorithms for Standard Cell Placement on Hypercube Multi-Processors. *IEEE Transactions on Parallel and Distributed Systems*, 1:91–106, 1990.
- [23] M.P. Bastos and C.C. Ribeiro. Reactive Tabu Search with Path Relinking for the Steiner Problem in Graphs. In: *Essays and Surveys in Metaheuristics*, C.C. Ribeiro and P. Hansen, editors, Kluwer, 2001 (this volume).
- [24] R. Battiti and G. Tecchiolli. Parallel Biased Search for Combinatorial Optimization: Genetic Algorithms and TABU. *Microprocessors and Microsystems*, 16:351–367, 1992.
- [25] R. Battiti and G. Tecchiolli. The Reactive Tabu Search. *ORSA Journal on Computing*, 6:126–140, 1994.
- [26] J.E. Beasley. OR-Library: Distributing Test Problems by Electronic Mail. *Journal of the Operational Research Society*, 41:1069–1072, 1990.
- [27] T.C. Belding. The Distributed Genetic Algorithm Revisited. In: *Proceedings of the Sixth International Conference on Genetic Algorithms*, L. Eschelman, editor, pages 114–121, Morgan Kaufmann, 1995.
- [28] R. Bianchini and C.M. Brown. Parallel Genetic Algorithms on Distributed-Memory Architectures. *Transputer Research and Applications*, 6:67–82, 1993.

- [29] M. Bolondi and M. Bondanza. *Parallelizzazione di un algoritmo per la risoluzione del problema del commesso viaggiatore*. M.Sc. Dissertation, Dipartimento di Elettronica e Informazione, Politecnico di Milano, 1993.
- [30] R.K. Brunner and L.V. Kale. Adapting to Load on Workstation Clusters. In: *Proceedings of the Seventh Symposium on the Frontiers of Massively Parallel Computation*, pages 106–112, IEEE Computer Society Press, 1999.
- [31] M. Bubak and K. Sowa. Object-Oriented Implementation of Parallel Genetic Algorithms. In: *High Performance Cluster Computing: Programming and Applications*, R. Buyya, editor, vol. 2, pages 331–349, Prentice Hall, 1999.
- [32] B. Bullnheimer, G. Kotsis, and C. Strauss. Parallelization Strategies for the Ant System. *Applied Optimization*, 24:87–100, 1998.
- [33] D. Butenhof. *Programming with POSIX Threads*. Addison Wesley, 1997.
- [34] R. Buyya (editor). *High Performance Cluster Computing: Architectures and Systems*. Prentice-Hall, 1999.
- [35] E. Cantú-Paz. A Survey of Parallel Genetic Algorithms. *Calculateurs Parallèles, Réseaux et Systèmes Répartis*, 10:141–171, 1998.
- [36] E. Cantú-Paz. Implementing Fast and Flexible Parallel Genetic Algorithms, volume III. In: *Practical Handbook of Genetic Algorithms*, L.D. Chambers, editor, pages 65–84, CRC Press, 1999.
- [37] E. Cantú-Paz. *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer, 2000.
- [38] E. Cantú-Paz. *Migration Policies, Selection Pressure, and Parallel Evolutionary Algorithms*. To appear in: *Journal of Heuristics*.
- [39] E. Cantú-Paz and D.E. Goldberg. Predicting Speedups of Idealized Bounding Cases of Parallel Genetic Algorithms. In: *Proceedings of the Seventh International Conference on Genetic Algorithms*, T. Bäck, editor, pages 113–121, Morgan Kaufmann, 1997.
- [40] E. Cantú-Paz and D.E. Goldberg. Parallel Genetic Algorithms: Theory and Practice. *Computer Methods in Applied Mechanics and Engineering*, 186:221–238, 2000.
- [41] S.A. Canuto. *Local Search for the Prize-Collecting Steiner Tree Problem* (in Portuguese). M.Sc. Dissertation, Department of Computer Science, Catholic University of Rio de Janeiro, 2000.
- [42] S.A. Canuto, M.G.C. Resende, and C.C. Ribeiro. Local Search with Perturbations for the Prize-Collecting Steiner Tree Problem in Graphs. To appear in: *Networks*.

- [43] N. Carriero and D. Gelernter. How to Write Parallel Programs: A Guide to the Perplexed. *ACM Computing Surveys*, 21:323–357, 1989.
- [44] A. Casotto, F. Romeo, and A. Sangiovanni-Vincentelli. A Parallel Simulated Annealing Algorithm for the Placement of Macro-Cells. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-6:727–751, 1987.
- [45] A. Casotto and A. Sangiovanni-Vincentelli. Placement of Standard Cells using Simulated Annealing on the Connection Machine. In: *Proceedings of the 1987 International Conference on Computer-Aided Design*, pages 350–353, Santa Clara, 1987.
- [46] C.B. Cavalcante, V.C. Cavalcante, C.C. Ribeiro, and C.C. de Souza. Parallel Cooperative Approaches for the Labor Constrained Scheduling Problem. In: *Essays and Surveys in Metaheuristics*, C.C. Ribeiro and P. Hansen, editors, Kluwer, 2001 (this volume).
- [47] C.B. Cavalcante, Y. Colombani, S. Heipcke, and C.C. Souza. Scheduling under Labour Resource Constraints. *Constraints*, 5:415–422, 2000.
- [48] C.B. Cavalcante, C.C. Souza, M.W. Savelsbergh, Y. Wang, and L.A. Wolsey. Scheduling Projects with Labor Constraints. To appear in: *Discrete Applied Mathematics*.
- [49] G. Cavalheiro, F. Galilée, and J.-L. Roch. Athapaskan-1: Parallel Programming with Asynchronous Tasks. Online document, <http://www-apache.imag.fr/software/ath1/publications/files/98-ath1-yale.ps.gz>, last visited on March 14, 2001.
- [50] N. Carriero and D. Gelernter. Linda in Context. *Communications of the ACM*, 32:444–458, 1989.
- [51] J. Chakrapani and J. Skorin-Kapov. Massively Parallel Tabu Search for the Quadratic Assignment Problem. *Annals of Operations Research*, 41:327–341, 1993.
- [52] J. Chakrapani and J. Skorin-Kapov. Connection Machine Implementation of a Tabu Search Algorithm for the Traveling Salesman Problem. *Journal of Computing and Information Technology*, 1:29–63, 1993.
- [53] P. Chalermwat, T. El-Ghazawi, and J. LeMoigne. 2-Phase GA-Based Image Registration on Parallel Clusters. *Future Generation Computer Systems*, 17:467–476, 2001.
- [54] J.A. Chandy and P. Banerjee. Parallel Simulated Annealing Strategies for VLSI Cell Placement. In: *Proceedings of the 9th International Conference on VLSI Design*, Bangalore, 1996.

- [55] J.A. Chandy, S. Kim, B. Ramkumar, S. Parkes, and P. Banerjee. An Evaluation of Parallel Simulated Annealing Strategies with Applications to Standard Cell Placement. *IEEE Transactions on Computer Aided Design*, 16:398–410, 1997.
- [56] A. Chipperfield and P. Fleming. Parallel Genetic Algorithms. In: *Parallel and Distributed Computing Handbook*, A.Y. Zomaya, editor, pages 1118–1143, McGraw-Hill, 1996.
- [57] A. Colorni, M. Dorigo, and V. Maniezzo. Distributed Optimization by Ant Colonies. In: *Proceedings of the European Conference on Artificial Life*, pages 134–142, Paris, Elsevier, 1991.
- [58] U.S. da Costa, D.B. Déharbe, and A.M. Moreira. Variable Orderings of BDDs with Parallel Genetic Algorithms. In: *Proceedings of the International Conference on Parallel and Distributes Processing Techniques and Applications*, pages 1181–1186, Las Vegas, CSREA Press, 2000.
- [59] T.G. Crainic and M. Gendreau. Cooperative Parallel Tabu Search for Capacitated Network Design. To appear in: *Journal of Heuristics*.
- [60] T.G. Crainic and M. Toulouse. Parallel Metaheuristics. In: *Fleet Management and Logistics*, T.G. Crainic and G. Laporte, editors, pages 205–251, Kluwer, 1998.
- [61] T.G. Crainic, M. Toulouse, and M. Gendreau. Parallel Asynchronous Tabu Search in Multicommodity Location-Allocation with Balancing Requirements. *Annals of Operations Research*, 63:277–299, 1995.
- [62] T.G. Crainic, M. Toulouse, and M. Gendreau. Synchronous Tabu Search Parallelization Strategies for Multicommodity Location-Allocation with Balancing Requirements. *OR Spektrum*, 17:113–123, 1995.
- [63] T.G. Crainic, M. Toulouse, and M. Gendreau. Towards a Taxonomy of Parallel Tabu Search Algorithms. *INFORMS Journal of Computing*, 9:61–72, 1997.
- [64] D.E. Culler, P.S. Jaswinder, and A. Gupta. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann, 1999.
- [65] V.-D. Cung, T. Mautor, P. Michelon, and A. Tavares. A Scatter Search Based Approach for the Quadratic Assignment Problem. In: *Proceedings of the IEEE International Conference on Evolutionary Computation and Evolutionary Programming*, T. Baeck, Z. Michalewicz, and X. Yao, editors, pages 165–170, IEEE, 1997.
- [66] L. Dagum and R. Menon. OpenMP: An Industry-Standard API for Shared-Memory Programming. *IEEE Computational Science and Engineering*, 5:46–55, 1998.

- [67] F. Darema, S. Kirkpatrick, and V. Norton. Parallel Algorithms for Chip Placement by Simulated Annealing. *IBM Journal of Research and Development*, 31:391–402, 1987.
- [68] I. De Falco, R. Del Balio, A. Della Cioppa, and E. Tarantino. A Parallel Genetic Algorithm for Transonic Airfoil. In: *Proceedings of the IEEE International Conference on Evolutionary Computing*, pages 429–434, University of Western Australia, 1995.
- [69] G. Di Caro and M. Dorigo. A Mobile Agents Approach to Adaptive Routing. In: *Proceedings of the 31st Hawaii International Conference on System*, pages 74–83, IEEE Computer Society Press, 1998.
- [70] N. Dodd. Slow Annealing Versus Multiple Fast Annealing Runs: An Empirical Investigation. *Parallel Computing*, 16:269–272, 1990.
- [71] M. Dorigo, G. Di Caro, and L.M. Gambardella. Ant Algorithms for Discrete Optimization. *Artificial Life*, 5:137–172, 1999.
- [72] C.W. Duin and S. Voss. Efficient Path and Vertex Exchange in Steiner Tree Algorithms. *Networks*, 29:89–105, 1997.
- [73] M.D. Durand. Parallel Simulated Annealing: Accuracy vs. Speed in Placement. *IEEE Design and Test of Computers*, 6:8–34, 1989.
- [74] M.D. Durand and S.R. White. Trading Accuracy for Speed in Parallel Simulated Annealing Algorithms. *Parallel Computing*, 26:135–150, 2000.
- [75] H.T. Eikelder, M. Verhoeven, T. Vossen, and E. Aarts. A Probabilistic Analysis of Local Search. In: *Metaheuristics: Theory and Applications*, I. Osman and J. Kelly, editors, pages 605–618, Kluwer, 1996.
- [76] A. Fachat and K.H. Hoffman. Implementation of Ensemble-Based Simulated Annealing with Dynamic Load Balancing under MPI. *Computer Physics Communications*, 107:49–53, 1997.
- [77] E. Felten, S.C. Karlin, and S.W. Otto. The Traveling Salesman Problem on a Hypercubic, MIMD Computer. In: *Proceedings of the 1985 International Conference on Parallel Processing*, pages 6–10, 1985.
- [78] T.A. Feo and M.G.C. Resende. A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem. *Operations Research Letters*, 8:67–71, 1989.
- [79] T.A. Feo and M.G.C. Resende. Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, 6:109–133, 1995.
- [80] T.A. Feo, M.G.C. Resende, and S.H. Smith. A Greedy Randomized Adaptive Search Procedure for Maximum Independent Set. *Operations Research*, 42:860–878, 1994.

- [81] F. Fernandez, J.M. Sanchez, M. Tomassini, and J.A. Gomez. A Parallel Genetic Programming Tool Based on PVM. *Lecture Notes in Computer Science*, 1697:241–248, 1999.
- [82] P. Festa and M.G. Resende. GRASP: An Annotated Bibliography. In: *Essays and Surveys in Metaheuristics*, C.C. Ribeiro and P. Hansen, editors, Kluwer, 2001 (this volume).
- [83] C.N. Fiechter. A Parallel Tabu Search Algorithm for Large Traveling Salesman Problems. *Discrete Applied Mathematics*, 51:243–267, 1994.
- [84] M.J. Flynn. Very High-Speed Computing Systems. *Proceedings of the IEEE*, 54:1901–1909, 1966.
- [85] I. Foster. *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Addison-Wesley, 1995.
- [86] R. Frost. Ensemble Based Simulated Annealing. Online document, <http://www-rohan.sdsu.edu/~frostr/Ebsa/Ebsa.html>, last visited on February 11, 2001.
- [87] B.-L. Garcia, J.-Y. Potvin, and J.-M. Rousseau. A Parallel Implementation of the Tabu Search Heuristic for Vehicle Routing Problems with Time Windows Constraints. *Computers and Operations Research*, 21:1025–1033, 1994.
- [88] S. Geman and D. Geman. Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Transactions on Pattern Analysis and Mechanical Intelligence*, 6:721–741, 1984.
- [89] M. Gendreau, F. Guertin, J-Y. Potvin, and E. Taillard. Parallel Tabu Search for Real-Time Vehicle Routing and Dispatching. *Transportation Science*, 33:381–390, 1999.
- [90] A. Globus, J. Lawton, and T. Wipke. Automatic Molecular Design using Evolutionary Techniques. *Nanotechnology*, 10:290–299, 1999.
- [91] F. Glover. Tabu Search – Part I. *ORSA Journal on Computing*, 1:190–206, 1989.
- [92] F. Glover. Tabu Search – Part II. *ORSA Journal on Computing*, 2:4–32, 1990.
- [93] F. Glover. Ejection Chains, Reference Structures and Alternating Path Methods for Traveling Salesman Problems. *Discrete Applied Mathematics*, 65:223–253, 1996.
- [94] F. Glover. Tabu Search and Adaptive Memory Programming — Advances, Applications and Challenges. In: *Interfaces in Computer Science and Operations Research*, R.S. Barr, R.V. Helgason, and J.L. Kennington, editors, pages 1–75, Kluwer, 1996.

- [95] F. Glover. Multi-Start and Strategic Oscillation Methods — Principles to Exploit Adaptive Memory. In: *Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research*, M. Laguna and J.L. González-Velarde, editors, pages 1–24, Kluwer, 2000.
- [96] F. Glover and M. Laguna. *Tabu Search*. Kluwer, 1997.
- [97] F. Glover, M. Laguna, and R. Martí. Fundamentals of Scatter Search and Path Relinking. *Control and Cybernetics*, 39:653–684, 2000.
- [98] F. Glover and E. Pesch. TSP Ejection Chains. *Discrete Applied Mathematics*, 76:165–181, 1997.
- [99] V.S. Gordon and D. Whitley. Serial and Parallel Genetic Algorithms as Function Optimizers. In: *Proceedings of the Fifth International Conference on Genetic Algorithms*, S. Forrest, editor, pages 177–183, Morgan Kaufmann, 1993.
- [100] D.R. Greening. Parallel Simulated Annealing Techniques. *Physica D*, 42:293–306, 1990.
- [101] J.J. Grefenstette. Parallel Adaptive Algorithms for Function Optimizations. Technical Report CS-81-19, Vanderbilt University, 1981.
- [102] W. Gropp, S. Huss-Lederman, A. Lumsdaine, E. Lusk, B. Nitzberg, W. Saphir, and M. Snir. *MPI: The Complete Reference, Volume 2 – The MPI Extensions*. MIT Press, 1998.
- [103] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing-Interface*. MIT Press, 1994.
- [104] A. Gueist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine — A User’s Guide and Tutorial for Networked Parallel Computing*. MIT Press, 1994 (online document, <http://www.netlib.org/pvm3/book/pvm-book.html>, last visited on May 7, 2001).
- [105] High Performance FORTRAN Forum. High Performance Fortran. Online document, <http://dacnet.rice.edu/Depts/CRPC/HPFF/>, last visited on May 7, 2001.
- [106] J.H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. University of Michigan Press, 1975.
- [107] J.H. Holland. Genetic Algorithms. *Scientific American*, 267:44–50, 1992.
- [108] H. Hoos and T. Stützle. Towards a Characterisation of the Behaviour of Stochastic Local Search Algorithms for SAT. *Artificial Intelligence*, 112:213–232, 1999.

- [109] L. Ingber. Lester Ingber's Archive. Online document, <http://www.ingber.com/>, last visited on February 11, 2001.
- [110] Institute of Electric and Electronic Engineering. *Information Technology - Portable Operating Systems Interface (POSIX) – Part 1 – Amendment 2: Threads Extension*. 1995.
- [111] L.V. Kale, M. Bhandarkar, R. Brunner, and J. Yelon. Multiparadigm, Multilingual Interoperability: Experience with Converse. *Lecture Notes in Computer Science*, 1388:111–112, 1998.
- [112] L.V. Kale, R. Brunner, J. Phillips, and K. Varadarajan. Application Performance of a Linux Cluster using Converse. *Lecture Notes in Computer Science*, 1586:483–495, 1999.
- [113] L.V. Kale and S. Krishnan. Charm++: Parallel Programming with Message-Driven Objects. In: *Parallel Programming using C++*, G.V. Wilson and P. Lu, editors, pages 175–213, MIT Press, 1996.
- [114] S. Kim, J.A. Chandy, S. Parkes, B. Ramkumar, and P. Banerjee. ProperPLACE: A Portable, Parallel Algorithm for Cell Placement. In: *Proceedings of the 8th International Parallel Processing Symposium*, pages 932–941, Cancun, 1994.
- [115] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimisation by Simulated Annealing. *Science*, 220:671–680, 1983.
- [116] G. Kliewer, K. Klohs, and S. Tschöke. Parallel Simulated Annealing Library (parSA): User Manual. Technical Report, Computer Science Department, University of Paderborn, 1999.
- [117] A. Knies, M. O'Keefe, and T. MacDonald. High Performance FORTRAN: A Practical Analysis. *Scientific Programming*, 3:187–199, 1994.
- [118] C. Koelbel, D. Loveman, R. Schreiber, G. Steele Jr., and M. Zosel. *The High Performance FORTRAN Handbook*. The MIT Press, 1994.
- [119] U. Kohlmorgen, H. Schmeck, and K. Haase. Experiences with Fine-Grained Parallel Genetic Algorithms. *Annals of Operations Research*, 90:203–220, 1999.
- [120] S.A. Kravitz and R.A. Rutenbar. Placement by Simulated Annealing on a Multiprocessor. *IEEE Transactions on Computer Aided Design*, 6:534–549, 1987.
- [121] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing Design and Analysis of Parallel Algorithms*. Benjamin/Cummings, 1994.

- [122] S.-Y. Lee and K.G. Lee. Asynchronous Communication of Multiple Markov Chains in Parallel Simulated Annealing. In: *Proceedings of the International Conference on Parallel Processing*, volume III, pages 169–176, St. Charles, 1992.
- [123] T.-H. Lai and S. Sahni. Anomalies in Parallel Branch-And-Bound Algorithms. *Communications of the ACM*, 27:594–602, 1984.
- [124] T.-H. Lai and A. Sprague. A Note on Anomalies in Parallel Branch-and-Bound Algorithms with One-to-One Bounding Functions. *Information Processing Letters*, 23:119–122, 1986.
- [125] K.G. Lee and S.-Y. Lee. Efficient Parallelization of Simulated Annealing using Multiple Markov Chains: An Application to Graph Partitioning. In: *Proceedings of the International Conference on Parallel Processing*, volume III, pages 177–180, St. Charles, 1992.
- [126] S.-Y. Lee and K.G. Lee. Synchronous and Asynchronous Parallel Simulated Annealing with Multiple Markov Chains. *IEEE Transactions on Parallel and Distributed Systems*, 7:993–1008, 1996.
- [127] D. Levine. A Parallel Genetic Algorithm for the Set Partitioning Problem. Technical Report ANL-9423, Argonne National Laboratory, 1994.
- [128] B. Lewis. *Multithreaded Programming with Java Technology*. Prentice Hall, 2000.
- [129] Y. Li, P.M. Pardalos, and M.G.C. Resende. A Greedy Randomized Adaptive Search Procedure for Quadratic Assignment Problem. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 16:237–261, 1994.
- [130] G.-J. Li and B.W. Wah. Coping with Anomalies in Parallel Branch-and-Bound Algorithms. *IEEE Transactions on Computers*, C-35:568–573, 1986.
- [131] B. Mans and C. Roucairol. Performances of Parallel Branch and Bound Algorithms with Best-First Search. *Discrete Applied Mathematics*, 66:57–76, 1996.
- [132] N. Marco and S. Lanteri. A Two Level Parallelization Strategy for Genetic Algorithms Applied to Optimum Shape Design. *Parallel Computing*, 26:377–397, 2000.
- [133] S.L. Martins, P.M. Pardalos, M.G.C. Resende, and C.C. Ribeiro. GRASP Procedures for the Steiner Problem in Graphs. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 43:133–146, 1998.
- [134] S.L. Martins, M.G.C. Resende, C.C. Ribeiro, and P. Pardalos. A Parallel GRASP for the Steiner Tree Problem in Graphs using a Hybrid Local Search Strategy. *Journal of Global Optimization*, 17:267–283, 2000.

- [135] S.L. Martins, C.C. Ribeiro, and N.R. Rodriguez. Parallel Computing Environments. To appear in: *Handbook of Combinatorial Optimization*, P. Pardalos and M.G.C. Resende, editors, Oxford.
- [136] S.L. Martins, C.C. Ribeiro, and M.C. Souza. A Parallel GRASP for the Steiner Problem in Graphs. *Lecture Notes in Computer Science*, 1457:285–297, 1998.
- [137] T.G. Mattson. Scientific Computation. In: *Parallel and Distributed Computing Handbook*, A.Y. Zomaya, editor, pages 981–1002, Mc-Graw-Hill, 1996.
- [138] K. Mehlhorn. A Faster Approximation for the Steiner Problem in Graphs. *Information Processing Letters*, 27:125–128, 1988.
- [139] J. Merlin, S. Baden, S. Fink, and B. Chapman. Multiple Data Parallelism with HPF and KeLP. *Future Generation Computer Systems*, 15:393–405, 1999.
- [140] J. Merlin and A. Hey. An Introduction to High Performance FORTRAN. *Scientific Programming*, 4:87–113, 1995.
- [141] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, 1996.
- [142] M. Miki, T. Hiroyasu, and M. Kasai. Application of the Temperature Parallel Simulated Annealing to Continous Optimization Problems. *IPSL Transaction*, 41:1607–1616, 2000.
- [143] L.B. Morales, R. Garduño-Juárez, J.M. Aguilar-Alvarado, and F.J. Riveros-Castro. A Parallel Tabu Search for Conformational Energy Optimization of Oligopeptides. *Journal of Computational Chemistry*, 21:147–156, 2000.
- [144] H.S. Morse. *Practical Parallel Computing*. AP Professional, 1994.
- [145] MPI Forum. MPI: A Message-Passing Interface Standard. *The International Journal of Supercomputing Applications and High Performance Computing*, 8:159–416, 1994.
- [146] MPI Forum. A Message-Passing Interface Standard. Online document, <http://www mpi-forum.org/docs/docs.html>, last visited on February 19, 2001.
- [147] MPI Forum. MPI-2: Extensions to the Message-Passing Interface. Online document, <http://www mpi-forum.org/docs/docs.html>, last visited on February 19, 2001.

- [148] H. Muhlenbein. Parallel Genetic Algorithms in Combinatorial Optimization. In: *Computer Science and Operations Research: New Developments in their Interfaces*, O. Balci, R. Sharda, and S. Zenios, editors, pages 441–456, Pergamon Press, 1992.
- [149] R.A. Murphrey, P.M. Pardalos, and L.S. Pitsoulis. A Parallel GRASP for the Data Association Multidimensional Assignment Problem. *IMA Volumes in Mathematics and its Applications*, 106:159–180, 1998.
- [150] S. Oak and H. Wong. *Java Threads*. O'Reilly, 1997.
- [151] L.S. Ochi, L.M. Drummond, A.O. Victor, and D.S. Vianna. A Parallel Evolutionary Algorithm for Solving the Vehicle Routing Problem with Heterogeneous Fleet. *Future Generation Computer Systems Journal*, 14:285–292, 1998.
- [152] L. Osborne and B. Gillett. A Comparison of Two Simulated Annealing Algorithms Applied to the Directed Steiner Problem on Networks. *ORSA Journal on Computing*, 3:213–225, 1991.
- [153] M. Oussaidène, B. Chopard, O.V. Pictel, and M. Tomassini. Parallel Genetic Programming and its Application to Trading Model Induction. *Parallel Computing*, 23:1183–1198, 1997.
- [154] P. Pardalos, L. Pitsoulis, and M.G. Resende. A Parallel GRASP Implementation for the Quadratic Assignment Problem. In: *Parallel Algorithms for Irregular Problems: State of Art*, A. Ferreira and J. Rolim, editors, pages 115–133, Kluwer, 1995.
- [155] P. Pardalos, L. Pitsoulis, and M.G. Resende. A Parallel GRASP for MAX-SAT. *Lecture Notes in Computer Science*, 1184:575–585, 1996.
- [156] S. Parkes, J.A. Chandy, and P. Banerjee. A Library-Based Approach to Portable, Parallel, Object-Oriented Programming: Interface, Implementation, and Application. In: *Proceedings of the ACM Supercomputing'94 Conference*, pages 69–78, Washington, 1994.
- [157] C.C. Pettey, M.R. Leuze, and J. Grefenstette. A Parallel Genetic Algorithm. In: *Proceedings of the Second International Conference on Genetic Algorithms*, J. Grefenstette, editor, pages 155–161, Lawrence Erlbaum Associates, 1987.
- [158] G.F. Pfister. *In Search of Clusters*. Prentice-Hall, 1998.
- [159] B. Planquelle, J.-F. Méhaut, and N. Revol. Multi-Cluster Approach with PM2. In: *Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications*, pages 779–785, Las Vegas, 1999.

- [160] A. Plastino, C.C. Ribeiro, and N.R. Rodriguez. A Framework for SPMD Applications. In: *Proceedings of the XII Brazilian Symposium on Computer Architecture*, pages 245–252, São Pedro, 2000.
- [161] S.C. Porto and C.C. Ribeiro. Parallel Tabu Search Message-Passing Synchronous Strategies for Task Scheduling under Precedence Constraints. *Journal of Heuristics*, 1:207–223, 1995.
- [162] S.C. Porto and C.C. Ribeiro. A Tabu Search Approach to Task Scheduling on Heterogeneous Processors under Precedence Constraints. *International Journal of High Speed Computing*, 7:45–71, 1995.
- [163] S.C. Porto and C.C. Ribeiro. A Case Study on Parallel Synchronous Implementations of Tabu Search Based on Neighborhood Decomposition. *Investigación Operativa*, 5:233–259, 1996.
- [164] S.C. Porto, J.P. Kitajima, and C.C. Ribeiro. Performance Evaluation of a Parallel Tabu Search Task Scheduling Algorithm. *Parallel Computing*, 26:73–90, 2000.
- [165] M. Prais and C.C. Ribeiro. Reactive GRASP: An Application to a Matrix Decomposition Problem in Traffic Assignment. *INFORMS Journal on Computing*, 12:164–176, 2000.
- [166] M.J. Quinn. *Parallel Computing: Theory and Practice*. McGraw-Hill, 1994.
- [167] K.H. Randall. *Cilk: Efficient Multithreaded Computing*. Ph.D. Dissertation, MIT, Department of Electrical Engineering and Computer Science, 1998.
- [168] C. Rego and C. Roucairol. A Parallel Tabu Search Algorithm using Ejection Chains for the VRP. In: *Metaheuristics: Theory and Applications*, I.H. Osman and J.P. Kelly, editors, pages 253–295, Kluwer, 1996.
- [169] C.R. Reeves. Genetic Algorithms. In: *Modern Heuristic Techniques for Combinatorial Problems*, C.R. Reeves, editor, pages 151–196, Blackwell, 1993.
- [170] M.G.C. Resende. Computing Approximate Solutions of the Maximum Covering Problem Using GRASP. *Journal of Heuristics*, 4:161–171, 1998.
- [171] M.G.C. Resende, T.A. Feo, and S.H. Smith. Algorithm 787: Fortran Subroutines for Approximate Solutions of Maximum Independent Set Problems Using GRASP. *ACM Transactions on Mathematical Software*, 24:386–394, 1998.

- [172] M.G.C. Resende, P. Pardalos, and Y. Li. Algorithm 754: Fortran Subroutines for Approximate Solutions of Dense Quadratic Assignment Problems Using GRASP. *ACM Transactions on Mathematical Software*, 22:104–118, 1996.
- [173] M.G.C. Resende, L. Pitsoulis, and P. Pardalos. Approximate Solution of Weighted MAX-SAT Problems Using GRASP. *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, 35:393–405, 1997.
- [174] M.G.C. Resende, L. Pitsoulis, and P. Pardalos. Fortran Subroutines for Computing Approximate Solutions of MAX-SAT Problems Using GRASP. *Discrete Applied Mathematics*, 100:95–13, 2000.
- [175] M.G.C. Resende and C.C. Ribeiro. A GRASP for Graph Planarization. *Networks*, 29:173–189, 1997.
- [176] C.C. Ribeiro, M. Minoux, and M.C. Penna. An Optimum Column-Generation-with-Ranking Algorithm for Very Large Scale Set Partitioning Problems in Traffic Assignment. *European Journal of Operational Research*, 41:232–239, 1989.
- [177] C.C. Ribeiro and M.G.C. Resende. Algorithm 797: Fortran Subroutines for Approximate Solutions of Graph Planarization Problems Using GRASP. *ACM Transactions on Mathematical Software*, 25:341–352, 1999.
- [178] W. Rivera-Gallego. A Genetic Algorithm for Circulant Euclidean Distance Matrices. *Journal of Applied Mathematics and Computation*, 97:197–208, 1998.
- [179] J.S. Rose, W.M. Snelgrove, and Z.G. Vranesic. Parallel Cell Placement Algorithms with Quality Equivalent to Simulated Annealing. *IEEE Transactions on Computer-Aided Design*, 7:387–396, 1998.
- [180] P. Roussel-Ragot and G. Dreyfus. A Problem-Independent Parallel Implementation of Simulated Annealing: Models and Experiments. *IEEE Transactions on Computer-Aided Design*, 9:827–835, 1990.
- [181] O. Roux and C. Fonlupt. Ant Programming: Or How to use Ants for Automatic Programming. In: *Proceedings of the 2nd International Workshop on Ant Colony Optimization*, pages 121–129, Brussels, 2000.
- [182] R.A. Rutenbar and S.A. Kravitz. Layout by Annealing in a Parallel Environment. In: *Proceedings of the IEEE International Conference on Computer Design*, pages 434–437, Port Chester, 1986.
- [183] A. Salhi. Parallel Implementation of a Genetic-Programming Based Tool for Symbolic Regression. *Information Processing Letters*, 66:299–307, 1998.

- [184] J. Schulze and T. Fahle. A Parallel Algorithm for the Vehicle Routing Problem with Time Window Constraints. *Annals of Operations Research*, 86:585–607, 1999.
- [185] Scientific Computing Associates. *Linda's User's Guide and Reference Manual, version 4.0.1 – SP2/POE*. 1995.
- [186] B. Selman, H. Kautz, and B. Cohen. Noise Strategies for Improving Local Search. In: *Proceedings of the 12th National Conference on Artificial Intelligence*, pages 337–343, Seattle, MIT Press, 1994.
- [187] G.A. Sena, D. Megherbi, and G. Isern. Implementation of a Parallel Genetic Algorithm on a Cluster of Workstations: Traveling Salesman Problem, a Case Study. *Future Generation Computer Systems*, 17:477–488, 2001.
- [188] T. Stützle. Parallelization Strategies for Ant Colony Optimization. *Lecture Notes in Computer Science*, 1498:722–731, 1998.
- [189] T. Stützle and H. Hoos. The MAX-MIN Ant System and Local Search for the Traveling Salesman Problem. In: *Proceedings of the 1997 IEEE 4th International Conference on Evolutionary Computation*, pages 308–313, IEEE Press, 1997.
- [190] Supercomputing Technologies Group. CILK 5.3.1 Reference Manual. MIT Laboratory for Computer Science. Online document, <http://supertech.lcs.mit.edu/cilk>, last visited on March 14, 2001.
- [191] E. Taillard. Robust Taboo Search for the Quadratic Assignment Problem. *Parallel Computing*, 7:443–455, 1991.
- [192] E. Taillard. Parallel Taboo Search Techniques for the Job Shop Scheduling Problem. *ORSA Journal on Computing*, 6:108–117, 1994.
- [193] E. Taillard. Ant Systems. To appear in: *Handbook of Applied Optimization*, P. Pardalos and M.G.C. Resende, editors, Oxford.
- [194] E. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J.-Y. Potvin. A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows. *Transportation Science*, 31:170–186, 1997.
- [195] H. Takahashi and A. Matsuyama. An Approximate Solution for the Steiner Problem in Graphs. *Mathematica Japonica*, 24:573–577, 1980.
- [196] E.-G. Talbi, Z. Hafidi, and J.-M. Geib. A Parallel Adaptive Tabu Search Approach. *Parallel Computing*, 24:2003–2019, 1998.

- [197] E.-G. Talbi, Z. Hafidi, and J-M. Geib. Parallel Adaptive Tabu Search for Large Optimization Problems. In: *Metaheuristics: Advances and Trends in Local Search Paradigms for Optimization*, S. Voss, S. Martello, I.H. Osman, and C. Roucairol, editors, pages 255–266, Kluwer, 1999.
- [198] E.-G. Talbi and T. Muntean. Hill-climbing, Simulated Annealing and Genetic Algorithms: A Comparative Study. In: *Proceedings of the International Conference on Systems Sciences: Task Scheduling in Parallel and Distributed Systems*, H. El-Rewini and T. Lewis, editors, pages 565–573, IEEE Computer Society Press, 1993.
- [199] E.-G. Talbi, O. Roux, C. Fonlupt, and D. Robillard. Parallel Ant Colonies for Combinatorial Optimization Problems. *Lecture Notes in Computer Science*, 1586:239–247, 1999.
- [200] E.-G. Talbi, O. Roux, C. Fonlupt, and D. Robillard. Parallel Ant Colonies for the Quadratic Assignment Problem. *Future Generation Computer Systems*, 17:441–449, 2001.
- [201] S.N. Talukdar, L. Baerentzen, and A. Gove. Asynchronous Teams: Cooperation Schemes for Autonomous Agents. *Journal of Heuristics*, 4:295–321, 1998.
- [202] R. Tanese. Distribued Genetic Algorithms. In: *Proceedings of the Third International Conference on Genetic Algorithms*, J.D. Schaffer, editor, pages 434–439, Morgan Kaufmann, 1989.
- [203] M.G.A. Verhoeven and E.H.L. Aarts. Parallel Local Search. *Journal of Heuristics*, 1:43–65, 1995.
- [204] M.G.A. Verhoeven, M.E.M. Severens, and E.H.L. Aarts. Local Search for Steiner Trees in Graphs. In: *Modern Heuristic Search Methods*, V.J. Rayward-Smith, I.H. Osman, C.R. Reeves, and G.D. Smith, editors, pages 117–129, Wiley, 1996.
- [205] S. Voss. Tabu Search: Applications and Prospects. In: *Network Optimization Problems*, D.-Z. Du and P.M. Pardalos, editors, pages 333–353, World Scientific, 1993.
- [206] D.W. Walker. The Design of a Standard Message Passing Interface for Distributed Memory Concurrent Computers. *Parallel Computing*, 20:657–673, 1994.
- [207] E.E. Witte, R.D. Chamberlain, and M.A. Franklin. Parallel Simulated Annealing using Speculative Computation. *IEEE Transactions on Parallel and Distributed Systems*, 2:483–494, 1991.

- [208] C.P. Wong and R.D. Fiebrich. Simulated Annealing-Based Circuit Placement Algorithm on the Connection Machine System. In: *Proceedings of the International Conference on Computer Design*, pages 78–82, Rye Brook, 1987.

14 ACCELERATING STRATEGIES IN COLUMN GENERATION METHODS FOR VEHICLE ROUTING AND CREW SCHEDULING PROBLEMS

Guy Desaulniers¹, Jacques Desrosiers², and Marius M. Solomon³

¹École Polytechnique and GERAD
C.P. 6079, Succ. Centre-Ville
Montréal, Québec, Canada H3C 3A7
guyd@crt.umontreal.ca

²École des Hautes Études Commerciales and GERAD
3000, chemin de la Côte-Sainte-Catherine
Montréal, Québec, Canada H3T 2A7
Jacques.Desrosiers@hec.ca

³Northeastern University and GERAD
314 Hayden Hall Boston
MA 02115, USA
Solomon@cba.neu.edu

Abstract: This paper focuses on accelerating strategies used in conjunction with column generation to solve vehicle routing and crew scheduling problems. We describe techniques directed at speeding up each of the five phases of the solution process: pre-processor, subproblem, master problem, branch-and-bound, and post-optimizer. In practical applications, these methods often were key elements for the viability of this optimization approach. The research cited here shows their use led to computational gains, and notably to solutions that could not have been obtained otherwise due to practical problem complexity and size. In particular, we present recent methods directed at the integer programming aspect of the approach that were instrumental in substantially reducing the integrality gap found in certain applications, thereby helping to efficiently produce excellent quality solutions.

14.1 INTRODUCTION

Vehicle routing problems consist of finding least cost or maximum profit feasible *routes* for a set of *vehicles* such that all requirements imposed by a given set of tasks are met. The definition of crew scheduling problems is similar and is obtained by replacing *routes* and *vehicles* with *schedules* and *crews*, respectively. Vehicle routes are generally subject to simple feasibility rules such as vehicle capacity and task precedence, while crew schedules must often comply with a variety of complex government regulations and collective agreement rules that increase the difficulty of these latter problems. Given the excellent results achieved in separately solving Vehicle Routing and Crew Scheduling (VRCS) problems, combined VRCS problems have recently been addressed in the literature. These problems, which involve routing vehicles and scheduling crews simultaneously, introduce new challenges mostly due to the need for synchronizing vehicles and crews.

VRCS problems, either considered independently or simultaneously, are important but complex practical problems. Their importance results from the need to manage vehicle expenses and crew salaries which can reach hundreds of millions of dollars annually. The main difficulty of VRCS problems comes from their size as they give rise to very large-scale mixed integer programming problems. Additionally, the modeling aspects must deal with the complexity of various work rules. Over the past ten years, column generation embedded in a branch-and-bound search tree has been used with great success to find optimal or high quality approximate solutions for this type of problems.

VRCS problems can be formulated as mixed integer programs with decomposable structures. Column generation (or equivalently, Dantzig-Wolfe decomposition) applied to these problems is considered a primal method; Lagrangian relaxation (see Geoffrion [25]) used on the same decomposable structures is seen as an equivalent dual method. Column generation was first suggested by Ford and Fulkerson [22] for a maximal multicommodity network flow problem. It was then presented by Dantzig and Wolfe [7] for linear programs with decomposable structures and by Gilmore and Gomory [26] for the cutting-stock problem.

Column generation involves a master problem (or a coordinator) and a subproblem which comprises one or several column generators. It is a two-level method which allows for an intensive use of accelerating strategies at both the coordinator and the generator levels. Furthermore, schemes directed at boosting efficiency can be used as well during the branch-and-bound phase to get integer solutions, or in suitable pre-processing and post-optimizing phases. In fact, to obtain the most of such a combined branch-and-bound and column generation approach (also known as branch-and-price), one must exploit to a maximum all problem structures at hand through the use of such techniques.

This paper presents several accelerating strategies that have been used in conjunction with column generation to solve VRCS problems. Most of them are components of the GENCOL optimizer developed at GERAD, an Operations Research center in Montréal, which has been used successfully to solve problems

in: vehicle routing with time windows (Desrosiers, Soumis, and Desrochers [18]; Desrochers, Desrosiers, and Solomon [15]; Kohl et al. [31]), pickup and delivery with time windows (Dumas et al. [20]), as well as dial-a-ride for physically disabled persons (Desrosiers, Dumas, and Soumis [17]), urban transit crew scheduling (Desrochers and Soumis [14]; Rousseau and Desrosiers [37]), multiple depot vehicle scheduling (Ribeiro and Soumis [36]; Desaulniers, Lavigne, and Soumis [11]), aircraft routing (Desaulniers et al. [8]), air crew pairing (Desaulniers et al. [10]), air crew rostering for pilots (Gamache and Soumis [23]) and flight attendants (Gamache et al. [24]), and locomotive and car assignment (Ziarati et al. [39, 40]; Cordeau et al. [5]; Lingaya et al. [34]). Recently, Barnhart et al. [3] offered their perspective on computational issues that need to be addressed when implementing column generation methods for huge integer programs.

Often, these speed up techniques were key elements for the viability of the column generation approach. Without them, it would have been almost impossible to obtain quality solutions to various applications, in a reasonable amount of computer time. By using efficient internal approximations and early termination, column generation approaches can also be turned into optimization based heuristics and used for comparison with other approximate methods for a given class of VRCS problems.

We will present accelerating strategies for each of the five phases of the solution process: pre-processor, subproblem, master problem, branch-and-bound, and post-optimizer, while noting that some involve interactions between several phases. To put things in context, the next section provides a unified solution framework for column generation approaches applied to VRCS problems. Then, the subsequent sections describe various methods that have been used in the literature to substantially accelerate each of the above stages of the solution process.

14.2 A SOLUTION FRAMEWORK FOR VRCS PROBLEMS

Essentially all VRCS problems can be formulated as mixed integer multicommodity network flow problems with additional linking constraints as well as resource variables (see Desaulniers et al. [9]). Without loss of generality, we assume throughout this paper that the objective function consists of minimizing a given cost function.

Commodities represent vehicle or crew types. Each commodity can be associated with a single vehicle or crew member. The flow structure of each commodity is defined as an elementary path on a time-space network, where arcs represent various activities while nodes are locations at specific times. Linking constraints are imposed on all tasks to perform: customers and dial-a-ride requests to satisfy; flight, train, and bus legs to cover; pairings to assign; and so on. They also limit, for example, the number of available vehicles and crews of each type at all depots and bases. Resource variables are used to model local restrictions on a single route or schedule. They are accumulated along the arcs of a path, and updated and verified at nodes. A resource represents

a constraint-specific unit of measurement, such as the elapsed time since vehicle departure from the depot location, or the cumulative work time of a crew member since the last periodic break.

In an application such as the Vehicle Routing Problem with Time Windows (VRPTW), the nodes are the locations of the customers to service exactly once, while the arcs represent the paths between these customers. Resources are associated with the visiting times and the vehicle loads at the customer locations. For the air crew pairing problem, the nodes are the airports, and the arcs can represent single flights, connections and ground waitings, among other things. Arcs can also correspond to multiple activities such as duties defined as full workdays. Tasks are the flight legs to cover exactly once, while resources are used to model most of the collective agreement restrictions. In some environments such as the locomotive assignment problem and the air crew rostering problem for flight attendants, tasks (e.g., trains and pairings, respectively) must be covered more than once.

By using an extension of the Dantzig and Wolfe [7] decomposition approach, we define a master problem - which involves all the linking constraints of the mixed integer multicommodity network flow problem treated, and a subproblem - which keeps all the local restrictions on the feasibility of each path (see Desaulniers et al. [9]). The master problem becomes a generalized set partitioning problem where each variable or column is a feasible path on the corresponding commodity network. The subproblem is a collection of resource constrained shortest path problems, one for each commodity.

An integer solution to the multicommodity flow problem is obtained using branching and cutting decisions. At each node of the search tree, the classical column generation approach proceeds as follows to derive a lower bound:

- The simplex method is used to solve the linear relaxation of the generalized set partitioning problem for the current set of columns, in the process calculating dual variables to be used for pricing out new columns.
- The subproblem is used to check whether a new column with a negative marginal cost exists. If so, it is added to the current linear programming problem, which is then re-optimized; if not, the solution to the linear relaxation of the master problem is optimal and provides a lower bound for the current node.

Because VRCS problems are complex, a number of enhancements are necessary to help the standard column generation approach accommodate very large-scale problems. They comprise heuristic methods used to find approximate solutions at the various phases of the solution process. These may also be considered control strategies designed to reduce solution time by exploiting distinctive features of the application on hand. These techniques make decisions regarding the distribution of the effort between the algorithm's components.

14.3 PRE-PROCESSOR STRATEGIES

Heuristics designed for a pre-processor are utilized at the start of the solution process, and if necessary, at every node of the branch-and-bound search tree.

Resource window reduction. Methods that reduce the time windows or, more generally, the resource windows, speed up the constrained shortest path algorithms that are used each time the column generators are called. Indeed, they reduce the resource state space. This technique was originally introduced by Desrochers, Desrosiers, and Solomon [15] and widely applied thereafter.

Heuristic arc elimination. Eliminating undesirable arcs or those unlikely to be used reduces the network sizes. This is common practice for the huge networks arising in air transportation as described, for example, by Desaulniers et al. [10] for air crew pairing applications.

Permanent arc elimination. As shown by Mingozi et al. [35], arcs can be permanently eliminated from the subproblem networks when a given criterion is met. To be used, this criterion requires a primal integer solution as well as a feasible dual solution to the linear relaxation of the problem.

Initial primal and dual solutions. An initial primal solution provides an upper bound on the optimal value that can be used to prune the branch-and-bound search tree. Moreover, such a solution is helpful to derive an initial dual solution which in turn is used in the pricing out phase. Better dual values can reduce the number of iterations between the coordinator and the column generators. Indeed, the stopping criterion of the column generation process is based on the column marginal costs, and therefore it is directly related to the values of the dual variables. This justifies the need to use efficient mechanisms in order to obtain good approximate dual variable values rapidly. This idea which only recently received attention can be exemplified as follows. For the VRPTW, the insertion heuristic of Solomon [38] could provide a fast initial primal solution, while the insertion costs generate a readily available dual solution.

Task aggregation. Another way to reduce the size of the networks, and subsequently, the row- and column-sizes of the master problem is by combining tasks. This has been used by Ioachim et al. [30] for the dial-a-ride problem, by means of mini-clusters, that is, groups of requests arising from persons willing to travel in the same direction at about the same time. Another example can be found in Hane et al. [28] where they addressed fleet assignment problems occurring in hub-and-spoke networks. In such a network structure, it is easy to identify pairs of flights incident to a spoke that have to be flown by the same aircraft.

Task sequencing. In the same dial-a-ride application for physically disabled persons, *a priori* sequencing of requests at the same location decreases the combinatorial aspect of the constrained shortest path algorithm.

Problem partitioning. Based on different criteria such as time or space, very large-scale problems can be divided into smaller ones, and if possible,

recombined at the end of the solution process. Examples of such partitioning strategies can be found in Desrosiers, Dumas, and Soumis [17] for a dial-a-ride application and in Gamache et al. [24] for an airline crew rostering application.

Rolling horizon. This is a time oriented problem partitioning. Given a division of the problem into possibly overlapping time slices, the first slice is solved and the earlier half (or a portion) of its solution is frozen. The next slice is then solved using the results of the preceding frozen solution as initial conditions on vehicles and/or crews. This classical technique continues to be used frequently as exemplified by the work of Ziarati et al. [40] on rail transport.

14.4 SUBPROBLEM STRATEGIES

Dynamic programming procedures for solving resource constrained shortest path problems are pseudo-polynomial algorithms that allow for multiple visits at a node (see Desrosiers et al. [16]). These provide heuristic solutions to the elementary shortest path versions which are \mathcal{NP} -hard problems in the strong sense (Dror [19]). Such an algorithm associates with each partial path a label L , i.e., an $(r + 1)$ -vector composed of the r resources and the marginal cost component of the state space. Given two labels L_1 and L_2 at the same node, the first eliminates the second, if $L_1 \leq L_2$. This is the dominance process, that is, states are compared and only Pareto optimal ones are kept. Below are some of the procedures that can be used to accelerate these algorithms.

Restricted networks. The solution process can use restricted networks to obtain a good initial set of columns. One can *a priori* select just a subset of the best arcs or dynamically ignore some arcs and nodes according to the dual information. Similar to the scaling algorithms used in network theory (see Ahuja et al. [1]), the discarded arcs can be reintroduced subsequently. Examples of such strategies can be found in Barnhart et al. [2], Dumas et al. [20], and Ioachim et al. [30], to name a few. If some nodes and arcs cannot be part of any negative marginal cost path, these can be eliminated from the current computations. For the VRPTW for example, a threshold value on the dual variable at a customer node can be computed *a priori* as the minimum insertion cost of that customer within a route. Discarding a customer node also eliminates all its incident arcs.

State space reduction. In this special case of network restriction, time resources for example, can be temporarily computed to the nearest ten minutes instead of the nearest minute (see Gamache et al. [24]). The range of the new units is smaller compared to the initial ones which means that different levels of consumption are now represented by the same value. This increases the dominance between labels and reduces the solution time. Additionally, dominance can be increased by making use of some tolerance parameters.

Node treatment. Given that in several applications such as the classical VRPTW, the networks are time oriented but may contain cycles, it is useful to consider the nodes in increasing order of a time resource. (This heuristic

treatment of the nodes has later conducted to the development of the more efficient concept of label order rather than node order; see Desrosiers et al. [16]).

Pulling versus reaching. While algorithmic complexity is the same for these two strategies, from an implementation point of view it is more efficient to pull to a node all the labels from predecessor nodes so that the dominance process is only applied once at this node while considering all its labels (see Desrochers and Soumis [13]).

Marginal cost interval reduction. As proposed in Crainic and Rousseau [6] for the airline crew scheduling problem, interval reduction can be applied to the marginal cost component since column generators are only looking for negative marginal cost paths.

Maximum size of the label list. In order to control the combinatorial aspect of the shortest path dynamic programming algorithm, a maximum number of labels can be retained at each node of the networks.

Partial pricing of the commodity networks. Since the simplex method does not require the selection of the most negative marginal cost variable, it is often beneficial to consider only some of the commodity networks each time the pricing out phase is called. The rationale for this is that all column generators use the same dual variables at a given iteration, hence the generators tend to produce columns covering the same tasks. Considering the results of the previous iterations, non promising commodity networks can temporarily be discarded. The reduction in the solution time per iteration may easily compensate for the increase in the total number of iterations required. This is particularly important when the number of commodities is rather large, as is the case for rostering problems where it equals the number of employees (typically, several hundreds). Gamache et al. [24] used this approach successfully to produce flight attendant rosters.

Multiple paths. A well known strategy to accelerate column generation is to return to the coordinator many negative marginal cost columns at each iteration. This is even more beneficial if these replicate the structure of an optimal integer solution. In many VRCS problems for example, solutions are made of task-disjoint columns. This can be achieved by using greedy strategies to select the columns. In the context of the analytic center cutting plane method, Goffin and Vial [27] have recently shown that the performance of this decomposition method is mathematically related to the variance-covariance matrix of the selected columns: the performance increases with the selection of non-correlated columns.

Subproblem aggregation. A very popular approach when the number of subproblems is large consists of aggregating identical subproblems (see Desaulniers, Lavigne, and Soumis [11]). This diminishes the chances of generating the same columns and reduces the number of subproblems to solve at each iteration.

Subtour elimination. As mentioned early in this section, the elementary path condition has been relaxed so that cycles can occur within a path. Small subtours of the form (i, j, i) , called *2-cycles*, can however easily be eliminated. An early implementation of this method for the VRPTW can be found in Kolen, Rinnooy Kan, and Trienekens [33].

Path composition. In certain cases, it is known that optimal or near optimal solutions comprise columns with only a very small number of tasks, say 2 to 5, with a large proportion of columns involving just 2 or 3. In this case, it is preferable to sequentially generate columns with 2 or 3 tasks, and generate new ones involving up to 5 tasks only toward the end of the solution process. This can be generalized to other column structures. In the air crew pairing problem for example, it might be interesting to sequentially generate pairings with an increasing number of working days (see Crainic and Rousseau [6]).

14.5 MASTER PROBLEM STRATEGIES

As previously stated, the simplex method is used to solve the linear relaxation of the generalized set partitioning problem for the current columns. The primary role of the master problem is to provide the dual variables used for pricing out new columns. Its secondary role is to act as a lower bounding procedure for the branch-and-bound tree. Accelerating techniques for the master phase are as follows.

Column elimination. When the number of columns in the master problem becomes too large, a subset of the nonbasic columns can be removed according to their current marginal cost or other criteria (for instance, columns that have not entered the master problem's basis in the last n column generation iterations, where n is a predefined parameter). To ensure proper convergence of the column generation process, this procedure should not be performed too often and a certain number of nonbasic columns should remain in the master problem after column removal.

Dynamic constraints. When the master problem contains a large number of constraints, it may be possible to relax some of them and reintroduce them when needed (that is, when the solution of a linear relaxation violates them). This is widely used for non convex constraints such as the integrality requirements. Such a strategy is also advantageous when the relaxed constraints have a low probability of being violated. Furthermore, this approach can be used as a heuristic to handle certain nonlinear constraints. For instance, in a railcar assignment application (Lingaya et al. [34]), nonlinear constraints of the form $\prod_i F_i(X) = 0$, where X is the vector of decision variables and the functions F_i are linear, were relaxed and evaluated after the solution of each linear relaxation. When such a constraint was violated, a heuristic branching decision setting function $F_i(X) = 0$ for a selected i was imposed to ensure its satisfaction.

Covering versus partitioning. In many VRCS problems, allowing over-covering of tasks that need to be covered exactly, that is changing the task covering constraints sign from “equal to” to “greater than or equal to” (e.g., for the VRPTW, replacing $= 1$ by ≥ 1 , for the number of times a customer is visited) has no impact on the optimal solution. This is true if each additional column covering a subset of the tasks already covered by another column is feasible and less costly than the original column (see Desrochers, Desrosiers, and Solomon [15]). Computationally, the linear relaxation of a set covering problem is easier to solve than that of a set partitioning problem. Hence, over-covering can be used as a heuristic method to rapidly compute good estimates of the dual variables.

Perturbation. Surplus variables can be used to replicate over-covering of the tasks to perform. Additionally, slack variables can be of interest for under-covering. Limiting these slack and surplus variables to small intervals gives a perturbation strategy of the right-hand side of the task-covering constraints. Moreover, the solution of the perturbed problem provides a valid lower bound for the branch-and-bound process. This and the prior technique were illustrated by Desaulniers et al. [12] for a crew pairing application.

Stabilization. This is the next step to the above perturbation strategy. Given *a priori* interval estimates of the dual values, this information can be easily incorporated into the master problem structure by assigning these lower and upper bounds as the cost coefficients of the slack and surplus variables (du Merle et al. [21]). Restricting the dual variables is equivalent to relaxing the primal problem so that, again, the stabilized problem also provides a valid lower bound for the branch-and-bound process. It can be observed here that the set covering problem, as compared to the set partitioning problem, halves the dual space and therefore produces a powerful restriction of it.

Multipliers adjustment. Any heuristic for the adjustment of the dual variables can be useful. One of these is Lagrangian relaxation applied to the master problem. For instance, in Kohl and Madsen [32], the optimal multipliers are found using a method exploiting the benefits of subgradient and bundle methods.

Competing algorithms. When the master problem is highly degenerated, it is often better to use the dual simplex algorithm rather than the primal simplex algorithm. Other algorithms, such as barrier and interior point methods, may also be available to solve the master problem. Therefore, a heuristic can be used at each iteration of the column generation process to select which algorithm should be applied to solve the current master problem. This heuristic can even trigger a change of algorithms if the algorithm first chosen does not perform as anticipated.

14.6 BRANCH-AND-BOUND STRATEGIES

This section examines methods developed for various components of a branch-and-bound approach: a bounding procedure to evaluate the potential of each node in the search tree, a way to explore the search tree, a set of rules to fathom unwanted branch-and-bound nodes, and a branching procedure to separate the feasible region of the unfathomed nodes. Each of these can benefit from the use of accelerating techniques such as the ones proposed below. Moreover, the next strategy proved fundamental in obtaining integer solutions.

Dynamic column generation. In current practice, columns are generated at the root node of the tree, and then used throughout. Yet, as illustrated by the results obtained by Hoffman and Padberg [29] on airline crew pairing problems, and Caprara et al. [4] on rail crew pairing problems, finding quality integer solutions from this static set of columns is often difficult. This is because this *a priori* set is not large enough to force the closure of the integrality gap below 1%. Dynamic column generation at each node of the search tree introduces new columns that are instrumental in completing a partial integer solution.

Depth first search. It is well known that depth first exploration, when combined to an appropriate branching strategy, usually provides a good first integer solution. The upper bound derived from this solution can then be used to fathom several of the unexplored nodes. When solution time is of prime importance, the depth first search is often performed without backtracking.

Search stopping rule. Once a good solution has been found, a maximum amount of additional computer time to use and/or a maximum number of additional branch-and-bound nodes to explore can be imposed to limit overall solution time.

Early branching. In order to avoid the well-known tailing off behavior of column generation, a heuristic rule can be devised to prematurely stop the linear relaxation solution process, for example, when the value of the objective function does not improve sufficiently in a given number of master problem iterations. In this case, the approximate linear relaxation solution does not necessarily provide a lower bound. However, since such a bound is not needed when using a depth first strategy without backtracking, branching decisions can be imposed to define a new restricted master problem to solve. In this case, the value of the objective function can still decrease at a son node (see Gamache et al. [24]).

Note that, using the appropriate Lagrangian function, a lower bound can still be computed when the column generation process is stopped prematurely (see Desaulniers et al. [9]). However, that bound may not be as good as the one provided by the master problem linear relaxation optimal value which could be obtained by completing the column generation process. This tradeoff between bound quality and computational time also occurs when applying Lagrangian relaxation since it is very difficult in practice to find a null subgradient.

Heuristic cutting plane separation algorithm. To obtain a tighter lower bound at each node of the search tree, it might be possible to add violated cutting planes to the corresponding linear relaxation. Since the identification of violated cuts can be quite expensive, it might be preferable to use a heuristic separation algorithm that will speed up the search process but will not guarantee the identification of such a cut if one exists. For instance, Kohl et al. [31] have used a greedy separation algorithm to identify the violated 2-path cuts for the VRPTW.

Fathoming. Fathoming branch-and-bound nodes can be done using tolerance parameters. For instance, a node can be fathomed if its lower bound is greater than the cost of the best integer solution found so far minus a given tolerance value.

Rounding to 1. Given a fractional solution to the linear relaxation of the generalized set partitioning problem, decisions are imposed by fixing at 1 the corresponding selected variables, and removing incompatible columns accordingly. This is generally done for variables close to 1. However, in an air rostering application (see Gamache and Soumis [23]), it was valuable to fix at 1 all variables with a value greater than or equal to 0.6. In this environment, the right-hand side of the task-covering constraints is usually greater than 1 which provides more flexibility and reduces the risk of making a poor decision. Note that, when time permits, it is also possible to fix at 1 selected arc flow variables or combinations of them. These less aggressive decisions can often be handled at the subproblem level.

Rounding to the next integer. A similar strategy can be used when aggregated subproblems are used and the right-hand sides of the task-covering constraints are greater than 1. In this case which may arise for instance in a locomotive assignment problem (see Ziarati et al. [40]), selected master problem variables can be rounded to the next integer.

Multiple decisions. As seen above, many decisions can be taken simultaneously to reduce the size of the search tree in a depth first approach (see Gamache et al. [24], and Ziarati et al. [40]). These decisions can be of various types and should be chosen to be complementary in order to avoid undesirable infeasibility.

Competing branching strategies. When multiple branching strategies are available to derive integer solutions, they can be ordered a priori and, at any branching node, the strategy selected is the first applicable at that node when the order is followed. Additional flexibility can be achieved by using a heuristic scoring system during the solution process that will order the different strategies at each node of the search tree. Applications with multiple branching strategies can be found in Ziarati et al. [40] and Cordeau et al. [5].

14.7 POST-OPTIMIZER STRATEGIES

In this last phase, given an integer solution to a problem, it is sometimes still possible to improve it in several ways. Here are three heuristics that can be used:

Local reoptimizations. Well known improvement procedures such as the 2- and 3-opt interchange heuristics as well as metaheuristics such as tabu search, variable neighborhood search and simulated annealing can be applied in an attempt to improve the current integer solution.

Partial reoptimization. As previously discussed, very large-scale problems are often solved using partitioning and rolling horizon strategies. By successively re-considering some new portions of a problem, one can improve the value of the objective function.

Local relaxations. This has been used specifically for the dial-a-ride application. Bus itineraries were obtained using hard time windows at the pickup and delivery locations. By using local reoptimizations with soft time windows on certain itineraries, i.e., relaxing the hard time window restrictions, it was possible to eliminate several route patterns disliked by the schedulers. It should be mentioned that in this kind of application, one of the main goals is to reduce the number of complaints from both the schedulers and the clients.

14.8 COMPUTATIONAL BENEFITS

The literature cited here highlights the importance of using acceleration methods. Their use has not only resulted in efficiency gains, but more importantly has produced solutions that would have otherwise been out of reach due to practical problem size and complexity. Moreover, generic techniques can be tailored to specific applications to get better results. Note that in many cases, the accelerated process remains optimal. Nevertheless, some procedures destroy the optimality of the branch-and-price approach.

While the overall results obtained by different researchers speak to the benefits of acceleration, specific comparisons illustrating actual savings are scant. The only attempt that we are aware of is that by Gamache et al. [24] for the air crew rostering problem. Embedding many heuristics described above in the proposed method, the authors have accelerated the standard column generation by three orders of magnitude and obtained excellent quality solutions. In test problems where comparison was possible, the optimality gap was no wider than 0.6%.

14.9 CONCLUSIONS

We have discussed accelerating strategies for column generation directed at the solution of vehicle routing and crew scheduling problems. We discussed techniques for each of the five phases of the solution process. In practice, these techniques have proved instrumental for the viability of this optimization approach. Since the advantages of individual procedures may vary with the

application at hand, implementation should proceed in a step-by-step fashion. This way, the incremental benefit of each method may become more readily apparent.

Concerning the LP aspect of the method, accelerating strategies are now part of all commercial simplex software packages. Since column generation is mainly based on the same mathematical structure, it is not surprising that good implementations of column generation approaches make intensive use of similar heuristic strategies.

In addition to these more well known methods, substantial progress has recently been made for the IP aspect. A case in point is dynamic column generation at each branching node. Indeed, the results obtained by generating new columns at each node show the advantages of using such techniques for speed and solution quality. They further suggest that it may be more important to refresh the set of columns than to design a better branching strategy. We believe these specific results will be shown to be valid in most cases by future research.

References

- [1] R.K. Ahuja, T. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice-Hall, 1993.
- [2] C. Barnhart, L. Hatay, and E.L. Johnson. Deadhead Selection for the Long-Haul Crew Pairing Problem. *Operations Research*, 43:491–499, 1995.
- [3] C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, and P.H. Vance. Branch-and-Price: Column Generation for Solving Huge Integer Programs. *Operations Research*, 46:316–329, 1998.
- [4] A. Caprara, M. Fischetti, and P. Toth. A Heuristic Algorithm for the Set Covering Problem. *Lecture Notes in Computer Science*, 1084:72–84, 1996.
- [5] J.-F. Cordeau, G. Desaulniers, N. Lingaya, F. Soumis, and J. Desrosiers. Simultaneous Locomotive and Car Assignment at VIA Rail Canada. *Les Cahiers du GERAD*, G-98-61, Montréal, 1998.
- [6] T.-G. Crainic and J.-M. Rousseau. The Column Generation Principle and the Airline Crew Scheduling Problem. *INFOR*, 25:136–151, 1987.
- [7] G.B. Dantzig and P. Wolfe. Decomposition Principle for Linear Programs. *Operations Research*, 8:101–111, 1960.
- [8] G. Desaulniers, J. Desrosiers, Y. Dumas, S. Marc, B. Rioux, M.M. Solomon, and F. Soumis. Crew Pairing at Air France. *European Journal of Operational Research*, 97:245–259, 1997.

- [9] G. Desaulniers, J. Desrosiers, I. Ioachim, M.M. Solomon, F. Soumis, and D. Villeneuve. *A Unified Framework for Deterministic Time Constrained Vehicle Routing and Crew Scheduling Problems*. In: *Fleet Management and Logistics*, T.G. Crainic and G. Laporte, editors, pages 57–93, Kluwer, 1998.
- [10] G. Desaulniers, J. Desrosiers, M.M. Solomon, and F. Soumis. Daily Aircraft Routing and Scheduling. *Management Science*, 43:841–855, 1997.
- [11] G. Desaulniers, J. Lavigne, and F. Soumis. Multi-Depot Vehicle Scheduling Problems with Time Windows and Waiting Costs. *European Journal of Operational Research*, 111:479–494, 1998.
- [12] G. Desaulniers, J. Desrosiers, A. Lasry, and M. M. Solomon. Crew Pairing for a Regional Carrier. *Lecture Notes in Economics and Mathematical Systems*, 471:19–41, 1999.
- [13] M. Desrochers and F. Soumis. A Reoptimization Algorithm for the Shortest Path Problem with Time Windows. *European Journal of Operational Research*, 35:242–254, 1988.
- [14] M. Desrochers and F. Soumis. A Column Generation Approach to the Urban Transit Crew Scheduling Problem. *Transportation Science*, 23:1–13, 1989.
- [15] M. Desrochers, J. Desrosiers, and M.M. Solomon. A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows. *Operations Research*, 40:342–354, 1992.
- [16] J. Desrosiers, Y. Dumas, M.M. Solomon, and F. Soumis. Time Constrained Routing and Scheduling. In: *Network Routing*, M.O. Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser, editors, *Handbooks in Operations Research and Management Science*, vol. 8, pages 35–139, Elsevier, 1995.
- [17] J. Desrosiers, Y. Dumas, and F. Soumis. The Multiple Vehicle Dial-a-Ride Problem. *Lecture Notes in Economics and Mathematical Systems*, 308:15–27, 1988.
- [18] J. Desrosiers, F. Soumis, and M. Desrochers. Routing with Time Windows by Column Generation. *Networks*, 14:545–565, 1984.
- [19] M. Dror. Note on the Complexity of the Shortest Path Models for Column Generation in VRPTW. *Operations Research*, 42:977–978, 1994.
- [20] Y. Dumas, J. Desrosiers, and F. Soumis. The Pickup and Delivery Problem with Time Windows. *European Journal of Operational Research*, 54:7–22, 1991.
- [21] O. du Merle, D. Villeneuve, J. Desrosiers, and P. Hansen. Stabilized Column Generation. *Discrete Mathematics*, 194:229–237, 1999.

- [22] L.R. Ford, and D.R. Fulkerson. A Suggested Computation for Maximal Multi-commodity Network Flows. *Management Science*, 5:97–101, 1958.
- [23] M. Gamache and F. Soumis. A Method for Optimally Solving the Rostering Problem. In: *Operations Research in the Airline Industry*, G. Yu, editor, pages 124–157, Kluwer, 1998.
- [24] M. Gamache, F. Soumis, G. Marquis, and J. Desrosiers. A Column Generation Approach for Large Scale Aircrew Rostering Problems. *Operations Research*, 47:247–263, 1999.
- [25] A.M. Geoffrion. Lagrangean Relaxation for Integer Programming. *Mathematical Programming Studies*, 2:82–114, 1974.
- [26] P.C. Gilmore and R.E. Gomory. A Linear Programming Approach to the Cutting-Stock Problem. *Operations Research*, 9:849–859, 1961.
- [27] J.-L. Goffin and J.-P. Vial. Multiple Cuts in the Analytic Center Cutting Plane Method. *SIAM Journal on Optimization*, 11:266–288, 2000.
- [28] C. Hane, C. Barnhart, E.L. Johnson, R. Marsten, G.L. Nemhauser, and G. Sigismondi. The Fleet Assignment Problem: Solving a Large-Scale Integer Program. *Mathematical Programming*, 70:211–232, 1995.
- [29] K.L. Hoffman and M. Padberg. Solving Airline Crew Scheduling Problems by Branch-and-Cut. *Management Science*, 39:657–682, 1993.
- [30] I. Ioachim, J. Desrosiers, Y. Dumas, M.M. Solomon, and Daniel Villeneuve. A Request Clustering Algorithm for Door-to-Door Handicapped Transportation. *Transportation Science*, 29:63–79, 1995.
- [31] N. Kohl, J. Desrosiers, O.B.G. Madsen, M.M. Solomon, and F. Soumis. 2-Path Cuts for the Vehicle Routing Problem with Time Windows. *Transportation Science*, 33:101–116, 1999.
- [32] N. Kohl and O.B.G. Madsen. An Optimization Algorithm for the Vehicle Routing Problem with Time Windows Based on Lagrangian Relaxation. *Operations Research*, 45:395–406, 1997.
- [33] A.W.J. Kolen, A.H.G. Rinnooy Kan, and H.W.J.M. Trienekens. Vehicle Routing with Time Windows. *Operations Research*, 35:266–273, 1987.
- [34] N. Lingaya, J.-F. Cordeau, G. Desaulniers, J. Desrosiers, and F. Soumis. Operational Car Assignment at VIA Rail Canada. *Les Cahiers du GERAD*, G-2000-55, Montréal, 2000.
- [35] A. Mingozzi, M.A. Boschetti, S. Ricciardelli, and L. Bianco. A Set Partitioning Approach to the Crew Scheduling Problem. *Operations Research*, 47:873–888, 1999.

- [36] C. Ribeiro and F. Soumis. A Column Generation Approach to the Multiple Depot Vehicle Scheduling Problem. *Operations Research*, 42:41–53, 1994.
- [37] J.M. Rousseau and J. Desrosiers. Results Obtained with CREW-OPT, a Column Generation Method for Transit Crew Scheduling. *Lecture Notes in Economics and Mathematical Systems*, 430:349–358, 1995.
- [38] M.M. Solomon. Algorithms for the Vehicle Routing and Scheduling Problem with Time Window Constraints. *Operations Research*, 35:254–265, 1987.
- [39] K. Ziarati, F. Soumis, J. Desrosiers, S. Gélinas, and A. Saintonge. Locomotive Assignment with Heterogeneous Consists at CN North America. *European Journal of Operational Research*, 97:281–292, 1997.
- [40] K. Ziarati, J. Desrosiers, F. Soumis, and M.M. Solomon. A Branch-First, Cut-Second Approach for Locomotive Assignment. *Management Science*, 45:1156–1168, 1999.

15

GRASP: AN ANNOTATED BIBLIOGRAPHY

Paola Festa¹ and Mauricio G.C. Resende²

¹Mathematics and Computer Science Department
University of Salerno, 84081 Baronissi (SA), Italy
paofer@unisa.it

²Information Sciences Research Center
AT&T Labs Research, 180 Park Avenue, Room C241
Florham Park, NJ 07932 USA
mgcr@research.att.com

Abstract: A greedy randomized adaptive search procedure (GRASP) is a metaheuristic for combinatorial optimization. It is a multi-start or iterative process, in which each GRASP iteration consists of two phases, a construction phase, in which a feasible solution is produced, and a local search phase, in which a local optimum in the neighborhood of the constructed solution is sought. Since 1989, numerous papers on the basic aspects of GRASP, as well as enhancements to the basic metaheuristic have appeared in the literature. GRASP has been applied to a wide range of combinatorial optimization problems, ranging from scheduling and routing to drawing and turbine balancing. This paper is an annotated bibliography of the GRASP literature from 1989 to 2001.

15.1 INTRODUCTION

Optimization problems that involve a large finite number of alternatives often arise in the private and public sectors of the economy. In these problems, given a finite solution set X and a real-valued function $f : X \rightarrow \mathbb{R}$, one seeks a solution $x^* \in X$ with $f(x^*) \leq f(x)$, $\forall x \in X$. Common examples include designing efficient telecommunication networks, constructing cost effective airline crew schedules, and producing efficient routes for waste management pickup. To find the optimal solution in a combinatorial optimization problem it is theoretically possible to enumerate the solutions and evaluate each with respect to

the stated objective. However, in practice, it is often infeasible to follow such a strategy of complete enumeration because the number of combinations often grows exponentially with the size of problem.

Much work has been done over the last five decades to develop optimal seeking methods that do not explicitly require an examination of each alternative. This research has given rise to the field of *combinatorial optimization* (see Papadimitriou and Steiglitz [1]), and an increasing capability to solve ever larger real-world problems. Nevertheless, most problems found in industry and government are either computationally intractable by their nature, or sufficiently large so as to preclude the use of exact algorithms. In such cases, heuristic methods are usually employed to find good, but not necessarily guaranteed optimal solutions. The effectiveness of these methods depends upon their ability to adapt to a particular realization, avoid entrapment at local optima, and exploit the basic structure of the problem. Building on these notions, various heuristic search techniques have been developed that have demonstrably improved our ability to obtain good solutions to difficult combinatorial optimization problems. The most promising of such techniques include simulated annealing (Kirkpatrick [2]), tabu search (Glover [3,4], Glover and Laguna [5]), genetic algorithms (Goldberg [6]), variable neighborhood search (Hansen and Mladenović [7]), and GRASP (Greedy Randomized Adaptive Search Procedures) (Feo and Resende [8,9]).

A GRASP is a multi-start or iterative process (Lin and Kernighan [10]), in which each GRASP iteration consists of two phases, a construction phase, in which a feasible solution is produced, and a local search phase, in which a local optimum in the neighborhood of the constructed solution is sought. The best overall solution is kept as the result.

In the construction phase, a feasible solution is iteratively constructed, one element at a time. The basic GRASP construction phase is similar to the semi-greedy heuristic proposed independently by Hart and Shogan [11]. At each construction iteration, the choice of the next element to be added is determined by ordering all candidate elements (i.e. those that can be added to the solution) in a candidate list C with respect to a greedy function $g : C \rightarrow \mathbb{R}$. This function measures the (myopic) benefit of selecting each element. The heuristic is adaptive because the benefits associated with every element are updated at each iteration of the construction phase to reflect the changes brought on by the selection of the previous element. The probabilistic component of a GRASP is characterized by randomly choosing one of the best candidates in the list, but not necessarily the top candidate. The list of best candidates is called the *restricted candidate list* (RCL). This choice technique allows for different solutions to be obtained at each GRASP iteration, but does not necessarily compromise the power of the adaptive greedy component of the method.

As is the case for many deterministic methods, the solutions generated by a GRASP construction are not guaranteed to be locally optimal with respect to simple neighborhood definitions. Hence, it is almost always beneficial to apply a local search to attempt to improve each constructed solution. A local search

algorithm works in an iterative fashion by successively replacing the current solution by a better solution in the neighborhood of the current solution. It terminates when no better solution is found in the neighborhood. The *neighborhood structure* N for a problem P relates a solution s of the problem to a subset of solutions $N(s)$. A solution s is said to be *locally optimal* if there is no better solution in $N(s)$. The key to success for a local search algorithm consists of the suitable choice of a neighborhood structure, efficient neighborhood search techniques, and the starting solution.

While such local optimization procedures can require exponential time (Johnson, Papadimitriou, and Yannakakis [12]) from an arbitrary starting point, empirically their efficiency significantly improves as the initial solution improves. The result is that often many GRASP solutions are generated in the same amount of time required for the local optimization procedure to converge from a single random start. Furthermore, the best of these GRASP solutions is generally significantly better than the single solution obtained from a random starting point.

It is difficult to formally analyze the quality of solution values found by using the GRASP methodology. However, there is an intuitive justification that views GRASP as a repetitive sampling technique. Each GRASP iteration produces a sample solution from an unknown distribution of all obtainable results. The mean and variance of the distribution are functions of the restrictive nature of the candidate list. For example, if the cardinality of the restricted candidate list is limited to one, then only one solution will be produced and the variance of the distribution will be zero. Given an effective greedy function, the mean solution value in this case should be good, but probably suboptimal. If a less restrictive cardinality limit is imposed, many different solutions will be produced implying a larger variance. Since the greedy function is more compromised in this case, the mean solution value should degrade. Intuitively, however, by order statistics and the fact that the samples are randomly produced, the best value found should outperform the mean value. Indeed, often the best solutions sampled are optimal.

An especially appealing characteristic of GRASP is the ease with which it can be implemented. Few parameters need to be set and tuned, and therefore development can focus on implementing efficient data structures to assure quick GRASP iterations. Finally, GRASP can be trivially implemented in parallel. Each processor can be initialized with its own copy of the procedure, the instance data, and an independent random number sequence. The GRASP iterations are then performed in parallel with only a single global variable required to store the best solution found over all processors.

In this article, we provide an annotated bibliography of the GRASP literature up to early 2001. This document contains references related to GRASP that have either appeared in the literature or as technical reports. We first look at tutorials and surveys. Papers that propose enhancements to the basic heuristic are considered next. Following that, we examine GRASP as a component of a hybrid metaheuristic. Parallelization of GRASP and GRASP source code

follow. The paper concludes with a literature review of operations research and computer science applications of GRASP as well as industrial applications.

If the reader is aware of any uncited reference, incorrectly cited reference, or update to a cited reference, please contact the authors.

- [1] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, 1982.

A classic book on combinatorial optimization.

- [2] S. Kirkpatrick. Optimization by Simulated Annealing: Quantitative Studies. *Journal of Statistical Physics*, 34:975–986, 1984.

Description of the simulated annealing metaheuristic.

- [3] F. Glover. Tabu search – Part I. *ORSA Journal on Computing*, 1:190–206, 1989.

Description of the tabu search metaheuristic.

- [4] F. Glover. Tabu search – Part II. *ORSA Journal on Computing*, 2:4–32, 1990.

Description of the tabu search metaheuristic.

- [5] F. Glover and M. Laguna. *Tabu Search*. Kluwer, 1997.

Book on metaheuristics and in particular tabu search.

- [6] D.E Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.

Book on genetic algorithms.

- [7] P. Hansen and N. Mladenović. An Introduction to Variable Neighborhood Search. In: S. Voss, S. Martello, I.H. Osman, and C. Roucairol, editors, *Meta-Heuristics, Advances and Trends in Local Search Paradigms for Optimization*, pages 433–458. Kluwer, 1998.

Description of the variable neighborhood search metaheuristic.

- [8] T.A. Feo and M.G.C. Resende. A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem. *Operations Research Letters*, 8:67–71, 1989.

This is the first paper to explicitly describe a GRASP. See page 365.

- [9] T.A. Feo and M.G.C. Resende. Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, 6:109–133, 1995.

An early survey of GRASP. See page 329.

- [10] S. Lin and B.W. Kernighan. An Effective Heuristic Algorithm for the Traveling-Salesman Problem. *Operations Research*, 21:498–516, 1973.

An early random multistart local search technique.

- [11] J.P. Hart and A.W. Shogan. Semi-Greedy Heuristics: An Empirical Study. *Operations Research Letters*, 6:107–114, 1987.

This paper presents a randomized greedy heuristic, called semi-greedy heuristic.

- [12] D.S. Johnson, C.H. Papadimitriou, and M. Yannakakis. How Easy is Local Search? *Journal of Computer and System Sciences*, 17:79–100, 1988.

Study of the computational complexity of local search.

15.2 TUTORIALS AND SURVEYS

Since GRASP is a relatively recent optimization method, tutorials and surveys on this subject are limited. Below are a few examples of introductory material on GRASP.

- [13] T.A. Feo and M.G.C. Resende. Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, 6:109–133, 1995.

In this tutorial paper, the authors define the various components comprising a GRASP. A general trivial implementation of GRASP on a parallel computer is also discussed. The GRASP literature until 1994 is surveyed.

- [14] J.L. González. GRASP. In: *Heuristic Optimization and Neural Networks in Operations Management and Engineering*, A. Díaz, editor, pages 143–161. Editorial Paraninfo, 1996.

This is a chapter on GRASP in a book on heuristic procedures for optimization. In Spanish.

- [15] M.G.C. Resende. Greedy Randomized Adaptive Search Procedures (GRASP). To appear in: *Encyclopedia of Optimization*, Kluwer, 2001.

This paper surveys greedy randomized adaptive search procedures. The basic GRASP is explained in detail and enhancements to the basic procedure are described. Several applications of GRASP are reported, showing how this method can find good approximate solutions to operations research problems and industrial applications.

- [16] C.M.D. Silveira. GRASP – A Heuristic for Solving Combinatorial Optimization Problems. Technical Report, Institute of Informatics, Federal University of Rio Grande do Sul, Porto Alegre, 1999.

The aim of this report is to provide an exhaustive description of the features of GRASP as a metaheuristic method for solving hard combinatorial optimization problems. The various components comprising a generic GRASP are defined and it is also shown through

examples how to develop GRASPs for combinatorial optimization problems. In Portuguese.

- [17] L.S. Pitsoulis and M.G.C. Resende. Greedy Randomized Adaptive Search Procedures. In: *Handbook of Applied Optimization*, P.M. Pardalos and M.G.C. Resende, editors, Oxford University Press, 2001.

This chapter surveys GRASP. Multi-start heuristics are seen as a way to apply local search to solve combinatorial optimization problems. GRASP is shown to, in some ways, improve upon greedy or random multi-start procedures. Enhancements to GRASP, such as reactive GRASP, hybrid GRASP, and use of long-term memory are discussed. The parallelization of GRASP is also considered. The chapter ends with a survey of GRASP for solving problems in logic, assignment, and location.

15.3 ENHANCEMENTS TO THE BASIC GRASP

The standard (or basic) GRASP consists of repeated applications of GRASP construction (using a fixed candidate list strategy), followed by hill climbing local search. Enhancements to this basic GRASP are found in the following papers.

- [18] J.L. Bresina. Heuristic-Biased Stochastic Sampling. In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 271–278, 1996.

This paper presents a generalization of iterative sampling called Heuristic-Biased Stochastic Sampling (HBSS). The two search techniques have the same overall control structure. The difference lies in how a choice is made at each decision point. HBSS uses a given search heuristic to focus its exploration. The degree of focusing is determined by a chosen *bias function* that reflects the confidence one has in the heuristic's accuracy. This methodology can be directly applied in a GRASP construction phase, by biasing the selection of RCL elements to favor those with higher greedy function values.

- [19] J. Mockus, E. Eddy, A. Mockus, L. Mockus, and G.V. Reklaitis. *Bayesian Discrete and Global Optimization*. Kluwer, 1997.

This book describes the Bayesian approach to discrete optimization. A Bayesian heuristic algorithm version of GRASP is described.

- [20] J.B. Atkinson. A Greedy Randomised Search Heuristic for Time-Constrained Vehicle Scheduling and the Incorporation of a Learning Strategy. *Journal of the Operational Research Society*, 49:700–708, 1998.

Two forms of adaptive search called *local* and *global* adaptation are identified. In both search techniques, the greedy function takes into account a quantity that measures heuristically the quality of the partial solution. While in local adaptation the decisions made within a particular run influence only the subsequent performance of the

heuristic, global adaptation involves making decisions that affect the performance of the heuristic in subsequent runs. See page 342.

- [21] C. Fleurent and F. Glover. Improved Constructive Multistart Strategies for the Quadratic Assignment Problem Using Adaptive Memory. *INFORMS Journal on Computing*, 11:198–204, 1999.

Study of alternatives to memoryless multistart approaches like GRASP is conducted for the quadratic assignment problem. Adaptive memory strategies retain and analyze features of selected solutions in order to provide a basis for improving future executions of the constructive process. The authors show that the most effective strategies are tabu search methods, which use memory in both improving and constructive phases. See page 354.

- [22] M. Laguna and R. Martí. GRASP and Path Relinking for 2-Layer Straight Line Crossing Minimization. *INFORMS Journal on Computing*, 11:44–52, 1999.

This paper incorporates to GRASP a path relinking strategy to search for improved outcomes. Path relinking generates new solutions by exploring trajectories connecting high quality solutions. Starting from an *initiating solution*, path relinking generates a path in the neighborhood space that leads toward the other solutions, called *guiding solutions*. This is accomplished by selecting moves that introduce attributes contained in the guiding solutions. See page 363.

- [23] M. Prais. *Parameter Variation in GRASP Procedures*. Ph.D. Thesis, Department of Computer Sciences, Catholic University of Rio de Janeiro, Rio de Janeiro, 2000.

This thesis describes a GRASP for a matrix decomposition problem in TDMA traffic assignment. It proposes Reactive GRASP, a refinement of GRASP where the RCL parameter is adjusted dynamically to favor values that produce good solutions. Reactive GRASP is compared with other RCL strategies on matrix decomposition, set covering, maximum satisfiability and graph planarization.

- [24] M. Prais and C.C. Ribeiro. Reactive GRASP: An Application to a Matrix Decomposition Problem in TDMA Traffic Assignment. *INFORMS Journal on Computing*, 12:164–176, 2000.

A refinement of GRASP, called Reactive GRASP, is proposed. Instead of using a fixed value for the basic parameter that defines the restrictiveness of the candidate list during the construction phase, Reactive GRASP self-adjusts the parameter value according to the quality of the solutions previously found. See pages 355 and 362.

- [25] M. Prais and C.C. Ribeiro. Parameter Variation in GRASP Procedures. *Investigación Operativa*, 9:1–20, 2000.

The GRASP RCL parameter α that determines the size of the restricted candidate list can be adjusted, leading to different behavior of the GRASP implementation. This paper investigates four strategies for the variation of the parameter α : 1) reactive – α is self-adjusted along the iterations; 2) uniform – α is randomly chosen from a discrete uniform probability distribution; 3) hybrid – α is randomly chosen from a fixed probability distribution concentrated around the value corresponding to the purely greedy choice; 4) fixed – α has a fixed value close to the purely greedy choice. The reactive strategy is the most flexible and adherent to the characteristics of the specific problem to be solved. In Portuguese.

- [26] C.C. Ribeiro, E. Uchoa, and R.F. Werneck. A Hybrid GRASP with Perturbations and Adaptive Path-Relinking for the Steiner Problem in Graphs. Technical Report, Department of Computer Science, Catholic University of Rio de Janeiro, Rio de Janeiro, 2000.

This paper describes a hybrid GRASP with weight perturbations and adaptive path-relinking heuristic (HGP-PR) for the Steiner problem in graphs. In this multi-start approach, the greedy randomized construction phase of a GRASP is replaced by the combination of several construction heuristics with a weight perturbation strategy. A strategic oscillation scheme combining intensification and diversification elements is used for the perturbation of the original weights. The improvement phase circularly explores two different local search strategies. An adaptive path-relinking technique is applied to a set of elite solutions as an intensification strategy. Computational experiments on a large set of benchmark problems of three different classes are reported. The new heuristic HGP-PR outperformed other algorithms, obtaining consistently better or comparably good solutions for all classes of test problems.

15.4 GRASP IN HYBRID METAHEURISTICS

GRASP has been used in conjunction with other metaheuristics. The following papers illustrate these hybrid techniques.

- [27] M. Laguna and J.L. González-Velarde. A Search Heuristic for Just-in-Time Scheduling in Parallel Machines. *Journal of Intelligent Manufacturing*, 2:253–260, 1991.

The proposed hybrid metaheuristic combines elements of GRASP with elements of tabu search. See page 339.

- [28] M. Yagiura and T. Ibaraki. Genetic and Local Search Algorithms as Robust and Simple Optimization Tools. In: *Meta-Heuristics: Theory and Applications*, I.H. Osman and J.P. Kelly, editors, pages 63–82. Kluwer, 1996.

Various metaheuristics such as random multi-start local search (MLS) and genetic algorithm (GA) are implemented in this paper and their

performance compared. The objective of the authors is not to propose the most powerful technique, but to compare general tendencies of various algorithms. From their analysis it results that a GRASP type modification of MLS improves its performance and that GA combined with local search is quite effective if long computational time is allowed.

- [29] R. Colomé and D. Serra. Consumer Choice in Competitive Location Models: Formulations and Heuristics. To appear in: *Papers in Regional Science*.

The authors propose a hybrid GRASP which uses tabu search in the local search phase. See page 365.

- [30] H. Ramalhinho Lourenço, J.P. Paixão, and R. Portugal. Metaheuristics for the Bus-Driver Scheduling Problem. Technical Report, Department of Economics and Management, Universitat Pompeu Fabra, Barcelona, 1998.

GRASP is used in a genetic algorithm to implement a type of crossover called *perfect offspring*. See page 340.

- [31] H. Delmaire, J.A. Díaz, E. Fernández, and M. Ortega. Reactive GRASP and Tabu Search Based Heuristics for the Single Source Capacitated Plant Location Problem. *INFOR*, 37:194–225, 1999.

A hybrid heuristic is proposed. It embeds a reactive GRASP in a tabu search algorithm as a diversification method that provides elite candidate sets. Intensification is also done. See page 346.

- [32] R.K. Ahuja, J.B. Orlin, and A. Tiwari. A Greedy Genetic Algorithm for the Quadratic Assignment Problem. *Computers and Operations Research*, 27:917–934, 2000.

In this paper a hybrid genetic algorithm for the QAP is presented. GRASP is used to generate the initial population. See page 355.

- [33] M. Armony, J.G. Klincewicz, H. Luss, and M.B. Rosenwein. Design of Stacked Self-Healing Rings using a Genetic Algorithm. *Journal of Heuristics*, 6:85–105, 2000.

A genetic algorithm for design of stacked self-healing rings is proposed. The objective is to optimize the trade-off between the cost of connecting nodes to the ring and the cost of routing demand on multiple rings. The initial population of the genetic algorithm is made up of randomly generated solutions as well as solutions generated by a GRASP. Computational comparisons are made with a commercial integer programming package.

- [34] S. Abdinnour-Helm and S.W. Hadley. Tabu Search Based Heuristics for Multi-Floor Facility Layout. *International Journal of Production Research*, 38:365–383, 2000.

Two two-stage heuristics are proposed for solving the multi-floor facility layout problem. GRASP/TS applies a GRASP to find the initial layout and tabu search to refine the initial layout, based on total inter/intra-floor costs. Computational tests indicate that GRASP/TS compares favorably with heuristics that do not rely on exact algorithms.

- [35] C.C. Ribeiro, E. Uchoa, and R.F. Werneck. A Hybrid GRASP with Perturbations and Adaptive Path-Re-linking for the Steiner Problem in Graphs. Technical Report, Department of Computer Science, Catholic University of Rio de Janeiro, Rio de Janeiro, 2000.

This paper propose and describes a hybrid GRASP with weight perturbations and adaptive path-relinking heuristic (HGP-PR) for the Steiner problem in graphs. See page 332.

15.5 PARALLEL GRASP

GRASP can be easily implemented in parallel by dividing iterations among several processors. Other parallelization schemes have also been used to implement parallel GRASPs. The following papers exemplify parallel GRASP.

- [36] T.A. Feo, M.G.C. Resende, and S.H. Smith. A greedy Randomized Adaptive Search Procedure for Maximum Independent Set. *Operations Research*, 42:860–878, 1994.

A GRASP for approximately solving the maximum independent set problem is described. The proposed heuristic can be easily implemented in parallel by decomposing the problem into smaller subproblems, each defined by conditioning on vertices being in the solution. An implementation of this algorithm was tested on a MIMD computer with up to eight processors. Average linear speedup is observed. See page 347.

- [37] P.M. Pardalos, L. Pitsoulis, T. Mavridou, and M.G.C. Resende. Parallel Search for Combinatorial Optimization: Genetic Algorithms, Simulated Annealing and GRASP. *Lecture Notes in Computer Science*, 980:317–331, 1995.

This paper summarizes some parallel search techniques for approximating the global optimal solution of combinatorial optimization problems. For large-scale problems, one of the main limitations of heuristic search is its computational complexity. Therefore, efficient parallel implementation of search algorithms can significantly increase the size of the problems that can be solved.

- [38] P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. A Parallel GRASP Implementation for the Quadratic Assignment Problem. In: *Parallel Algorithms for Irregularly Structured Problems – Irregular'94*, A. Ferreira and J. Rolim, editors, pages 115–130, Kluwer, 1995.

Efficient parallel techniques for large-scale sparse quadratic assignment problems are discussed. The paper provides a detailed description of a parallel implementation on an MIMD computer of the sequential GRASP proposed by Li, Pardalos, and Resende (1994) for solving the QAP. The GRASP iterations are distributed among the processors. Each processor is given its own input data and random number sequence and are run independently. A shared global variable stores the value of the incumbent solution.

- [39] P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. A Parallel GRASP for MAX-SAT Problems. *Lecture Notes in Computer Science*, 1184:575–585, 1996.

A parallel GRASP for weighted maximum satisfiability (MAX-SAT) problem is proposed. The GRASP is based on the serial GRASP presented by Resende, Pitsoulis, and Pardalos (1997). The parallel implementation distributes the GRASP iterations among several processors operating in parallel, avoiding that two processors have as input the same random number generator seed. The best solution found among all processors is identified and used as solution of the problem.

- [40] A.C.F. Alvim. *Parallelization Strategies for the Metaheuristic GRASP*. Master's Thesis, Department of Computer Science, Catholic University of Rio de Janeiro, Rio de Janeiro, 1998.

Two parallelization strategies for GRASP are discussed and compared: parallelization by distributing GRASP iterations and parallelization by varying the GRASP random parameter α . Both strategies are adapted to several parallel computation models, such as MPI (Message Passing Interface) and PVM (Parallel Virtual Machine). In Portuguese.

- [41] A.C.F. Alvim and C.C. Ribeiro. Load Balancing in the Parallelization of the Metaheuristic GRASP. In: *Proceedings of the X Brazilian Symposium on Computer Architecture*, pages 279–282, Búzios, 1998.

Two parallelization strategies for GRASP are compared. The difference between the two strategies concerns the way in which data is partitioned: pre-scheduled (static load balancing) or self-scheduled (dynamic load balancing). The strategies have been tested considering an application to optimal traffic assignment in TDMA satellite system. Best results have been obtained by using the self-scheduling strategy. In Portuguese.

- [42] S.L. Martins, C.C. Ribeiro, and M.C. Souza. A Parallel GRASP for the Steiner Problem in Graphs. *Lecture Notes in Computer Science*, 1457:285–297, 1998.

A parallelization of a sequential GRASP for the Steiner minimal tree problem is proposed. The procedure implemented is one of the procedures described in Martins, Pardalos, Resende, and Ribeiro (1999).

The parallelization is accomplished by distributing the GRASP iterations among the processors on a demand-driven basis.

- [43] R.A. Murphey, P.M. Pardalos, and L.S. Pitsoulis. A Parallel GRASP for the Data Association Multidimensional Assignment Problem. *The IMA Volumes in Mathematics and its Applications*, 106:159–180, 1998.

A parallel GRASP for finding good solutions for the data association problem is described. See page 353.

- [44] P.M. Pardalos, M.G.C. Resende, and J. Rappe. An Exact Parallel Algorithm for the Maximum Clique Problem. In: *High Performance Algorithms and Software in Nonlinear Optimization*, R. De Leone, A. Murli, P.M. Pardalos, and G. Toraldo, editors, pages 279–300. Kluwer, 1998.

A parallelized version of the exact algorithm of Carraghan and Pardalos (1990) for the unweighted maximum clique problem is described. A variant of the GRASP for the maximum independent set problem of Feo, Resende, and Smith (1994) is used for computing feasible solutions.

- [45] L.I.D. Rivera. *Evaluation of Parallel Implementations of Heuristics for the Course Scheduling Problem*. Master's Thesis, Instituto Tecnológico y de Estudios Superiores de Monterrey, Monterrey, 1998.

This thesis presents several parallel implementations of heuristics for the course scheduling problem. One of the heuristics is a GRASP. In Spanish.

- [46] S.L. Martins. *Parallelization Strategies for Metaheuristics in Distributed Memory Environments*. Ph.D. Thesis, Department of Computer Sciences, Catholic University of Rio de Janeiro, Rio de Janeiro, 1999.

This thesis considers parallelization strategies for metaheuristics in distributed memory environments. GRASPs for the Steiner tree problem in graphs are described and implemented in parallel. In Portuguese.

- [47] R.M. Aiex, M.G.C. Resende, and C.C. Ribeiro. Probability Distribution of Solution Time in GRASP: An Experimental Investigation. To appear in: *Journal of Heuristics*.

The authors study the probability distributions of solution time to a sub-optimal target value in five GRASPs that have appeared in the literature and for which source code is available. The distributions are estimated by running 12,000 independent runs of the heuristic. Standard methodology for graphical analysis is used to compare the empirical and theoretical distributions and estimate the parameters of the distributions. They conclude that the solution time to a sub-optimal target value fits a two-parameter exponential distribution. Hence, it is possible to approximately achieve linear speed-up by implementing GRASP in parallel.

- [48] R.M. Aiex, M.G.C. Resende, P.M. Pardalos, and G. Toraldo. GRASP with Path Relinking for the Three-Index Assignment Problem. Technical Report, AT&T Labs Research, Florham Park, 2000.

This paper describes variants of GRASP with path relinking for the three index assignment problem (AP3). Computational results show clearly that this GRASP for AP3 benefits from path relinking and that the variants considered in this paper compare well with previously proposed heuristics for this problem. The authors also show that the random variable "time to target solution," for all proposed GRASP with path relinking variants, fits a two-parameter exponential distribution. To illustrate the consequence of this, one of the variants of GRASP with path relinking is shown to benefit from parallelization.

- [49] S.L. Martins, M.G.C. Resende, C.C. Ribeiro, and P.M. Pardalos. A Parallel GRASP for the Steiner Tree Problem in Graphs using a Hybrid Local Search Strategy. *Journal of Global Optimization*, 17:267–283, 2000.

A parallel GRASP for the Steiner problem in graphs is described. See page 351.

15.6 SOURCE CODE

Several papers describe computer source code implementing GRASP. The following papers all fall into this category.

- [50] M.G.C. Resende, P.M. Pardalos, and Y. Li. Algorithm 754: Fortran Subroutines for Approximate Solution of Dense Quadratic Assignment Problems using GRASP. *ACM Transactions on Mathematical Software*, 22:104–118, 1996.

This paper describes a set of ANSI standard Fortran 77 subroutines to find approximate solutions to dense quadratic assignment problems having at least one symmetric flow or distance matrix. It is an optimized implementation of the algorithm described in Li, Pardalos, and Resende (1994).

- [51] P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. Algorithm 769: Fortran Subroutines for Approximate Solution of Sparse Quadratic Assignment Problems using GRASP. *ACM Transactions on Mathematical Software*, 23:196–208, 1997.

A version of the GRASP for the quadratic assignment problem of Li, Pardalos, and Resende (1994), tailored for sparse instances is proposed. A set of ANSI standard Fortran 77 subroutines are described.

- [52] M.G.C. Resende, T.A. Feo, and S.H. Smith. Algorithm 787: Fortran Subroutines for Approximate Solution of Maximum Independent Set Problems using GRASP. *ACM Transactions on Mathematical Software*, 24: 386–394, 1998.

This article describes a set of ANSI standard Fortran 77 subroutines to find an approximate solution of a maximum independent set problem. The GRASP used to produce the solutions is described in Feo, Resende, and Smith (1994).

- [53] P. Festa, P.M. Pardalos, and M.G.C. Resende. FORTRAN Subroutines for Computing Approximate Solution to Feedback Set Problems using GRASP. To appear in: *ACM Transactions on Mathematical Software*.

A set of ANSI standard Fortran 77 subroutines for approximately solving the feedback vertex and arc set problems is described. See page 350.

- [54] C.C. Ribeiro and M.G.C. Resende. Algorithm 797: Fortran Subroutines for Approximate Solution of Graph Planarization Problems using GRASP. *ACM Transactions on Mathematical Software*, 25:341–352, 1999.

This paper describes a set of Fortran subroutines that implements the GRASP for graph planarization of Resende and Ribeiro (1997).

- [55] M.G.C. Resende, L.S. Pitsoulis, and P.M. Pardalos. Fortran Subroutines for Computing Approximate Solutions of MAX-SAT Problems using GRASP. *Discrete Applied Mathematics*, 100:95–113, 2000.

A set of Fortran subroutines for computing approximate solutions of MAX-SAT problems is described. See page 344.

15.7 SCHEDULING

GRASP has been applied to a wide variety of scheduling problems. The following papers illustrate GRASPs for scheduling.

- [56] J.F. Bard and T.A. Feo. Operations Sequencing in Discrete Parts Manufacturing. *Management Science*, 35:249–255, 1989.

This paper presents a method for efficiently sequencing cutting operations associated with the manufacture of discrete parts. The problem is modeled as an integer program. This is relaxed via Lagrangian relaxation into a min-cut problem on a bipartite network. To obtain lower bounds, a max-flow algorithm is applied and the corresponding solution is input to a GRASP.

- [57] T.A. Feo and J.F. Bard. Flight Scheduling and Maintenance Base Planning. *Management Science*, 35:1415–1432, 1989.

See page 358.

- [58] T.A. Feo, K. Venkatraman, and J.F. Bard. A GRASP for a Difficult Single Machine Scheduling Problem. *Computers and Operations Research*, 18:635–643, 1991.

GRASP is applied in this paper to an unusually difficult scheduling problem with flow time and earliness penalties. Two greedy functions are developed and tested. The first is the difference between the flow time and earliness penalties, normalized by the processing time. The second function evaluates the cost of scheduling a job next by estimating the cost of the remaining schedule. The local search uses 2-exchange and insertion exchange.

- [59] M. Laguna and J.L. González-Velarde. A Search Heuristic for Just-in-Time Scheduling in Parallel Machines. *Journal of Intelligent Manufacturing*, 2:253–260, 1991.

This paper presents a hybrid GRASP/tabu search metaheuristic for the weighted earliness penalty problem with deadlines in identical parallel machines.

- [60] P. De, J.B. Ghosj, and C.E. Wells. Solving a Generalized Model for Con Due Date Assignment and Sequencing. *International Journal of Production Economics*, 34:179–185, 1994.

This paper deals with a generalized model for assigning a constant flow allowance (CON) due date to a set of jobs and sequencing them on a single machine. The problem is viewed as a 0-1 quadratic problem and a GRASP is proposed to solve the quadratic problem. The randomization strategy used was inspired by a gradient-based variable forcing methodology proposed by Pardalos and Rodgers (1990) for a branch and bound algorithm. The local search procedure is based on a definition of neighborhood in which two solutions are neighbors if they differ in the value of exactly one variable.

- [61] T.A. Feo, J.F. Bard, and S. Holland. Facility-Wide Planning and Scheduling of Printed Wiring Board Assembly. *Operations Research*, 43:219–230, 1995.

A GRASP is proposed to solve a multiple machine scheduling problem. See page 357.

- [62] J.F. Bard, T.A. Feo, and S. Holland. A GRASP for Scheduling Printed Wiring Board Assembly. *IIE Transactions*, 28:155–165, 1996.

GRASP is used for solving the daily scheduling problem that arises in printed wiring board assembly. See page 357.

- [63] T.A. Feo, K. Sarathy, and J. McGahan. A GRASP for Single Machine Scheduling with Sequence Dependent Setup Costs and Linear Delay Penalties. *Computers and Operations Research*, 23:881–895, 1996.

A GRASP for single machine scheduling with sequence dependent setup costs and linear delay penalties is presented. The greedy function of the GRASP construction phase proposed is made up of two components: the switch over cost and the opportunity cost associated with not inserting a specific job in the next position and instead,

inserting it after half of the unscheduled jobs have been scheduled. This greedy function tends to lead to a balance between the natural order and nearest neighbor approaches. The local search uses 2-exchange, insertion exchange, and a combination of the two.

- [64] J. Xu and S. Chiu. Solving a Real-World Field Technician Scheduling Problem. In: *Proceedings of the International Conference on Management Science and the Economic Development of China*, pages 240–248, 1996.

The objective of the field technician scheduling problem is to assign a set of jobs at different locations with time windows to technicians with different job skills. The greedy choice of the proposed GRASP is to select jobs with the highest unit weight. The local search implements four different moves, among them the 2-exchange and a swap that exchanges an assigned job with another job unassigned under the candidate schedule.

- [65] H. Ramalhinho Lourenço, J.P. Paixão, and R. Portugal. Metaheuristics for the Bus-Driver Scheduling Problem. Technical Report, Department of Economics and Management, Universitat Pompeu Fabra, Barcelona, 1998.

The authors present several metaheuristics for solving real crew scheduling problems that can be used in a transportation planning system, in real-time, and in a user-friendly environment. The metaheuristics proposed are based on the following approaches: GRASP, tabu search, and genetic algorithms. See page 333.

- [66] R.Z. Ríos-Mercado and J.F. Bard. Heuristics for the Flow Line Problem with Setup Costs. *European Journal of Operational Research*, 110:76–98, 1998.

This paper presents two new heuristics for the flowshop scheduling problem with sequence-dependent setup times and makespan minimization objective, one of which is a GRASP. Both heuristics are compared to a previously proposed algorithm, based on the traveling salesman problem (TSP), on randomly generated instances. When setup times are an order of magnitude smaller than processing times, the new approaches prove superior to the TSP-based heuristic. When both processing and setup times are identically distributed, the TSP-based heuristic outperforms the proposed procedures.

- [67] L.I.D. Rivera. *Evaluation of Parallel Implementations of Heuristics for the Course Scheduling Problem*. Master's Thesis, Instituto Tecnológico y de Estudios Superiores de Monterrey, Monterrey, 1998.

See page 336.

- [68] J. Xu and S. Chiu. Effective Heuristic Procedure for a Field Technician Scheduling Problem. To appear in: *Journal of Heuristics*.

The objective of the field technician scheduling problem is to assign a set of jobs at different locations with time windows to technicians with different job skills. Several heuristics are designed and tested for solving the problem: a pure greedy heuristic, a GRASP, and a local search algorithm. The greedy choice of the GRASP proposed is to select jobs with the highest unit weight. The local search implements four different moves, among them the 2-exchange and a swap that exchanges an assigned job with another job unassigned under the candidate schedule.

- [69] R.Z. Ríos-Mercado and J.F. Bard. An Enhanced TSP-Based Heuristic for Makespan Minimization in a Flow Shop with Setup Costs. *Journal of Heuristics*, 5:57–74, 1999.

An enhanced heuristic for minimizing the makespan of the flow shop scheduling problem with sequence-dependent setup times is presented. To tune the parameters, each component of the heuristic is tested over a wide range of problem instances. An experimental comparison with a GRASP reveals the conditions and data attributes where the proposed procedure works best.

- [70] S. Binato, W.J. Hery, D. Loewenstern, and M.G.C. Resende. A Greedy Randomized Adaptive Search Procedure for Job Shop Scheduling. In: *Essays and Surveys on Metaheuristics*, C.C. Ribeiro and P. Hansen, editors, Kluwer, 2001 (this volume).

This paper proposes a GRASP for the job shop scheduling problem. The standard GRASP is enhanced with a memory based intensification scheme, that improves the local search phase around good solutions. The adaptive greedy function is the makespan resulting from the addition of an unscheduled operation to those already scheduled. The greedy choice favors the operation with the minimum value of the greedy function. The local search uses the 2-exchange based on the disjunctive graph model proposed by Roy and Sussmann (1964).

15.8 ROUTING

GRASP has been applied to vehicle, aircraft, telecommunications, and inventory routing problems. The following papers illustrate the application of GRASP to routing problems.

- [71] C.A. Hjorring. *The Vehicle Routing Problem and Local Search Metaheuristics*. Ph.D. Thesis, University of Auckland, Auckland, 1995.

Three metaheuristics for effectively searching through the space of cyclic orders are developed. They are based on GRASP, tabu search, and genetic algorithms. For tabu search, different schemes are investigated to control the tabu list length, including a reactive tabu search method. To obtain good solutions when using the genetic algorithm, specialized crossovers are developed, and a local search

component is added. GRASP is used to construct an initial good solution.

- [72] G. Kontoravdis and J.F. Bard. A GRASP for the Vehicle Routing Problem with Time Windows. *ORSA Journal on Computing*, 7:10–23, 1995.

A GRASP is proposed for minimizing the fleet size of temporarily constrained vehicle routing problems with two types of service. The greedy function of the construction phase takes into account both the overall minimum insertion cost and the penalty cost. Local search is applied to the best solution found every five iterations of the first phase, rather than to each feasible solution.

- [73] M.F. Argüello, J.F. Bard, and G. Yu. A GRASP for Aircraft Routing in Response to Groundings and Delays. *Journal of Combinatorial Optimization*, 1:211–228, 1997.

This paper presents a GRASP to reconstruct aircraft routings in response to groundings and delays experienced over the course of the day. The objective is to minimize the cost of reassigning aircraft to flights taking into account available resources and other system constraints. See page 358.

- [74] J.B. Atkinson. A Greedy Randomised Search Heuristic for Time-Constrained Vehicle Scheduling and the Incorporation of a Learning Strategy. *Journal of the Operational Research Society*, 49:700–708, 1998.

A GRASP is proposed for solving a complex vehicle-scheduling problem with tight time windows and additional constraints. See page 331.

- [75] J.F. Bard, L. Huang, P. Jaillet, and M. Dror. A Decomposition Approach to the Inventory Routing Problem with Satellite Facilities. *Transportation Science*, 32:189–203, 1998.

A methodology is presented that decomposes the inventory routing problem with satellite facilities over the planning horizon, and then solves daily rather than multi-day vehicle routing problems. Three heuristics are proposed for solving the vehicle routing problem with satellite facilities: randomized Clarke-Wright, GRASP, and modified sweep. The GRASP proposed is a modified version of the GRASP of Kontoravdis and Bard (1995).

- [76] L.I.P. Resende and M.G.C. Resende. A GRASP for Frame Relay PVC Routing. In: *Extended Abstracts of the Third Metaheuristics International Conference*, pages 397–402, Angra dos Reis, 1999.

See page 360.

- [77] A. Corberán, R. Martí, and J.M. Sanchís. A GRASP for the Mixed Postman Problem. Technical Report, Department of Statistics and Operations Research, University of Valencia, Valencia, 2000.

The mixed postman problem, a generalization of the Chinese postman problem, is that of finding the shortest tour that traverses each edge of a given mixed graph (a graph containing both undirected and directed edges) at least once. This paper proposes a GRASP for the mixed postman problem.

- [78] C. Carreto and B. Baker. A GRASP Interactive Approach to the Vehicle Routing Problem with Backhauls. In: *Essays and Surveys on Metaheuristics*, C.C. Ribeiro and P. Hansen, editors, Kluwer, 2001 (this volume).

The incorporation of interactive tools into heuristic algorithms is investigated. A GRASP is used in the routes construction and improvement phase. The construction phase is implemented in a clustering heuristic that constructs the routes by clustering the remaining customers according to the vehicles defined by seeds while applying the 3-opt heuristic to reduce the total distance traveled by each vehicle. The greedy function takes into account routes with smallest insertion cost and customers with biggest difference between the smallest and the second smallest insertion costs and smallest number of routes they can traverse. As the local search phase, 3-opt is used.

15.9 LOGIC

GRASP has been applied to problems in logic, including SAT, MAX-SAT, and logical clause inference, as shown by the following papers.

- [79] M.G.C. Resende and T.A. Feo. A GRASP for Satisfiability. *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, 26:499–520, 1996.

This paper describes a GRASP for the satisfiability problem that can be also directly applied to both the weighted and unweighted versions of the maximum satisfiability problem. The adaptive greedy function is a hybrid combination of two functions. One function seeks to maximize the number of yet-unsatisfied clauses that become satisfied after the assignment of each construction iteration, while the other maximizes the number of yet-unassigned literals in yet-unsatisfied clauses that become satisfied if opposite assignments were to be made. The local search flips the assignment of each variable, one at a time, checking if the new truth assignment increases the number of satisfied clauses.

- [80] P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. A Parallel GRASP for MAX-SAT Problems. *Lecture Notes in Computer Science*, 1184:575–585, 1996.

A parallel GRASP is proposed for finding approximate solutions to the weighted maximum satisfiability problem. See page 335.

- [81] M.G.C. Resende, L.S. Pitsoulis, and P.M. Pardalos. Approximate Solution of Weighted MAX-SAT Problems using GRASP. *DIMACS Series*

on Discrete Mathematics and Theoretical Computer Science, 35:393–405, 1997.

This article proposes a GRASP for finding approximate solutions of weighted MAX-SAT problems. The greedy adaptive function is to maximize the total weight of yet-unsatisfied clauses that become satisfied after the assignment of each construction phase iteration. The local search uses the 1-flip neighborhood of a vector x , defined as the set of all binary vectors that differ from x in exactly one literal.

- [82] A.S. Deshpande and E. Triantaphyllou. A Greedy Randomized Adaptive Search Procedure (GRASP) for Inferring Logical Clauses from Examples in Polynomial Time and some Extensions. *Mathematical and Computer Modelling*, 27:75–99, 1998.

Two heuristics are presented in this article for inferring a small size Boolean function from complete and incomplete examples in polynomial time. Each example can be positive or negative depending on whether it must be accepted or rejected, respectively, by the target function. Both of the proposed heuristics are randomized in the sense that instead of choosing the best candidate element, a candidate list is built whose elements are assigned with evaluative function values close to the highest one.

- [83] M.G.C. Resende, L.S. Pitsoulis, and P.M. Pardalos. Fortran Subroutines for Computing Approximate Solutions of MAX-SAT problems using GRASP. *Discrete Mathematics*, 100:95–113, 2000.

A set of Fortran subroutines for computing approximate solutions of MAX-SAT problems is described. The algorithm implemented was proposed by Resende, Pitsoulis, and Pardalos (1997). Two versions of the subroutines are distributed. One version uses a neighborhood data structure in order to speed up the local search phase, while the second version, since it does not make use of this data structure, is more memory efficient but less time efficient. Computational results improve upon those in Resende, Pitsoulis, and Pardalos (1997) using an RCL parameter α randomly chosen each GRASP iteration from the interval $[0, 1]$.

15.10 PARTITIONING

Two types of partitioning problems have been treated with GRASP: number partitioning and VLSI circuit partitioning. The following papers describe these applications.

- [84] M.F. Argüello, T.A. Feo, and O. Goldschmidt. Randomized Methods for the Number Partitioning Problem. *Computers and Operations Research*, 23:103–111, 1996.

This paper presents randomized methodologies for solving the number partitioning problem (the partitioning of a finite set of integers

into two disjoint subsets such that the difference of the sums of the elements in the subsets is minimized). The greedy criterion consists in considering only large elements for differencing. Specific selection of the elements to be differenced is made at random. Differences are placed back into the list of remaining elements, and the process of selecting the next element is repeated. The proposed methods are greedy, randomized, and adaptive construction heuristics, but local search is omitted.

- [85] S. Areibi and A. Vannelli. A GRASP Clustering Technique for Circuit Partitioning. *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, 35:711–724, 1997.

This paper adapts a basic node interchange method for solving the circuit partitioning problem and develop a clustering technique that uses GRASP to generate clusters of moderate sizes. See page 364.

- [86] S.M. Areibi. GRASP: An Effective Constructive Technique for VLSI Circuit Partitioning. In: *Proceedings of IEEE Canadian Conference on Electrical and Computer Engineering*, Edmonton, 1999.

This article proposes a GRASP for obtaining good initial solutions for an iterative improvement technique. At each iteration of the randomized approach, the gains associated with moving modules to the current block being filled are examined, and a restricted candidate list is built using the modules with the highest gains.

15.11 LOCATION AND LAYOUT

Location and layout are another class of problems successfully handled by GRASP. The following papers show how GRASP is used in this context.

- [87] J.G. Klincewicz. Avoiding Local Optima in the p -Hub Location Problem using Tabu Search and GRASP. *Annals of Operations Research*, 40:283–302, 1992.

This paper proposes two heuristics based on tabu search and GRASP for the p -hub location problem. The objective is to overcome the difficulty that local search algorithms encounter. The local search procedure of the GRASP algorithm is based on the 2-exchange.

- [88] H. Delmaire, J.A. Díaz, E. Fernández, and M. Ortega. Comparing New Heuristics for the Pure Integer Capacitated Plant Location Problem. Technical Report DR97/10, Department of Statistics and Operations Research, Universitat Politecnica de Catalunya, Barcelona, 1997.

To solve the pure integer capacitated plant location problem, several heuristics are proposed: evolutionary algorithms, GRASP, simulated annealing, and tabu search. All the algorithms share the same neighborhood definition, which can be one of the following: client plant, client reassignment, client interchange, open plans interchange, and open-closed plants interchange.

- [89] K. Holmqvist, A. Migdalas, and P.M. Pardalos. Greedy Randomized Adaptive Search for a Location Problem with Economies of Scale. In: *Developments in Global Optimization*, I.M. Bomze, T. Csendes, R. Horst, and P.M. Pardalos, editors, pages 301–313. Kluwer, 1997.

This paper proposes a GRASP for finding approximate solutions to a facility location problem with concave costs. The greedy function of the construction phase favors the facilities that give lower cost for a customer, regarding the effect that already connected customers have on the solution. The neighborhood function is defined as changing facility connection for one customer. Instead of a time consuming computation of the objective function value for each neighborhood solution, the difference in cost for changing supplier is examined.

- [90] T.L. Urban. Solution Procedures for the Dynamic Facility Layout Problem. *Annals of Operations Research*, 76:323–342, 1998.

The concept of incomplete dynamic programming is applied to the dynamic facility layout problem and a lower bound for the general problem is developed. A GRASP and an initialized multi-greedy algorithm are described to provide a solution methodology for large problems. The GRASP is the algorithm proposed by Li, Pardalos, and Resende (1994) for dense quadratic assignment problems.

- [91] H. Delmaire, J.A. Díaz, E. Fernández, and M. Ortega. Reactive GRASP and Tabu Search Based Heuristics for the Single Source Capacitated Plant Location Problem. *INFOR*, 37:194–225, 1999.

The single source capacitated plant location problem is a discrete location problem that takes into account capacities in the plants to be opened and imposes that clients be served from a single open plant. The authors propose a hybrid heuristic that embeds reactive GRASP in a tabu search algorithm. See page 333.

- [92] M.J.N. Gomes and J.B.C. da Silva. An Experimental Evaluation of the GRASP Metaheuristic Applied to the Uncapacitated Location Problem. Technical Report 004/99, Department of Statistics and Computation, State University of Ceará, Fortaleza, 1999.

Two GRASPs, one using the ADD heuristic and the other using the DROP heuristic, are proposed for the uncapacitated location problem. Computational experiments with instances from Beasley's OR-Library show that GRASP-DROP dominates GRASP-ADD, while both GRASPs dominate ADD and DROP. In Portuguese.

- [93] S. Abdinnour-Helm and S.W. Hadley. Tabu Search Based Heuristics for Multi-Floor Facility Layout. *International Journal of Production Research*, 38:365–383, 2000.

Two two-stage heuristics are proposed for solving the multi-floor facility layout problem. See page 334.

- [94] T.L. Urban, W.-C. Chiang, and R.A. Russel. The Integrated Machine Allocation and Layout Problem. *International Journal of Production Research*, 76:2911–2930, 2000.

GRASP is used to solve quadratic assignment sub-problems in a model that aggregates quadratic assignment problems with several network flow problems with side constraints. This model is used to produce machine layouts where machines are not required to be placed in a functional or cellular layout.

15.12 GRAPH THEORETIC APPLICATIONS

Perhaps the largest class of problems for which GRASP has been applied is graph theory. The following papers illustrate the wide applicability of GRASP to these problems.

- [95] T.A. Feo, M.G.C. Resende, and S.H. Smith. A Greedy Randomized Adaptive Search Procedure for Maximum Independent Set. *Operations Research*, 42:860–878, 1994.

A GRASP for approximately solving the maximum independent set problem is described. The greedy function chosen orders admissible vertices with respect to the minimum admissible vertex degree. The adjective admissible is referred to a vertex that is not adjacent to any vertex in the current independent set. The neighborhood definition used in the local search is (2, 1)-exchange, where two nonadjacent vertices can be added to the current solution if a single vertex from the solution is removed. See page 334.

- [96] M. Laguna, T.A. Feo, and H.C. Elrod. A Greedy Randomized Adaptive Search Procedure for the Two-Partition Problem. *Operations Research*, 42:677–687, 1994.

A GRASP for the network 2-partition problem is proposed. The greedy function of the construction phase minimizes the augmented weight of the partition. For the local improvement phase, four alternative procedures are considered: best swap, first swap, slight swap, and slightest swap. The best strategies are slight and slightest swaps. Slight swap selects a near-minimum gain exchange at each iteration, while slightest swap chooses the absolute minimum gain.

- [97] E.M. Macambira and C.C. de Souza. A GRASP for the Maximum Clique Problem with Weighted Edges. In: *Proceedings of the XXIX Brazilian Symposium on Operations Research*, page 70, 1997.

The authors propose a branch-and-cut algorithm for the maximum clique problem with weighted edges. The initialization phase of the algorithm uses a GRASP to generate good starting solutions. The greedy function minimizes the sum of weights of the edges outgoing from vertices in the partition. The local search uses the exchange of one element in the current partition with one not in it.

- [98] R. Martí and M. Laguna. Heuristics and Meta-Heuristics for 2-Layer Straight Line Crossing Minimization. Technical Report, Department of Statistics and Operations Research, University of Valencia, Valencia, 1997.

See page 362.

- [99] M.G.C. Resende and C.C. Ribeiro. A GRASP for Graph Planarization. *Networks*, 29:173–189, 1997.

A GRASP for the graph planarization problem is described, extending the two-phase heuristic of Goldschmidt and Takvorian (*Networks*, v. 24, pp. 69–73, 1994). The implementation of the GRASP is described in detail. Computational experience on a large set of standard test problems is presented. On almost all test problems considered, the heuristic either matches or finds a better solution than previously described graph planarization heuristics. In several cases, previously unknown optimal solutions are found.

- [100] R.K. Ahuja, J.B. Orlin, and D. Sharma. New Neighborhood Search Structures for the Capacitated Minimum Spanning Tree Problem. Technical Report, Department of ISE, University of Florida, Gainesville, 1998.

This report presents two generalizations of the best known neighborhood structures for the capacitated minimum spanning tree problem. The new neighborhood structures defined allow cyclic exchanges of nodes among multiple subtrees simultaneously. The first structure proposed allows exchanges of single nodes among several subtrees. The second structure allows moves exchanging subsets of nodes spanning several trees. As the size of both these structures grows exponentially with the problem size, for searching the neighborhood efficiently the authors propose a heuristic search technique based on the concept of *improvement graph* which converts each possible cyclic exchange into a subset-disjoint cycle in the improvement graph and a profitable cyclic exchange into a negative cost cycle, heuristically identified using a modification of the shortest-path label-correcting algorithm. To judge the efficacy of the neighborhoods local improvement and tabu search algorithms have been developed. Local improvement uses a GRASP construction mechanism to generate repeated starting solutions for local improvement.

- [101] M. Laguna and R. Martí. A GRASP for Coloring Sparse Graphs. To appear in: *Computational Optimization and Applications*.

A GRASP for graph coloring is presented. The GRASP construction phase constructs the next coloring, one color at time. The greedy function chooses the vertex having the maximum degree among the uncolored vertices adjacent to at least one colored vertex. At each step, the local search combines the two smallest cardinality color classes into one and tries to find a valid color for each violating vertex.

- [102] E.M. Macambira and C.N. Meneses. A GRASP Algorithm for the Maximum Weighted Edge Subgraph Problem. Technical Report, Department of Statistics and Computation, University of Ceará, Fortaleza, 1998.

A GRASP for the maximum weighted edge subgraph problem is proposed to overcome the difficulties encountered by local search methods. The greedy function of the construction phase favors the vertices corresponding to the maximum sum of the weights associated with its outgoing edges. The local search tries to improve the actual solution by simply swapping one element in the solution set with one not belonging to the solution. In Portuguese.

- [103] S.L. Martins, C.C. Ribeiro, and M.C. Souza. A Parallel GRASP for the Steiner Problem in Graphs. *Lecture Notes in Computer Science*, 1457:285–297, 1998.

See page 336.

- [104] P.M. Pardalos, M.G.C. Resende, and J. Rappe. An Exact Parallel Algorithm for the Maximum Clique Problem. In: *High Performance Algorithms and Software in Nonlinear Optimization*, R. De Leone, A. Murli, P.M. Pardalos, and G. Toraldo, editors, pages 279–300. Kluwer, 1998.

See page 336.

- [105] M.G.C. Resende. Computing Approximate Solutions of the Maximum Covering Problem using GRASP. *Journal of Heuristics*, 4:161–171, 1998.

A GRASP for the maximum covering problem is described. The greedy function is the total weight of the yet-uncovered demand points that become covered after the selection. The local search procedures uses a 2-exchange neighborhood structure. The GRASP is shown to find near optimal solutions. See page 360.

- [106] M.G.C. Resende, T.A. Feo, and S.H. Smith. Algorithm 787: Fortran Subroutines for Approximate Solution of Maximum Independent Set Problems using GRASP. *ACM Transactions on Mathematical Software*, 24: 386–394, 1998.

See page 338.

- [107] E. Rolland, R.A. Patterson, and H. Pirkul. Memory Adaptive Reasoning and Greedy Assignment Techniques for the Capacitated Minimum Spanning Tree Problem. Technical Report, Department of Accounting and Management Information Systems, Fisher College of Business, The Ohio State University, Columbus, 1998.

A GRASP for the capacitated minimum spanning tree problem is proposed. Testing, however, is limited to only a semi-greedy version of the GRASP.

- [108] J. Abello, P.M. Pardalos, and M.G.C. Resende. On Maximum Clique Problems in Very Large Graphs. *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, 50:199–130, 1999.

An approach for clique and quasi-clique computations in very large multi-digraphs is presented. The authors discuss graph decomposition schemes that break up the original problem into several pieces of manageable dimensions. A semi-external memory GRASP is presented to approximately solve the maximum clique problem and maximum quasi-clique problem. The construction phase uses vertex degrees as a guide for construction. The local search uses a simple $(2, 1)$ -exchange. See page 361.

- [109] P. Festa, P.M. Pardalos, and M.G.C. Resende. FORTRAN Subroutines for Computing Approximate Solution to Feedback Set Problems using GRASP. To appear in: *ACM Transactions on Mathematical Software*.

This article describes a set of ANSI standard Fortran 77 subroutines to find approximate solutions of both the feedback vertex set problem and the feedback arc set problem. The GRASP of Pardalos, Qian, and Resende (1999) is used to produce the approximate solutions of the feedback set problem. Feedback arc set problems are converted into feedback vertex set problems and solved.

- [110] S.L. Martins, P.M. Pardalos, M.G.C. Resende, and C.C. Ribeiro. Greedy Randomized Adaptive Search Procedures for the Steiner Problem in Graphs. *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, 43:133–145, 1999.

Four versions of a GRASP for approximately solving general instances of the Steiner problem in graphs are proposed. One is implemented and tested. The construction phase is based on the distance network heuristic. A distance network corresponding to the original graph is built. Associated with each edge of the distance network is a weight that takes into account the shortest distances in the original graph. With respect to the new weight distribution, Kruskal's algorithm is used to solve the minimum spanning tree (MST) problem and the edges in the MST so computed are replaced by the edges in the corresponding shortest paths in the original graph. The local search is based on insertions and eliminations of nodes to and from the current solution.

- [111] P.M. Pardalos, T. Qian, and M.G.C. Resende. A Greedy Randomized Adaptive Search Procedure for the Feedback Vertex Set Problem. *Journal of Combinatorial Optimization*, 2:399–412, 1999.

This article describes a GRASP for finding approximate solutions to the feedback vertex set problem on a digraph. Several greedy functions are tested, all of them taking into account vertices with high degree. The local search procedure tries at each iteration to eliminate redundant vertices. Some efficient problem reduction techniques are also described. They are useful both to simplify the problem instance and to determine whether a digraph is acyclic or not.

- [112] C.C. Ribeiro and M.G.C. Resende. Fortran Subroutines for Approximate Solution of Graph Planarization Problems using GRASP. *ACM Transactions on Mathematical Software*, 25:341–352, 1999.

See page 338.

- [113] S.L. Martins, M.G.C. Resende, C.C. Ribeiro, and P.M. Pardalos. A Parallel GRASP for the Steiner Tree Problem in Graphs using a Hybrid Local Search Strategy. *Journal of Global Optimization*, 17:267–283, 2000.

This paper presents a GRASP for the Steiner problem in graphs. The construction phase is based on the Mehlhorn distance network heuristic, which consists of computing the modified distance network graph and using Kruskal's algorithm to solve the minimum spanning tree problem for the resulting graph. The local search is done by using a combination of key-path based local search and node based local search. See page 337.

- [114] I.H. Osman, B. Al-Ayoubi, M. Barake, and M. Hasan. A Greedy Random Adaptive Search Procedure for the Weighted Maximal Planar Graph Problem. Technical Report, School of Business and Center for Advanced Mathematical Sciences, American University of Beirut, Beirut, 2000.

A GRASP is proposed and tested for the weighted maximal planar graph problem. The construction is a randomized version of the Green and Al-Hakim algorithm (1985). A new data structure is introduced, reducing the complexity of the construction from $O(n^3)$ to $O(n^2)$. Local search uses four types of moves proposed by Pesch, Glover, Bartsch, Salewski, and Osman (1995).

- [115] C.C. Ribeiro, E. Uchoa, and R.F. Werneck. A Hybrid GRASP with Perturbations and Adaptive Path-Relinking for the Steiner Problem in Graphs. Technical Report, Department of Computer Science, Catholic University of Rio de Janeiro, Rio de Janeiro, 2000.

This paper propose and describes a hybrid GRASP with path-relinking for the Steiner problem in graphs. See page 332.

- [116] R. Martí. Arc Crossing Minimization in Graphs with GRASP. To appear in: *IIE Transactions*.

This paper presents a GRASP for minimizing straight-line crossings in hierarchical graphs. See page 363.

- [117] R. Martí and V. Estruch. Incremental Bipartite Drawing Problem. To appear in: *Computers and Operations Research*, 2001.

A GRASP is proposed for the incremental arc crossing minimization problem for bipartite graphs. See page 363.

15.13 QUADRATIC AND OTHER ASSIGNMENT PROBLEMS

GRASP has been applied to several hard assignment problems, including quadratic, biquadratic, multidimensional, and frequency assignment. The following articles cover these problems.

- [118] Y. Li, P.M. Pardalos, and M.G.C. Resende. A Greedy Randomized Adaptive Search Procedure for the Quadratic Assignment Problem. *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, 16:237–261, 1994.

A GRASP for the quadratic assignment problem is described. Construction first makes two assignments, and then completes the solution by making assignments, one at a time. The greedy function is assignment interaction cost. The local search procedure is a 2 assignment exchange.

- [119] T.A. Feo and J.L. González-Velarde. The Intermodal Trailer Assignment Problem: Models, Algorithms, and Heuristics. *Transportation Science*, 29:330–341, 1995.

A GRASP is proposed for solving the problem of assigning highway trailers to railcar hitches in intermodal transportation. See page 358.

- [120] P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. A Parallel GRASP Implementation for the Quadratic Assignment Problem. In: *Parallel Algorithms for Irregularly Structured Problems – Irregular'94*, A. Ferreira and J. Rolim, editors, pages 115–130. Kluwer, 1995.

This paper discusses an efficient parallel implementation of GRASP for sparse quadratic assignment problems. See page 335.

- [121] M.G.C. Resende, P.M. Pardalos, and Y. Li. Algorithm 754: Fortran Subroutines for Approximate Solution of Dense Quadratic Assignment Problems using GRASP. *ACM Transactions on Mathematical Software*, 22:04–118, 1996.

See page 337.

- [122] P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. Algorithm 769: Fortran Subroutines for Approximate Solution of Sparse Quadratic Assignment Problems using GRASP. *ACM Transactions on Mathematical Software*, 23:196–208, 1997.

See page 337.

- [123] R.A. Murphey, P.M. Pardalos, and L.S. Pitsoulis. A Parallel GRASP for the Data Association Multidimensional Assignment Problem. *The IMA Volumes in Mathematics and its Applications*, 106:159–180, 1998.

A GRASP for finding good solutions for the data association multidimensional assignment problem is described. At each discrete time

interval, the data set is formulated as a multidimensional assignment problem (MAP) with a maximum likelihood cost function. A near-optimal solution to each MAP is obtained with a GRASP. The proposed method can be easily parallelized to substantially decrease the running time.

- [124] T. Mavridou, P.M. Pardalos, L.S. Pitsoulis, and M.G.C. Resende. A GRASP for the Biquadratic Assignment Problem. *European Journal of Operational Research*, 105:613–621, 1998.

This paper proposes a GRASP for finding approximate solutions of the biquadratic assignment problem. As in the case of GRASP for the quadratic assignment problem, the construction phase has two stages. The first stage simultaneously makes four assignments, selecting the pairs corresponding to the smallest interaction costs, while the second stage makes the remaining assignments, one at time. The greedy function in the second stage selects the assignment corresponding to the minimum interaction cost with respect to the already-made assignments. In the local search phase, 2-exchange local search is applied to the permutation constructed in the first phase.

- [125] E.L. Pasiliao. A Greedy Randomized Adaptive Search Procedure for the Multi-Criteria Radio Link Frequency Assignment Problem. Technical Report, Department of ISE, University of Florida, Gainesville, 1998.

A GRASP for computing approximate solutions to a radio link frequency assignment problem is proposed. The objective is to minimize the order and the span of the solution set. See page 360.

- [126] M.C. Rangel, N.M.M. de Abreu, P.O. Boaventura Netto, and M.C.S. Boeres. A Modified Local Search for GRASP in the Quadratic Assignment Problem. Technical Report, Production Engineering Program, COPPE, Federal University of Rio de Janeiro, Rio de Janeiro, 1998.

An improvement of the local search phase of the GRASP proposed by Li, Pardalos, and Resende (1994) for solving the quadratic assignment problem is proposed. The new strategy amplifies the local search range and improves the local search's efficiency.

- [127] A.J. Robertson. A Set of Greedy Randomized Adaptive Local Search Procedure (GRASP) Implementations for the Multidimensional Assignment Problem. To appear in: *Computational Optimization and Applications*.

This report introduces four GRASP implementations for the multidimensional assignment problem by combining two constructive methods (randomized greedy and randomized max regret) and two local search methods (2-exchange and variable depth exchange). At each iteration of the randomized greedy construction phase, a set of best assignments is constructed from which a random element is selected and added to the solution set. The greedy function of the randomized max regret construction method is a measure of the competition

between the two leading cost candidates. The maximum regret value corresponds to the candidate assignment that has the largest winning margin between itself and its next highest competitor. The variable depth exchange is an extension of the 2-exchange method that allows a more extensive search of the surrounding neighborhood space.

- [128] C. Fleurent and F. Glover. Improved Constructive Multistart Strategies for the Quadratic Assignment Problem using Adaptive Memory. *INFORMS Journal on Computing*, 11:198–204, 1999.

Adaptive memory strategies that are the heart of tabu search methods are shown to be a foundation for alternative, enhanced, multi-start approaches. This paper illustrates that constructive multistart methods, such as Random Restart and GRASP, can be improved by the addition of memory and associated heuristic search principles. The improved results indicate that these principles (learning, intensification, candidate list strategies, POP) are not limited to applications with transition neighborhoods, as in local search, but can also be useful for applications characterized by constructive (and destructive) neighborhoods. The paper shows that the GRASP for QAP of Li, Pardalos, and Resende (1994) can be improved upon by using these memory strategies. See page 331.

- [129] C.A.S. Oliveira and F.C. Gomes. Two Metaheuristics for Channel Allocation in Mobile Telephony. Technical Report, Artificial Intelligence Laboratory, Universidade Federal do Ceará, Fortaleza, 1999.

The cellular frequency assignment problem described in this paper consists of minimizing the total system interference in mobile phone covered areas, with respect to co-channel and adjacent-channel interference. Two metaheuristics are proposed: GRASP and Asynchronous Team (A-Teams). See page 361.

- [130] L.S. Pitsoulis. *Algorithms for Nonlinear Assignment Problems*. Ph.D. Thesis, Department of Industrial and Systems Engineering, University of Florida, 1999.

This dissertation presents GRASPs for solving the following NP-hard nonlinear assignment problems (NAPs): quadratic assignment problem (QAP), biquadratic assignment problem (BiQAP), turbine balancing problem (TBP), and multidimensional assignment problem (MAP). Computational results indicate that all of the suggested algorithms are among the best in the literature in terms of solution quality and computational time.

- [131] R.K. Ahuja, J.B. Orlin, and A. Tiwari. A Greedy Genetic Algorithm for the Quadratic Assignment Problem. *Computers and Operations Research*, 27:917–934, 2000.

This report suggests a genetic algorithm for the QAP that incorporates the construction phase of the GRASP for QAP of Li, Pardalos, and Resende (1994) to generate the initial population.

- [132] R.M. Aiex, M.G.C. Resende, P.M. Pardalos, and G. Toraldo. GRASP with Path Relinking for the Three-Index Assignment Problem. Technical Report, AT&T Labs Research, Florham Park, 2000.

This paper describes variants of GRASP with path relinking for the three index assignment problem (AP3). See page 337.

- [133] X. Liu, P.M. Pardalos, S. Rajasekaran, and M.G.C. Resende. A GRASP for Frequency Assignment in Mobile Radio Networks. *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, 52:195–201, 2000.

A GRASP for frequency assignment is described in this paper. See page 361.

- [134] M. Prais and C.C. Ribeiro. Reactive GRASP: An Application to a Matrix Decomposition Problem in TDMA Traffic Assignment. *INFORMS Journal on Computing*, 12:164–176, 2000.

This paper describes a GRASP for a matrix decomposition problem arising in the context of traffic assignment in communication satellites. The local search phase of the GRASP proposed is based on a new neighborhood, defined by constructive and destructive moves. See pages 331 and 362.

- [135] M.C. Rangel, N.M.M. Abreu, and P.O. Boaventura Netto. GRASP in the QAP: An Acceptance Bound for Initial Solutions. *Pesquisa Operacional*, 20:45–58, 2000.

This paper presents a modified version of the GRASP algorithm proposed by Li, Pardalos, and Resende (1994) for the quadratic assignment problem. The new GRASP uses a criterion to accept or reject a given initial solution, thus trying to avoid searches that eventually can be fruitless. It computes a normalized limit cost, defined with the aid of QAP upper and lower bounds easily obtained and discards all solutions with cost less than the computed limit. In Portuguese.

- [136] A. Srinivasan, K.G. Ramakrishnan, K. Kumaram, M. Aravamudam, and S. Naqvi. Optimal Design of Signaling Networks for Internet Telephony. In: *Proceedings of the Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies*. Online document, <http://www.ieee-infocom.org/2000/papers/352.ps>, Tel Aviv, 2000.

GRASP is used in an approach for efficient design of a signaling network for a network of software switches supporting Internet telephony. See page 362.

- [137] L.S. Pitsoulis, P.M. Pardalos, and D.W. Hearn. Approximate Solutions to the Turbine Balancing Problem. *European Journal of Operational Research*, 130:147–155, 2001.

In this paper, the turbine balancing problem is formulated as a standard quadratic assignment problem, and a GRASP implementation for solving the resulting problem is presented.

15.14 MANUFACTURING

GRASP has been used to address several optimization problems in manufacturing. The following papers are examples of this.

- [138] J.F. Bard and T.A. Feo. Operations Sequencing in Discrete Parts Manufacturing. *Management Science*, 35:249–255, 1989.

This paper presents a method for efficiently sequencing the cutting operations associated with the manufacture of discrete parts. See page 338.

- [139] T.A. Feo and J.F. Bard. The Cutting Path and Tool Selection Problem in Computer-Aided Process Planning. *Journal of Manufacturing Systems*, 8:17–26, 1989.

The purpose of this paper is to provide a method for minimizing the sum of tool setup and volume removal times associated with metal cutting operations on a flexible machine. The problem is modeled as an integer program, then relaxed into a min-cut problem on a simple network. After obtaining a tentative solution, the authors use a GRASP to identify good feasible points corresponding to alternative process plans. These are seen to speed convergence during branch and bound.

- [140] J.F. Bard and T.A. Feo. An Algorithm for the Manufacturing Equipment Selection Problem. *IIE Transactions*, 23:83–92, 1991.

This paper provides a unified framework in which product and process demands can be related to manufacturing system requirements. The authors develop a nonlinear cost minimization model. The objective is to determine how many of each machine type to purchase and what fraction of the time each piece of equipment will be configured for a particular type of operation. Once the original problem is converted into a MILP, a depth-first branch and bound algorithm is used, employing the greedy randomized set covering heuristic of Feo and Resende (1989), to implicitly search for optimality. Viewing the contribution that any machine makes to satisfy the demand of any process as the unit benefit associated with that machine, a benefit-to-cost ratio is computed for each machine. To derive a feasible solution, the heuristic iteratively selects machines with largest ratio and updates benefits to take into account the remaining demand.

- [141] J.G. Klincewicz and A. Rajan. Using GRASP to Solve the Component Grouping Problem. *Naval Research Logistics*, 41:893–912, 1994.

Two new heuristics are proposed for solving a particular set partitioning problem that arises in robotics assembly, as well as in a number of other manufacturing and material logistics application areas.

The heuristics are GRASPs involving two alternate procedures for determining starting points: component-based and code-based.

- [142] T.A. Feo, J.F. Bard, and S. Holland. Facility-Wide Planning and Scheduling of Printed Wiring Board Assembly. *Operations Research*, 43:219–230, 1995.

This paper describes a decision support system known as INSITES, designed to assist Texas Instruments in the day-to-day assembly operations of their printed wiring board (PWB) facilities. A GRASP is used to solve the underlying multiple machine scheduling problem. The schedule produced at each GRASP iteration is evaluated based on one of five different optimization criteria. The choice of the criterion to be followed is made by the user to rank order the schedules provided by multiple GRASP iterations.

- [143] J.F. Bard, T.A. Feo, and S. Holland. A GRASP for Scheduling Printed Wiring Board Assembly. *IIE Transactions*, 28:155–165, 1996.

The assembly of printed wiring boards (PWBs) typically involves the coordination of thousands of components and hundreds of part numbers in a job shop environment with more than 50 different processes and workstations. The authors propose a GRASP for solving the daily scheduling problem that arises in such environment. The first phase of GRASP obtains a user-specified number of schedules. The greedy function is the product between the weighted processing time and the slack time window.

- [144] J. Yen, M. Carlsson, M. Chang, J.M. Garcia, and H. Nguyen. Constraint Solving for Inkjet Print Mask Design. *Journal of Imaging Science and Technology*, 44:391–397, 2000.

Print masks are used to control the firing of the nozzles, that is, to determine which nozzles on an inkjet printer cartridge are to spit an ink droplet at each particular instant in a multiple-pass print mode. Masks are generated by minimizing the total costs. A GRASP is proposed for automatic generation of print masks. It has been used to design the print masks for Hewlett Packard's wide format printers (DeskJet 2500C and 2500CM). This approach can shorten the turn-around time for print mask design.

15.15 TRANSPORTATION

GRASP has been used to find approximate solutions of problems in air, rail, and intermodal transportation. The following papers illustrate these applications.

- [145] T.A. Feo and J.F. Bard. Flight Scheduling and Maintenance Base Planning. *Management Science*, 35:1415–1432, 1989.

This paper presents a model that can be used by planners to both locate maintenance stations and develop flight schedules that better

meet the cyclical demand for maintenance. The problem is formulated as large-scale mixed integer program, i.e. a minimum cost, multicommodity flow network with integral constraints, where each airplane represents a separate commodity and each arc has an upper and lower capacity of flow. Since obtaining feasible solutions from the relative LP relaxation is difficult, the authors propose a GRASP.

- [146] T.A. Feo and J.L. González-Velarde. The Intermodal Trailer Assignment Problem: Models, Algorithms, and Heuristics. *Transportation Science*, 29:330–341, 1995.

This paper deals with the problem of optimally assigning highway trailers to railcar hitches in intermodal transportation. Using a set covering formulation, the problem is modeled as an integer linear program, whose linear programming relaxation yields a tight lower bound. This formulation also provides the basis for developing a branch-and-bound algorithm and a GRASP for solving the problem. The greedy strategy of the construction phase of GRASP consists in selecting at each step a feasible assignment of the most difficult to use available railcar together with the most difficult to assign trailer. To improve the constructed solution, a 2-exchange local search is applied, carrying out a complete enumeration of the solutions in the neighborhood.

- [147] M.F. Argüello, J.F. Bard, and G. Yu. A GRASP for Aircraft Routing in Response to Groundings and Delays. *Journal of Combinatorial Optimization*, 1:211–228, 1997.

This paper presents a GRASP to reconstruct aircraft routings in response to groundings and delays experienced over the course of the day. The objective is to minimize the cost of reassigning aircraft to flights taking into account available resources and other system constraints. The proposed heuristic is a neighborhood search technique that takes as input an initial feasible solution, so that the construction phase is omitted. Two types of partial route exchange operations are described. The first type is the exchange of flight sequences with identical endpoints. In the second type, the sequence of flights being exchanged must have the same origination airport, but the termination airports are swapped.

- [148] J.F. Bard. An Analysis of a Rail Car Unloading Area for a Consumer Products Manufacturer. *Journal of the Operational Research Society*, 48:873–883, 1997.

This paper discusses how to design and analyze the railcar unloading area of Procter & Gamble's principal laundry detergent plant. The related combinatorial problem of assigning railcars to positions on the platform and unloading equipment to railcars is modeled as a mixed-integer nonlinear program. To approximately solve the problem, four alternatives are proposed and evaluated with the help of a GRASP.

- [149] D. Sosnowska. Optimization of a Simplified Fleet Assignment Problem with Metaheuristics: Simulated Annealing and GRASP. In: *Approximation and Complexity in Numerical Optimization*, P.M. Pardalos, editor, Kluwer, 2000.

Two heuristics based on simulated annealing and GRASP are presented for finding approximate solutions for a simplified fleet assignment problem. Both methods are based on swapping parts of sequence of flight legs assigned to an aircraft (rotation cycle) between two randomly chosen aircrafts. In simulated annealing, the exchange is such that a solution is accepted according to a probability distribution, while in GRASP only exchanges leading to a better solution are permitted and the potentially best part of the assignment is conserved and the rest is randomly reattributed. The construction phase does not use a restricted candidate list explicitly, but a solution is built by simply trying to make the time interval between two flights as small as possible.

15.16 TELECOMMUNICATIONS

Telecommunications, including network design, is a field in which much work with GRASP has been done. The papers below illustrate this.

- [150] J.G. Klincewicz. Avoiding Local Optima in the p -Hub Location Problem using Tabu Search and GRASP. *Annals of Operations Research*, 40:283–302, 1992.

See page 345.

- [151] J. Xu and S. Chiu. Solving a Real-World Field Technician Scheduling Problem. In: *Proceedings of the International Conference on Management Science and the Economic Development of China*, pages 240–248, 1996.

See page 340.

- [152] F. Poppe, M. Pickavet, P. Arijs, and P. Demeester. Design Techniques for SDH Mesh-Restorable Networks. In: *Proceedings of the European Conference on Networks and Optical Communications*, vol. 2, *Core and ATM Networks*, pages 94–101, Antwerp, 1997.

To design low cost reliable telecommunication networks, the authors propose three algorithms: an integer linear programming algorithm (branch-and-cut-and-price), a GRASP, and a zoom-in approach that combines a genetic algorithm with deterministic optimization routines. The greedy choice of the proposed GRASP is to favor paths having lowest additional cost. The local search iteratively tries to reroute some paths in order to further decrease the overall network cost.

- [153] L.I.P. Resende and M.G.C. Resende. A GRASP for Frame Relay PVC Routing. In: *Extended Abstracts of the Third Metaheuristics International Conference*, pages 397–402, Angra dos Reis, 1999.

This paper describes a GRASP for routing permanent virtual circuits (PVC) for frame relay in telecommunications systems. The objective is to minimize PVC delays while balancing trunk loads. The greedy choice selects from the set of not yet routed PVCs the one that minimizes the delay while balancing the trunk loads. The local search procedure reroutes each PVC, one at a time, checking each time if the new route taken together with the remaining fixed routes improves the objective function.

- [154] M.G.C. Resende and O. Ulular. SMART: A Tool for AT&T Worldnet Access Design – Location of Cascade 9000 Concentrators. Technical Report, AT&T Labs Research, Florham Park, 1997.

This report describes SMART, a software tool for finding low cost configurations of Cascade 9000 concentrators in the AT&T Worldnet backbone access network. The concentrator location problem is stated and cost model is presented for concentrator configurations. This cost model is used in a GRASP, proposed for finding approximate solutions to the concentrator location problem. The greedy choice favors the points-of-presence (POPs) with smallest incremental cost. The local search implements a simple 2-exchange.

- [155] E.L. Pasiliao. A Greedy Randomized Adaptive Search Procedure for the Multi-Criteria Radio Link Frequency Assignment Problem. Technical Report, Department of ISE, University of Florida, Gainesville, 1998.

A GRASP is presented for computing approximate solutions to the radio link frequency assignment problem. If a feasible solution exists, the objective is to minimize both the order and the span of the solution set. The local search procedure attempts to eliminate each channel from the communication network. This process is repeated until an attempt to eliminate every frequency in the solution set has been made without success.

- [156] M.G.C. Resende. Computing Approximate Solutions of the Maximum Covering Problem using GRASP. *Journal of Heuristics*, 4:161–171, 1998.

A GRASP for facility location on a network with the objective of maximizing service coverage is proposed. See page 349.

- [157] J. Xu and S. Chiu. Effective Heuristic Procedure for a Field Technician Scheduling Problem. Technical Report, US WEST Advanced Technologies, Boulder, 1998.

See page 341.

- [158] J. Abello, P.M. Pardalos, and M.G.C. Resende. On Maximum Clique Problems in Very Large Graphs. *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, 50:119–130, 1999.

GRASP is used to identify cliques and quasi-cliques in very large multi-digraphs that arise from a telephone call detail database. See page 350.

- [159] C.A.S. Oliveira and F.C. Gomes. Two Metaheuristics for Channel Allocation in Mobile Telephony. Technical Report, Artificial Intelligence Laboratory, Universidade Federal do Ceará, Fortaleza, 1999.

The cellular frequency assignment problem described consists in minimizing the total system interference in mobile phone covered areas, with respect to co-channel and adjacent-channel interference. Two metaheuristics are proposed: GRASP and Asynchronous Team (A-Team). The construction phase of the proposed GRASP is realized by a procedure that at each step chooses the next antenna to which a frequency will be assigned. In the RCL construction, priority is given to transmitters with fewer options of frequency assignment. To implement the local search phase, the authors use a down hill algorithm, that performs random perturbations in the solution, exchanging the frequency of one antenna by another randomly chosen. The stopping criterion used for the down hill algorithm is execution time.

- [160] M. Armony, J.G. Klincewicz, H. Luss, and M.B. Rosenwein. Design of Stacked Self-Healing Rings using a Genetic Algorithm. *Journal of Heuristics*, 6:85–105, 2000.

A hybrid genetic algorithm for design of stacked self-healing rings is proposed and tested. The initial population is made up of randomly generated solutions as well as solutions generated by a GRASP. See page 333.

- [161] X. Liu, P.M. Pardalos, S. Rajasekaran, and M.G.C. Resende. A GRASP for Frequency Assignment in Mobile Radio Networks. *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, 52:195–201, 2000.

A GRASP for frequency assignment is described. Local search uses simulated annealing. The construction phase uses two greedy functions. The first chooses a vertex from the set of unselected vertices with high saturation degrees. The second function is used to assign a frequency to the selected vertex. A frequency is selected from a set of permissible frequencies that contribute little additional cost to the objective function.

- [162] J.G. Klincewicz. Enumeration and Search Procedures for a Hub Location Problem with Economies of Scale. Technical Report, AT&T Labs, Middletown, 2000.

An optimal enumeration scheme, as well as other heuristics based on tabu search and GRASP are proposed for locating hubs in a communications or transportation network.

- [163] M. Prais and C.C. Ribeiro. Reactive GRASP: An Application to a Matrix Decomposition Problem in TDMA Traffic Assignment. *INFORMS Journal on Computing*, 12:164–176, 2000.

This paper describes a GRASP for matrix decomposition problem arising in the context of traffic assignment in communication satellites. A geostationary communication satellite has a number of spot beam antennas covering geographically distributed areas. According to the slot switching configuration on the on-board switch, the uplink traffic received at the satellite has to be immediately sent to ground areas through a set of transponders. The slot switching configurations are determined through the solution of a time slot assignment problem, which is equivalent to the decomposition of a nonnegative traffic matrix into the sum of a family of switching mode matrices. See pages 331 and 355.

- [164] A. Srinivasan, K.G. Ramakrishnan, K. Kumaram, M. Aravamudam, and S. Naqvi. Optimal Design of Signaling Networks for Internet Telephony. In: *Proceedings of the Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies*. Online document, <http://www.ieee-infocom.org/2000/papers/352.ps>, Tel Aviv, 2000.

This paper presents an approach for efficient design of a signaling network for a network of software switches supporting Internet telephony. Optimal load balancing for given demand forecast is formulated as a quadratic assignment problem, which is solved with a GRASP.

15.17 AUTOMATIC DRAWING

GRASP has been used to find approximate solutions to problems that arise in automatic graph drawing. The following papers illustrate this.

- [165] R. Martí and M. Laguna. Heuristics and Meta-Heuristics for 2-Layer Straight Line Crossing Minimization. Technical Report, Department of Statistics and Operations Research, University of Valencia, Valencia, 1997.

Extensive computational results are presented using 12 heuristics and two meta-heuristics for the 2-layer straight line crossing minimization problem. On dense graphs, a tabu search meta-heuristic does best with GRASP a close second. On low-density graphs, GRASP outperforms all other approaches.

- [166] M.G.C. Resende and C.C. Ribeiro. A GRASP for Graph Planarization. *Networks*, 29:173–189, 1997.

GRASP is applied to the graph planarization problem. See page 348.

- [167] E. Fernández and R. Martí. GRASP for Seam Drawing in Mosaicking of Aerial Photographic Maps. *Journal of Heuristics*, 5:181–197, 1999.

Commercial aerial photographic maps are often so large that it is necessary to produce one map from two or even more photographs.

These are combined, two at a time, in a process called *mosaicking*. The objective is to make the final map appear to be the product of a single photograph. The most difficult step in the mosaicking process is *seam-drawing*. This paper proposes a GRASP for solving the seam-drawing process.

- [168] M. Laguna and R. Martí. GRASP and Path Relinking for 2-Layer Straight Line Crossing Minimization. *INFORMS Journal on Computing*, 11:44–52, 1999.

This paper develops a GRASP for the problem of minimizing straight line crossings in a 2-layer graph. The method proposed can be coupled with a path relinking strategy to search for improved outcomes. The greedy criterion of the construction phase is based on the degree of the vertices and a value based restricted candidate list is used. Each step of the improvement phase consists in selecting each vertex to be considered for a move. A probabilistic selection rule is used such that vertices with high degree are more likely to be selected first at each step of this process. See page 331.

- [169] C.C. Ribeiro and M.G.C. Resende. Algorithm 797: Fortran Subroutines for Approximate Solution of Graph Planarization Problems using GRASP. *ACM Transactions on Mathematical Software*, 25:341–352, 1999.

See page 338.

- [170] R. Martí. Arc Crossing Minimization in Graphs with GRASP. To appear in: *IEE Transactions*, 2001.

A GRASP for minimizing straight-line crossings in hierarchical graphs is presented. GRASP is shown to be faster than more complex heuristics but produces lower-quality solutions. It is not as fast as simple heuristics, but finds better-quality solutions. Hence, it is a candidate for practical implementation in graph drawing systems.

- [171] R. Martí and V. Estruch. Incremental Bipartite Drawing Problem. To appear in: *Computers and Operations Research*, 2001.

The goal of limiting the number of arc crossings is a well accepted criterion of how well a graph is drawn. Incremental graph drawing supports interactive updates by users. A GRASP is proposed for the incremental arc crossing minimization problem for bipartite graphs. Computational experiments are done on 450 instances and results are compared with a branch and bound algorithm.

15.18 ELECTRICAL POWER SYSTEMS

GRASP has been applied to a transmission expansion problem in electrical power systems. The two papers below describe this application.

- [172] S. Binato, G.C. Oliveira, and J.L. Araújo. A Greedy Randomized Adaptive Search Procedure for Transmission Expansion Planning. Technical

Report, Research Centre for Electrical Energy – CEPEL, Rio de Janeiro, 1998.

This paper presents a GRASP for a long term transmission expansion planning problem. The greedy function is to minimize the load curtailment required to eliminate all operational violations. The local search phase is based on circuit exchanges, i.e. the procedure exchanges selected additions with unselected additions.

- [173] S. Binato and G.C. Oliveira. A Reactive GRASP for Transmission Network Expansion Planning. In: *Essays and Surveys in Metaheuristics*, C.C. Ribeiro and P. Hansen, editors, Kluwer, 2001 (this volume).

The GRASP previously proposed by Binato, Oliveira, and Araújo (1998) for solving a transmission network expansion problem is enhanced with the reactive scheme of Prais and Ribeiro (2000) to self-adjust the GRASP RCL parameter α . They also propose to apply a bias distribution function of Bresina (1996) to bias the random greedy construction phase towards the most promising variables.

15.19 VLSI DESIGN

GRASP has been applied to circuit partitioning problems in VLSI design, as indicated by the papers below.

- [174] S. Areibi and A. Vannelli. A GRASP Clustering Technique for Circuit Partitioning. *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, 35:711–724, 1997.

This paper adapts a basic node interchange scheme for solving the circuit partitioning problem and develops a clustering technique that uses GRASP to generate clusters of moderate sizes. The number of clusters is predetermined as a function of the number of partitions required. Initially, the heuristic reads the circuit description and resizes the blocks to be used by GRASP, which utilizes only the construction phase to generate the number of required clusters. The GRASP construction phase is followed by a post-processing stage, in which a simple dynamic hill climbing algorithm is used as local search to improve the initial solution generated.

- [175] S.M. Areibi. GRASP: An Effective Constructive Technique for VLSI Circuit Partitioning. In: *Proceedings of IEEE Canadian Conference on Electrical and Computer Engineering*, Edmonton, 1999.

A GRASP for VLSI circuit partitioning is proposed. See page 345.

15.20 MISCELLANEOUS APPLICATIONS

A number of papers on GRASP have appeared in fields as diverse as consumer choice theory and multitarget multisensor tracking. The following papers could

not be classified into one of the previous categories and are grouped as miscellaneous applications.

- [176] T.A. Feo and M.G.C. Resende. A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem. *Operations Research Letters*, 8:67–71, 1989.

GRASP is proposed for set covering. A value based restricted candidate list is used to construct solutions of difficult set covering problems that arise in computing the 1-width of the incidence matrix of Steiner triple systems. The local search is based on the elimination of redundant elements in the cover.

- [177] J.B. Ghosh. Computational Aspects of the Maximum Diversity Problem. *Operations Research Letters*, 19:175–181, 1996.

This paper addresses two variations of the maximum diversity problem. This problem arises when m elements are to be selected from an n -element population based on inter-element distances. Using a reduction from the vertex cover problem, the authors prove that the problem is NP-hard and propose a GRASP for approximately solving it. The neighborhood of a solution is the set of all solutions that can be obtained by replacing an element in the incumbent solution with one element that is not in it.

- [178] R. Colomé and D. Serra. Consumer Choice in Competitive Location Models: Formulations and Heuristics. To appear in: *Papers in Regional Science*.

This paper studies the importance of customer behavior with respect to distance or transportation costs in the optimality of locations obtained by traditional state-of-art competitive location models. The authors propose four models to represent the problem and propose a hybrid metaheuristic for solving it. The proposed method consists of two phases. In the first phase, a good initial solution is found by applying a GRASP procedure, while in the second phase the previous solution found is improved by applying a tabu search heuristic.

- [179] X. Gandibleux, D. Vancoppenolle, and D. Tuyttens. A First Making use of GRASP for Solving MOCO Problems. Technical Report, University of Valenciennes, 1998.

An extension of GRASP, to solve multi-objective combinatorial optimization (MOCO) problems, is considered. In particular, classical covering, assignment, knapsack, and scheduling problems with multiple objectives are used as benchmarks. Computational results compare GRASP solutions for a benchmark set of test problems and results are discussed in comparison with an exact method, when available. In French.

- [180] K. Holmqvist, A. Migdalas, and P.M. Pardalos. A GRASP Algorithm for the Single Source Uncapacitated Minimum Concave-Cost Network

Flow Problem. *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, 40:131–142, 1998.

This paper is concerned with the single source uncapacitated version of the minimum concave-cost network flow problem, that requires establishing a minimum cost flow through a given network from a single source to a set of sinks. The authors propose a GRASP that can be trivially implemented on parallel processors. The construction phase iteratively builds a tree starting from the source node. The elements of the restricted candidate list are end nodes of arcs with a cost close to the best one. The local search phase applies either of the two local search variants proposed by Guisewite and Pardalos (1990).

- [181] N. Krasnogor, D.A. Pelta, W. Russo, and G. Terrazas. A GRASP Approach to the Protein Structure Prediction Problem. Technical Report, LIFIA Lab, University of La Plata, La Plata, 1998.

This paper discusses the applicability of a GRASP for solving a special protein folding problem, an important problem in computational biology. The goal is to predict from the molecular sequence of a given protein its particular 3D structure.

- [182] R.A. Murphrey, P.M. Pardalos, and L.S. Pitsoulis. A Greedy Randomized Adaptive Search Procedure for the Multitarget Multisensor Tracking Problem. *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, 40:277–301, 1998.

A GRASP is presented for approximately solving the multitarget multisensor tracking problem, which can be interpreted as a collection of multidimensional assignment problems. Since the objective is to select a target hypothesis and partition of the measurements that is most likely to occur, a likelihood cost function and partitioning constraint set are developed. The GRASP construction phase creates the restricted candidate list containing the most likely to occur (lower cost) tuples. The local search explores all 2-exchange permutations.

- [183] P.L. Hammer and D.J. Rader, Jr. Maximally Disjoint Solutions of the Set Covering Problem. *Journal of Heuristics*, 7:131–144, 2001.

This paper describes the problem of finding two solutions of a set covering problem that have a minimum number of common variables. It is proved that this problem is NP-complete and three heuristics are proposed for solving it. Two of these algorithms find the solutions sequentially. One of them is a GRASP. The third algorithm finds solutions simultaneously. Each proposed heuristic is a variant of the standard greedy method for set covering problems, whose greedy choice favors unselected variables that maximize the number of uncovered rows that become covered. To reduce the overlap of any pair of solutions, a local search algorithm is used that swaps at

each iteration parts of the solution found with a set of variables not in it.

- [184] M.C. Medeiros, M.G.C. Resende, and A. Veiga. Piecewise Linear Time Series Estimation with GRASP. *Computational Optimization and Applications*, 19:127–144, 2001.

This paper describes a GRASP to build piecewise linear statistical models with multivariate thresholds. The construction phase consists of sequentially choosing hyperplanes until the maximum number of hyperplanes is reached. The greedy function orders the possible hyperplanes with respect to the sum of squared errors of the fitted data. The local search is a 2-exchange heuristic.

- [185] M.C. Medeiros, A. Veiga, and M.G.C. Resende. A Combinatorial Approach to Piecewise Linear Time Series Analysis. To appear in: *Journal of Graphical and Computational Statistics*.

This paper presents a new approach to modeling threshold processes, based on a linear model with time-varying parameters. The authors show that this formulation is closely related to the self-exciting threshold autoregressive models (SETAR) with the advantage that it incorporates linear multivariate thresholds. A GRASP is proposed to estimate the parameters of the model. The greedy choice takes into account the sum of squared errors of the fitted data. The local search is a 2-exchange heuristic.

- [186] J.R. Cano, O. Cordón, F. Herrera, and L. Sánchez. A Greedy Randomized Adaptive Search Procedure for the Clustering Problem. Technical Report, Department of Computer Science and Artificial Intelligence, University of Granada, 2000.

A GRASP for cluster analysis is described. Construction is done using a randomized greedy Kaufman procedure and local search uses the K -means algorithm. High quality solutions are found for benchmark problems. The best solutions are found with a hybrid GRASP/ K -means with Kaufman initialization.

- [187] L.S. Pitsoulis, P.M. Pardalos, and D.W. Hearn. Approximate Solutions to the Turbine Balancing Problem. *European Journal of Operational Research*, 130:147–155, 2001.

See page 356.

16

RECENT ADVANCES IN TABU SEARCH

Michel Gendreau

Centre de recherche sur les transports
Université de Montréal
C.P. 6128, succ. Centre-ville
Montréal, Canada H3C 3J7
michelg@crt.umontreal.ca

Abstract: We review some of the main advances witnessed in the field of Tabu Search (TS) during the last four to five years. We focus on what we see as the most significant trends in the area, namely: techniques to make the search more effective, hybrid methods, and new applications of tabu search.

16.1 INTRODUCTION

In this paper, we review some of the main advances witnessed in the field of Tabu Search (TS) during the last four to five years. This review is by no means extensive nor comprehensive, since it is extremely difficult to keep up with the intense activity in field such as TS; instead we have tried to focus on what we very subjectively see as the most significant trends in the area. The paper presentation is organized around three main themes, with a section devoted to each of these themes: (1) making the search more effective, (2) hybrids, (3) new applications.

16.2 MAKING THE SEARCH MORE EFFECTIVE

In spite of the numerous successes recorded when applying TS to a variety of problems, it is now known that TS often requires a very large number of iterations to find truly excellent solutions. Why is this so? We believe that there are two main answers to this question. The first is that, in many cases, the information collected (or that could be collected) during the search is not properly exploited. The second is that the search is often too local or spends

most of its time in a wrong subset of the solution space. As a result, solutions produced strongly depend on the starting point of the search and good solutions may be missed altogether. It is important to note that these answers are by no means mutually exclusive. In fact, in many situations, the search confines itself to a fairly narrow portion of the solution space because information is not properly exploited.

This overall situation slowly became apparent as TS was applied to more and more problems over the last decade. This led several researchers to develop search approaches that were explicitly aimed at increasing the effectiveness of the search process. While many of these approaches are not extremely recent, it is worth reviewing them because their use can dramatically improve the quality of results obtained by TS and because many newcomers to the field of metaheuristics are still, unfortunately, unaware of their existence. For these reasons, they remain, in our opinion, very “current”.

16.2.1 Exploiting available information better

The first group of these approaches revolves around the systematic use of available information. These approaches are based on the realization, borne out by empirical evidence, that significant “pieces” (or “fragments”) of good solutions are often encountered quite early in the searching process, but that a large number of iterations may be required to put together the “right fragments” in order to form a complete solution when the standard, basic TS algorithm is used. The challenge is thus to find some way of identifying and recombining these “good fragments”. This idea was first proposed by Glover [9] and Glover and Laguna [10] who coined the term *Vocabulary Building* to describe it. A key element in this approach is the assumption, reminescent of the Schema Theorem of *Genetic Algorithms*, that good fragments appear in “good” solutions. One should therefore look for good fragments in the better solutions that have been encountered in the search so far. As for the recombination process, it can often be interpreted and implemented as an intensification or diversification step. The intensification interpretation is immediate since new solutions are derived from good solutions previously encountered. The diversification interpretation stems from the fact that, in most cases, these new solutions are nonetheless significantly different from those from which they were derived. It is important to note that fragments may be defined explicitly (the most widely used option) or implicitly, a fact that is not usually recognized.

To clarify the idea of implicit vocabulary building, we now briefly illustrate it with an example taken from an application to hot roll mill scheduling (Lopez, Carter and Gendreau [15]). In this application, one must build a schedule covering about 24 hours for a hot roll mill in a steel plant. Such a schedule consists in a sequence of several hundred steel “slabs” to be rolled and must satisfy very complex scheduling rules that impact both the objective and constraints that need to be satisfied. The schedule is built from a large pool of slabs available for rolling, which allows the simultaneous construction of several (in fact, 6) disjoint schedules (solutions). The basic solution procedure is a straightforward

TS heuristic in which slabs are moved one at the time between the schedules and the set of unscheduled slabs. This procedure is complemented by a diversification step that is applied to best overall (6-schedule) solution found so far. In this step, blocks of consecutive slabs are swapped between one of the schedules (the “active schedule”) and all other schedules with the objective of improving the active schedule. The length and location within schedules of the swapped blocks are not defined *a priori*, but rather determined through local optimization by the diversification operator. To interpret this procedure in the context of Vocabulary Building, it must be noted that the blocks swapped into the active schedule by the diversification operator do in fact correspond to good subsequences of slabs appearing in the other “good” schedules of the best solution. In this implementation, one may also notice that the identification and recombination operations of Vocabulary Building are performed simultaneously.

Another major contribution in the quest for a better exploitation of available information is the concept of *Adaptive Memory* that was proposed by Rochat and Taillard [17] as a way to simultaneously achieve diversification and intensification of the search. This concept can be interpreted as an explicit implementation of the Vocabulary Building framework, since it mainly applies to problems for which solutions can be broken down in well-defined elementary components, such as routes in Vehicle Routing Problems. The elementary components of the best solutions encountered throughout the search are stored in a fixed-length list called the “adaptive memory”, sorted in decreasing order of quality (w.r.t to the objective function value of the solutions they were part of). The regular search process is interrupted from times to times to reconstruct a new solution by combining elementary components from various solutions in the memory using a biased sampling scheme and rules to ensure consistency. The search process is then restarted from this new solution. The algorithm usually terminates after a fixed number of calls to the memory. This technique has been applied in most cases with TS as the local search process, but this is not essential; for instance, local descent can be also used yielding a method known as “adaptive descent” (see, e.g., Gendreau et al. [5]). Adaptive memory approaches have proven to be extremely effective in various application contexts.

Another way to induce intensification/diversification by exploiting information from previously found “good” solutions is *Path relinking*. This method was proposed several years ago by Fred Glover, but seldom used until recently. The basic idea here is to examine the solutions found along paths (in the search space) linking two “good” solutions with the hope of finding even better solutions. Detailed explanations about possible applications and implementations of the technique can be found in Glover and Laguna [11].

16.2.2 Creating better starting points

Contrary to what was often believed in the early development stages of TS, experience has shown that TS heuristics can be quite sensitive to the quality of

the initial solutions from which the search is launched. This can be traced back to the fact that even a long search can remain quite close (in the search space) to its starting point. It is thus a good idea to look for better starting points for the search. This can obviously be achieved by developing sophisticated routines for creating initial solutions, but these are clearly highly dependent upon the specific problem being tackled. A more generic approach for creating better initial solutions consists in using a “multi-phase” approach. Under such an approach, one first creates several initial solutions, which can be built using fairly simple procedures, and performs a “short” TS from each of them; the best of the solutions produced by these “short” TS’s is then selected as the starting point for a “regular” application of TS. This technique was shown to be extremely effective in vehicle routing applications (see, e.g., Gendreau, Hertz and Laporte [6]) and on the Steiner tree problem (Gendreau, Larochelle and Sansò [7]) among others.

16.2.3 Using more powerful neighbourhood operators

Another common belief from the early years of metaheuristics that has recently come under close scrutiny is that one could rely on the inherent power of these methods to find good solutions, disregarding the inner heuristics around which the search strategy is organized. In fact, it is now generally accepted that if very simple moves are used, even TS will have difficulties escaping from valleys around local optima. Many researchers have thus started using much more powerful neighborhood operators, i.e., operators that allow for very significant modifications of the current solution at each iteration, such as Ejection Chains (Glover [9]), Cyclic Transfer, and Sequential Fan/Look Ahead strategies (see Glover and Laguna [11]). Neighborhoods can also be defined by relatively complex procedures. In most, if not all, cases, these more powerful operators define very large neighborhoods that can prove extremely costly to explore in an exhaustive fashion. This must be carefully considered to avoid inordinate computing times. Techniques such as sampling and other partial evaluation schemes can be extremely useful to cut down the neighborhood evaluation times in this context.

16.2.4 Parallel search strategies

The overall effectiveness of local search processes such as TS can also be significantly improved by increasing their globality through “parallel” exploration strategies. Under such strategies, one can achieve a much broader exploration of the search space by running “concurrently” several independent search threads. This parallel exploration can be implemented on true parallel computing architectures, but also in a quasi-parallel fashion on sequential machines. The main challenge faced by designers of parallel TS approaches is to make sure that the threads can be effectively coordinated, i.e., that they can exchange useful information between them. Numerous schemes have thus been proposed to organize multi-threaded search (see Crainic, Toulouse and Gendreau [11]).

One of the most successful ones consists in coupling parallel search with adaptive memory. Under this scheme, threads exchange information by sending their good solutions to a central, common adaptive memory. Other interesting schemes involve partitioning the solution space in a fixed or adaptive fashion. Experimental results indicate that parallel search can be quite efficient; it may even produce better solutions than similar sequential search schemes for a given amount of computing resources (Crainic, Toulouse and Gendreau [3]).

16.3 HYBRIDS

The basic idea behind the use of hybrid algorithms is that by combining principles from two or more approaches, one may be able to develop heuristics displaying the desirable properties of all the original methods. A typical example of this is given by TS-GA (Genetic Algorithms) hybrids where TS replaces the mutation operator of GA with the objective of achieving simultaneously aggressiveness (the TS component) and breadth (the GA component) of the search. Hybrids combining Scatter Search with TS have also been developed with similar objectives (see, for instance, Trafalis and Al-Harkan [18], for an interesting hybrid scatter genetic tabu search approach). Other successful hybrids involve coupling TS with integer programming techniques (there is a complete chapter on this topic in the book by Glover and Laguna[11]), Lagrangean methods (Gruenert[12]) or column generation (Crainic, Gendreau and Farvolden [2]).

An intriguing type of hybrid is the one combining TS with *Constraint Programming* (CP) techniques, in which the CP machinery is used to perform the evaluation of neighborhoods. This approach is particularly suitable when dealing with real-life problems with lots of additional “dirty” constraints. Using CP in this context to evaluate neighborhoods allows for the development of “clean” search procedures that can be applied to many variants of the same problem. The CP constraint propagation algorithms can also eliminate implicitly infeasible solutions during neighborhood evaluation; this turns out to be a critical feature when there are many complicating constraints to be dealt with (Pesant and Gendreau [16]).

While the most natural way of hybridizing approaches is to combine algorithmic principles of the original methods in a single, complex algorithmic design, there is an alternative way to exploit the hybridization concept: parallel multi-agent search architectures in which individual agents run “pure” methods and communicate among themselves. For instance, one can run concurrently a TS and a GA, or a TS and a branch-and-bound. One may even run several different TS heuristics alone or in combination with other methods. Experimentation with this approach using on the one hand a group of parallel TS threads and on the other a GA has produced intriguing results: adding the GA improves the overall quality of results, in spite of the fact that the GA never itself finds the best solution (Crainic and Gendreau[1]). More recent computational experiments with similar hybridization schemes have confirmed the effectiveness of parallel multi-agent search architectures (Le Bouthillier, Crainic, and Keller [13]; Le Bouthillier [14]).

16.4 NEW APPLICATIONS

Early applications of TS dealt almost exclusively with fairly standard combinatorial problems. In recent years, its realm of application has significantly expanded to include a variety of problems that differ substantially in nature from these early applications. This, in turn, has led to interesting methodological advances that were necessary to tackle the new applications. We now briefly describe some of these, indicating in each case the new challenges that algorithm designers must face to address them.

Complex non-convex continuous optimization problems are encountered in a large number of real-life applications. In many cases, existing solution techniques are unsatisfactory as they either only allow fairly small instances to be solved or provide low-quality heuristic answers. In these settings, TS becomes an interesting alternative. When considering this type of problems, we must distinguish between concave (minimization) problems and problems with no particular structure. Concave problems yield extreme-point solutions and are thus essentially of a combinatorial nature; as a result, they can be tackled by a fairly straightforward application of TS principles (using, for instance, neighborhoods defined by pivoting rules for problems with polyhedral feasible sets). The application of TS to unstructured problems with continuous variables is much more challenging. Two main issues to be addressed are the development of efficient neighborhood structures and the handling tabus. With respect to the latter issue in particular, one must find some way of recording the recent history of the search in an efficient manner. While application-specific answers to these issues may often be required, it is, however, possible to derive a general approach for these problems. It entails using a coordinate search strategy in which the value of only variable is modified at each iteration. Neighborhoods can then be defined with respect to the variables and various step sizes and tabus can be recorded as (variable, direction of change) pairs. Experiments with this search strategy have shown that it can be effective.

Multi-criteria optimization is another category of problems frequently encountered in real-life settings. The main difficulty encountered when dealing with such problems is the lack of a clear, well-defined objective to guide the search. This can be alleviated by constructing an artificial objective by weighing the various criteria considered. The weights used to construct this artificial objective can then be varied in a systematic or adaptive way reminiscent of strategic oscillation to explore a wide range of alternative solutions. An interesting consequence of the use of TS (or for that matter of any local search method) in this context is that one may build, as the search proceeds, a list of non-dominated (with respect to the original criteria) solutions, thus yielding when the search terminates a list of solutions defining a “quasi-efficient” frontier.

With the recent advances in modeling, algorithmic design and computing power, the inherent uncertainty present in many decision-making situations is now explicitly accounted for in many O.R. models. Stochastic optimization problems are notoriously difficult to solve, especially if they involve discrete

variables, unless they can be tackled using dynamic programming related approaches. TS has been applied in the past to stochastic programming problems, but these applications were limited to cases where the number of scenarios (possible realizations of the random variables) were quite small, thus allowing for an exact, albeit costly, evaluation of potential solutions. When the number becomes large or infinite (e.g., when some random variables follow Normal or Poisson distributions), such an approach is no longer possible, since the evaluation of a single candidate solution becomes prohibitive. It is possible to borrow ideas such as scenario sampling from the field of exact methods for continuous stochastic programming to develop TS approaches to such problems. The adaptation of these ideas in a TS framework is, however, far from trivial, since it makes the comparison of solutions and the definition of a stopping criterion difficult. Preliminary work in this area seems to indicate though that these difficulties can be overcome (Gendreau and Louveaux [8]).

Another emerging area for the application of TS is that of real-time problems where some data become available or are modified dynamically during the resolution process. Improvement methods such as TS are particularly attractive to tackle such problems, since they almost exhibit the “Anytime” property that is often required from real-time algorithms. (The “Anytime” property applies to algorithms that can return a feasible solution with any allotment of computing time, and that provide better and better solutions when they are given more time). The basic approach in this context consists in applying TS on the data currently available and in performing fast updates of problem data and current best solution(s) when data is modified. The use of parallel TS methods is particularly relevant in this context, especially if the real-time constraint is stringent. A number of applications in areas such as fleet management (Gendreau et al. [5]) and contingency plan optimization have clearly demonstrated the effectiveness of TS in real-time settings.

16.5 CONCLUSION

This short, and by no means extensive, overview of recent advances in Tabu Search methods should provide a definite indication of the vitality of TS as a research area. Many of research topics described above are still at a very early stage of development and there are still many opportunities for extremely interesting research projects in this field. As a final remark, it should be pointed out that many of ideas presented by Glover and Laguna [11] in their seminal 1997 book are still untapped.

References

- [1] T.G. Crainic and M. Gendreau. Towards an Evolutionary Method – Cooperative Multi-Thread Parallel Tabu Search Heuristic Hybrid. In: *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, S. Voss, S. Martello, I.H. Osman, and C. Roucairol, editors, Kluwer, pages 331–344, 1999.

- [2] T.G. Crainic, M. Gendreau, and J.M. Farvolden. Simplex-Based Tabu Search for the Multicommodity Capacitated Fixed-Charge Problem. *INFORMS Journal on Computing*, 12:223–236, 2000.
- [3] T.G. Crainic, M. Toulouse, and M. Gendreau. Synchronous Tabu Search Parallelization Strategies for Multicommodity Location-Allocation with Balancing Requirements. *OR Spektrum*, 17:113–123, 1995.
- [4] T.G. Crainic, M. Toulouse, and M. Gendreau. Towards a Taxonomy of Parallel Tabu Search Heuristics. *INFORMS Journal on Computing*, 9:61–72, 1997.
- [5] M. Gendreau, F. Guertin, J.-Y. Potvin, and É. Taillard. Parallel Tabu Search for Real-Time Vehicle Routing and Dispatching. *Transportation Science*, 33:381–390, 1999.
- [6] M. Gendreau, A. Hertz, and G. Laporte. A Tabu Search Heuristic for the Vehicle Routing Problem. *Management Science*, 40:1276–1290, 1994.
- [7] M. Gendreau, J.-F. Larochelle, and B. Sansò. A Tabu Search Heuristic for the Steiner Tree Problem. *Networks*, 34:163–172, 1999.
- [8] M. Gendreau and F.V. Louveaux. A Tabu Search Approach for Stochastic Integer Programming with Recourse. *Extended Abstracts of the 2nd Metaheuristics International Conference*, page 287, Sophia-Antipolis, 1997.
- [9] F. Glover. Ejection Chains, Reference Structures and Alternating Path Methods for the Traveling Salesman Problem. University of Colorado. Shortened version published in: *Discrete Applied Mathematics*, 65:223–253, 1996.
- [10] F. Glover and M. Laguna. Tabu Search. In: *Modern Heuristic Techniques for Combinatorial Problems*, C. Reeves, editor, pages 70–141, Blackwell, 1993.
- [11] F. Glover and M. Laguna. *Tabu Search*. Kluwer, 1997.
- [12] T. Gruenert. Lagrangean Tabu Search. In: *Essays and Surveys in Metaheuristics*, C.C. Ribeiro and P. Hansen, editors, Kluwer, 2001 (this volume).
- [13] A. LeBouthillier, T.G. Crainic, and R. Keller. Parallel Cooperative Evolutionary Algorithm for Vehicle Routing Problems with Time Windows. Paper presented at: *Odysseus 2000*, pages 78–81, Chania, 2000.
- [14] A. LeBouthillier. Modélisation UML pour une architecture coopérative appliquée au problème de tournées de véhicules avec fenêtres de temps. Publication CRT-2000-39, Centre de recherche sur les transports, Montréal, 2000.

- [15] L. Lopez, M.W. Carter, and M. Gendreau. The Hot Strip Mill Production Scheduling Problem: A Tabu Search Approach. *European Journal of Operational Research*, 106:317–335, 1998.
- [16] G. Pesant and M. Gendreau. A Constraint Programming Framework for Local Search Methods. *Journal of Heuristics*, 5:255–280, 1999.
- [17] Y. Rochat and É. Taillard. Probabilistic Diversification and Intensification in Local Search for Vehicle Routing. *Journal of Heuristics*, 1:147–167, 1995.
- [18] T. Trafalis and I. Al-Harkan. A Continuous Scatter Search Approach for Global Optimization. Extended abstract in: *Conference in Applied Mathematical Programming and Modeling (APMOD'95)*, London, 1995.

17 LAGRANGEAN TABU SEARCH

Tore Grünert

Operations Research, RWTH Aachen
Templergraben 64, D-52056 Aachen, Germany
Tore@or.rwth-aachen.de

Abstract: Tabu Search (TS) and Lagrangean relaxation (LR) are important techniques for solving large-scale combinatorial optimization problems. They have been applied successfully to a wide range of different problems from such diverse areas as routing and scheduling, network design, cutting and packing, production planning etc. In this paper we will outline how these techniques can be combined in order to obtain an even better performance. The basic idea behind such a combination is to identify promising neighbors of a current solution in the TS framework by LR. This gives a dual price and memory controlled search procedure. The principles and results of this algorithmic framework are illustrated by applying them to the solution of the capacitated warehouse location problem.

17.1 INTRODUCTION

Both Lagrangean relaxation and Tabu Search have been employed extensively as methods for solving difficult combinatorial optimization problems. In this paper we introduce a combination of these two techniques, the *Lagrangean Tabu Search* (LTS). It is a generally applicable metaheuristic which combines ideas from both fields. The LTS algorithm relies on the solution process of the Lagrangean dual. During that process new Lagrangean multipliers and corresponding solutions to the LR are computed by an iterative algorithm. These solutions are used to compute neighbor solutions in the framework of TS.

During the formal description of LTS we will assume that the following mixed-binary program has to be solved

$$z = \min cx + hy \quad (17.1)$$

s.t.

$$A^1x + B^1y \geq b^1 \quad (17.2)$$

$$A^2x + B^2y \geq b^2 \quad (17.3)$$

$$x \in I\!\!R^{n_1} \quad (17.4)$$

$$y \in \{0, 1\}^{n_2} \quad (17.5)$$

where $c \in I\!\!R^{n_1}$, $h \in I\!\!R_{+}^{n_2}$, $A^1 \in I\!\!R^{m_1 \times n_1}$, $A^2 \in I\!\!R^{m_2 \times n_1}$, $B^1 \in I\!\!R^{m_1 \times n_2}$, $B^2 \in I\!\!R^{m_2 \times n_2}$. Without loss of generality, it is assumed that all binary variables have non-negative objective function coefficients, which implies that a cost is charged whenever a variable is set to one. It is convenient to abbreviate the set of feasible solutions corresponding to (17.2)-(17.5) by X . The objective function value of a feasible solution $(x, y) \in X$ is denoted by $z(x, y)$. In this paper it will be assumed that it is necessary to fix a number of binary variables to one in order to obtain a primal feasible solution. This means that the typical greedy algorithm for such a problem will start with all binary variables equal to zero and fix variables to one in a sequential manner. Fixing one binary variable to one may, of course, imply that a set of other binary variables must be equal to zero. Typical combinatorial problems having this structure are, among many others, the traveling salesman problem, problems in graphs such as matching and optimal trees, the set packing and partitioning problem and the warehouse location problem. In the following the capacitated warehouse location problem will be used as an example for the application of LTS to a specific combinatorial optimization problem. We have chosen this problem since it is difficult to solve and has been widely studied.

The paper proceeds as follows. In Section 17.2 the LR technique and the use of Lagrangean heuristics is briefly reviewed. Then a short summary of the basic TS algorithm is given in Section 17.3. Based on these reviews, the Lagrangean Tabu Search is described in Section 17.4. An important special case of LR is the so-called *Lagrangean decomposition* (LD), where several sub-models of a problem are identified and solved independently in the LR. LD gives rise to a special version of LTS which is discussed in Section 17.5. In order to demonstrate the potential of LTS computational results for the capacitated warehouse location problems are given in Section 17.6. Finally, some conclusions and potential further developments are described in Section 17.7.

17.2 LAGRANGEAN RELAXATION

LR is one of the most popular techniques for solving mixed-integer programming problems. LR was initially applied to the Traveling Salesman Problem by Held and Karp in 1971 [16]. Geoffrion analyzed its theoretical properties and studied the problem in the context of integer programming [10]. LR can be used to compute bounds on the optimal value of the objective function and to guide heuristics in order to find good solutions. It can be argued that

Lagrangian-based heuristics were one of the first metaheuristics that found wide acceptance. We will show below that they make greedy algorithms more intelligent by basing the decisions about which variable to fix on the reduced costs of the variables. We now proceed by introducing LR and Lagrangean duality and describing the Lagrangean-based greedy (meta-) heuristic.

The LR approach can be described as follows: In the mixed-binary programming problem (17.1)-(17.5), it is assumed that there exists a set of nicely structured constraints (17.2), (17.4), and (17.5) and a set of constraints (17.3) which destroy the structure of the problem. The idea of LR is to eliminate the difficult constraints (17.3) and to penalize their violation using non-negative Lagrangean multipliers $\pi \in \mathbb{R}_+^{m_2}$. This gives the LR with respect to the constraints (17.3):

$$z_{LR}(\pi) = \min cx + hy + \pi(b^2 - A^2x - B^2y) \quad (17.6)$$

subject to (17.2), (17.4), and (17.5). Note that the objective function (17.6) can be re-arranged to

$$z_{LR}(\pi) = \min(c - \pi A^2)x + (h - \pi B^2)y + \pi b^2 \quad (17.7)$$

which demonstrates that the Lagrangean multipliers π adjust the value of the objective function entries of the variables. The adjusted values $\bar{c} = c - \pi A^2$ and $\bar{h} = h - \pi B^2$ are called the *reduced costs* of the variables x and y .

It is easy to prove that $z_{LR}(\pi) \leq z$ and thus (17.6) provides a lower bound on the optimal solution (17.1). The best lower bound is obtained by maximizing (17.6) over the possible values of π and is known as the Lagrangean dual:

$$z_{LD} = \max_{\pi \in \mathbb{R}_+^{m_2}} z_{LR}(\pi). \quad (17.8)$$

This problem is usually solved by subgradient optimization [22], dual adjustment techniques [9], bundle methods [17] or optimally with respect to the bound by column generation [10].

Example: Capacitated warehouse location

The Capacitated Warehouse (or Facility) Location Problem (CWLP) consists of selecting a subset of warehouses from a large set of potential warehouses J in order to meet the demand d_k of a set of customers $k \in K$. The aggregate demand that is serviced from a warehouse is not allowed to exceed its capacity $m_j, j \in J$. A fixed cost f_j is charged if the warehouse $j \in J$ is opened and a transportation cost c_{jk} for each unit of demand that is transported from the warehouse j to the customer k . We denote the sum of the demands for a set of customers $S \subseteq K$ by $d(S) = \sum_{k \in S} d_k$, the capacity provided by a set of warehouses $T \subseteq J$ by $m(T) = \sum_{j \in T} m_j$, and the cost of a set of warehouses $T \subseteq J$ by $f(T) = \sum_{j \in T} f_j$. The objective is to minimize the overall costs. The warehouse location problem is also referred to as the facility or site location problem and belongs to the set of fixed-charge network design problems. It is known that the CWLP is NP-hard and only moderate-size instances can be solved to optimality.

Using the notation introduced above the CWLP can be formulated as a mixed-binary programming model. The decision variable y_j is equal to one if the warehouse j is open and zero otherwise. The fraction of the demand of customer k , which is supplied from warehouse j is denoted by x_{jk} . A corresponding model is:

$$z = \min \sum_{j \in J} f_j y_j + \sum_{j \in J} \sum_{k \in K} c_{jk} x_{jk} \quad (17.9)$$

subject to

$$\sum_{j \in J} x_{jk} = 1 \text{ for all } k \in K \quad (17.10)$$

$$\sum_{k \in K} d_k x_{jk} \leq m_j y_j \text{ for all } j \in J \quad (17.11)$$

$$0 \leq x_{jk} \leq y_j \text{ for all } j \in J, k \in K \quad (17.12)$$

$$y_j \in \{0, 1\} \text{ for all } j \in J. \quad (17.13)$$

The objective function (17.9) minimizes the sum of fixed costs for opening a warehouse and transportation costs. The constraints (17.10) guarantee that each customer receives its entire demand. If some customer is served by a warehouse, then it is forced to be open as modeled by (17.11). The (redundant) constraints (17.12) state that the flow from a warehouse to a customer is non-negative and cannot exceed the customer's demand. They are included in order to strengthen the LP-relaxation of WCLP. Eventually, the fact that a warehouse can be either closed or open is modeled by (17.13).

There are several possible strategies for choosing the difficult constraints in the CWLP. These are discussed both from a theoretical and computational point of view in [8]. Here we follow [23] and relax the demand constraints (17.10) with multipliers $\pi_k, k \in K$. This results in the LR

$$\begin{aligned} z_{LR}(\pi) &= \min \sum_{j \in J} f_j y_j + \sum_{j \in J} \sum_{k \in K} c_{jk} x_{jk} + \sum_{k \in K} \pi_k \left(1 - \sum_{j \in J} x_{jk} \right) \\ &= \min \sum_{j \in J} f_j y_j + \sum_{j \in J} \sum_{k \in K} (c_{jk} - \pi_k) x_{jk} + \sum_{k \in K} \pi_k \end{aligned}$$

subject to (17.11), (17.12) and (17.13). The LR can be solved for any multiplier values π by solving $|J|$ continuous knapsack problems [23]. For each warehouse $j \in J$, we denote the optimal value of the continuous knapsack problem plus the fixed costs of opening the warehouse by $a_j(\pi)$. Hence, if there is no constraint on the number of open warehouses, one always has $y_j(\pi) = 1$, if $a_j(\pi) \leq 0$ and $y_j(\pi) = 0$, if $a_j(\pi) > 0$ as the optimal solution to LR. The incorporation of lower and upper bounds on the number of open warehouses is straightforward and is described in [23].

In addition to providing bounds on the optimal value of the objective function in branch and bound frameworks, LR has been applied successfully as a

guideline for heuristics. A number of examples are discussed in [6]. The general idea of these heuristics is to exploit the information which is generated during the optimization of the Lagrangean dual. After an initial convergence of this process, the vector of Lagrangean multipliers π determines a vector of decision variables $(x(\pi), y(\pi))$ as a solution of the corresponding LR. However, these variable assignments are usually infeasible for the primal problem. By infeasible we mean that the LP

$$z(y(\pi)) = \min cx + hy(\pi) \quad (17.14)$$

s.t.

$$A^1x + B^1y(\pi) \geq b^1 \quad (17.15)$$

$$A^2x + B^2y(\pi) \geq b^2 \quad (17.16)$$

$$x \in \mathbb{R}^{n_1} \quad (17.17)$$

does not have a feasible solution. On the other hand the LP may be feasible but one expects that it can be profitable to either fix binary variables with $y(\pi) = 1$ to zero or binary variables with $y(\pi) = 0$ to one. In both these cases a *repair algorithm* is invoked. The repair algorithm attempts to construct a feasible solution by fixing binary variables to one in order of increasing reduced costs or set some binary variables which have already been included to zero. For example, in the set covering problem, e.g. [7], one fixes further variables corresponding to columns to one until all rows have been covered and afterwards tries to eliminate unnecessary columns. In the CWLP on the other hand one usually fixes all binary variables with negative reduced costs to one even if the capacity of a subset of the corresponding warehouses provide enough capacity to serve all customers since one expects to save transportation costs. In the following we will denote the vectors of binary variables which result from the repair algorithm by \tilde{y} . When the problem at hand includes continuous variables it can sometimes be difficult to determine whether the LP (17.14)-(17.17) is feasible or not. In the worst case one may be forced to solve the LP by standard linear programming which may be very time-consuming, especially if the problem is infeasible. In this case one may relax the constraints (17.2) and (17.3) by adding artificial variables $s^1 \in \mathbb{R}^{m_1}, s^2 \in \mathbb{R}^{m_2}$, with high positive costs M , resulting in the primal problem

$$z_R(\tilde{y}) = \min cx + h\tilde{y} + Ms^1 + Ms^2 \quad (17.18)$$

$$\text{s.t.} \quad (17.19)$$

$$A^1x + B^1\tilde{y} + s^1 \geq b^1 \quad (17.20)$$

$$A^2x + B^2\tilde{y} + s^2 \geq b^2 \quad (17.21)$$

$$x \in \mathbb{R}^{n_1} \quad (17.22)$$

$$s^1 \in \mathbb{R}^{m_1}, s^2 \in \mathbb{R}^{m_2} \quad (17.23)$$

which always has a feasible solution and indicates how ‘far’ the current solution is from a feasible one. Most repair algorithms in the context of LR have been implemented as greedy algorithms which are guided by reduced costs. In many

of these algorithms one corrects the reduced cost during the run of the greedy algorithm in order to take into account that variables have been fixed to one after the solution of the LR. A good example for this technique is the set covering problem, e.g. [7]. Here the reduced costs of the column variables are adjusted according to the information about which rows have already been covered. The process of converting a dual Lagrangean solution into a primal feasible solution by the repair algorithm is iterated by computing new Lagrangean multipliers and activating the repair algorithm until some stopping criterion is satisfied. Usually the stopping criterion is based on the number of iterations, the convergence rate or the gap between the Lagrangean lower bound and the best primal solution. We now outline the general scheme of this type of the Lagrangean metaheuristic for covering problems, where feasibility can usually be obtained by fixing further binary variables to one:

- [1] (Initialization phase) Compute Lagrangean multipliers by some iterative algorithm (for example, subgradient optimization). Stop when preliminary convergence has been obtained. Let the cost of the incumbent be $c_{best} = \infty$.
- [2] (Repeated solution by the Greedy Algorithm) REPEAT
 - (a) (Initialization of the Greedy Algorithm) Input the relevant values of the solution of the LR, i.e. π and $y(\pi)$. Set the current primal binary solution equal to the current binary solution of the LR, i.e. $\tilde{y} := y(\pi)$. Set the set of included binary variables to those variables with $y(\pi) = 1$, i.e. $I := \{i \in \{1, \dots, n_2\} : y(\pi) = 1\}$.
 - (b) (Iteration) WHILE the binary solution \tilde{y} does not satisfy some problem-specific condition (e.g. feasibility, profitability) DO:
 - i. (Selection of the next binary variable) Select a binary variable $\tilde{y}_k, k \notin I$ with the lowest reduced cost value, and fix it to one, i.e. $\tilde{y}_k = 1$ and $I := I \cup \{k\}$.
 - ii. (Update of reduced costs) Adjust the reduced costs of all other variables $\tilde{y}_i, i \notin I$ in order to take the last variable assignment into account.
 - iii. (Primal partitioning) Solve the Linear Program (LP) (17.14)-(17.17) or a relaxation, e.g. (17.18)-(17.23). Let the value of the variables in the optimal solution to the LP be (x', y') .
 - (c) (Update of the incumbent) If $(x', y') \in X$ and $c(x', y') < c_{best}$ then let $(x_{best}, y_{best}) := (x', y')$.
 - (d) (Computation of Multipliers) Update the Lagrangean multipliers π by one or several iterations of a suitable multiplier update algorithm.
- UNTIL the stopping criterion is satisfied.

It can be seen that the repair algorithm in this case is implemented by the greedy loop 2b. Note that the greedy algorithm would be different if one deals

with packing problems, where it is usually necessary to fix further variables to zero. The most difficult problems are of the partitioning type where it may be necessary to perform swaps (i.e. setting a set of variables to zero and a different set of variables to one at the same time). This is, however, a general concern with constructive heuristics for such problems and the adaptation of the Lagrangean metaheuristic depends on the specific problem.

Example continued: Capacitated warehouse location

The typical Lagrangean-based greedy algorithm for the CWLP uses the values of the binary decision variables from the LR directly in order to construct a primal feasible solution. Denote the subset of open warehouses from the solution of the LR by $J^* := \{j \in J : y_j(\pi) = 1\}$. Then a primal solution can be found by solving a transportation problem with J^* as the set of suppliers and K as the set of customers. The transportation problem has a feasible solution whenever $m(J^*) \geq d(K)$. Therefore, if $m(J^*) < d(K)$, one can sort the warehouses in $J \setminus J^*$ in order of increasing values of $a_j(\pi)$ and add new warehouses to J^* in this order, i.e. set $\tilde{y}_j = 1$ until the condition $m(J^*) \geq d(K)$ is satisfied. It is easy to recognize that this algorithm is an adaptation of the greedy approach given above. If the value of the optimal solution to the transportation problem is denoted by $c(J^*)$, then the value of the corresponding CWLP is $f(J^*) + c(J^*)$.

For large instances it may be computationally expensive to solve the transportation problem exactly. Beasley [3], therefore, suggests the following greedy procedure: Consider the customers in decreasing demand order and supply each customer from its cheapest warehouse until either the warehouse has no capacity left or the entire demand has been satisfied. Of course, any other heuristic for the transportation problem can be used. An interesting approach is to solve the transportation problem exactly only if the value of the heuristic solution is close to or better than the value of the incumbent solution.

17.3 TABU SEARCH

TS combines ideas from network optimization and Artificial Intelligence and was first presented by Glover in 1986 [11]. It is a metaheuristic which controls underlying local search procedures in order to escape from local optima. The control is based on different types of memory, which is implemented by tabu lists and aspiration criteria. A basic TS algorithm is given below:

- [1] (Initialization) Input a solution (x, y) . Set tabu list empty.
- [2] (Iteration) WHILE stopping criterion not met DO:
 - (a) (Scan neighborhood) Evaluate all neighbor solutions $N(x, y)$.
 - (b) (Select move) Select the best neighbor solution $(x', y') \in N(x, y)$ which is not tabu or is tabu and satisfies an aspiration criterion.
 - (c) (Perform move) Let $(x, y) := (x', y')$ and update tabu-list correspondingly.

The above algorithm only gives a definition of the basic TS algorithm. An overview over the different and more advanced aspects of TS, such as intensification and diversification can be found in [13].

The basic decision in any TS implementation is the choice of a neighborhood. For any solution $(x, y) \in X$, its neighborhood $N(x, y) \subseteq X$ is defined by the set of solutions which are ‘close’ to (x, y) in some problem-specific way. In TS the neighborhood is scanned before a decision is made on where to move in the current iteration. The evaluation of the neighborhood is usually the most time-consuming part of TS. It can be improved by using e.g. randomization, auxiliary evaluation functions, and candidate lists. Here we suggest another approach, namely the definition of a neighborhood based on the LR techniques defined above.

Example continued: Capacitated warehouse location

A natural neighborhood structure for the CWLP consists of so-called ADD and DROP moves [21] or their combination, the so-called interchange moves. An ADD move adds an additional warehouse to the solution, whereas the DROP moves remove a warehouse. An exact evaluation of an ADD or DROP move requires the solution of a transportation problem. In order to save computation time, approximate measures have been suggested [20, 19].

A natural tabu list within the interchange framework is to forbid the introduction of recently dropped warehouses and to forbid the elimination of recently added warehouses. Diversification can be introduced by removing warehouses which have been present in most of the solutions and intensification by fixing a number of warehouses and considering only the resulting smaller problem.

Computation time can also be saved by employing the following candidate list strategy: For each customer, its candidate list contains the l (e.g. $l = 10$) ‘nearest’ warehouses (in terms of assignment costs c_{jk}). Now, if a warehouse is dropped, the new warehouse must be on the candidate list of at least one of the customers assigned to this warehouse.

17.4 LAGRANGEAN TABU SEARCH

The Lagrangean Tabu Search (LTS) combines the features of the LR and TS. The algorithm iteratively computes new Lagrangean multipliers – and, hence, new solutions to the LR – using the techniques mentioned above, e.g. subgradient optimization. Let $call > 0$ be a positive constant. Then LTS is activated every $call$ iterations by performing two consecutive phases. In the first phase, the Lagrangean solution is compared to the best feasible solution found so far (incumbent). The comparison is based on the values of the binary variables. If the values of the binary variables in the LR and the incumbent are equal, they are kept. On the contrary, if the values are different, the corresponding variables are added to the (initially empty) set of open variables in case they are admissible in terms of a tabu condition as illustrated next. The values of these variables are optimized in the second phase by a heuristic or an exact optimization algorithm which will be called ‘repair algorithm’. The basic

difference to LR is that the repair algorithm in LTS only works on a specified subset of the binary variables, namely the open variables. We now describe both phases formally.

The basis of the approach is the incumbent solution (x, y) and the solution $(x(\pi), y(\pi))$ of the LR. Define the symmetric difference $SD \in \{1, \dots, n_2\}$ of the two solutions to be the index set of all binary variables which have different values in the two solutions subject to the condition that these variables are not tabu, i.e. $SD := \{i \in \{1, \dots, n_2\} : y_i \neq y(\pi)_i \text{ and } y_i \text{ not tabu}\}$. Some useful tabu conditions will be specified later. Let the cardinality of SD be $s = |SD|$ and denote its complement by $\overline{SD} := \{1, \dots, n_2\} \setminus SD$. The algorithm first fixes all binary variables in \overline{SD} to their current values. Then the repair algorithm is used to complete this partial solution. The repair algorithm thus has to solve the following problem:

$$z(SD) = \min c\tilde{x} + h\tilde{y} \quad (17.24)$$

s.t.

$$A^1\tilde{x} + B^1\tilde{y} \geq b^1 \quad (17.25)$$

$$A^2\tilde{x} + B^2\tilde{y} \geq b^2 \quad (17.26)$$

$$\tilde{x} \in I\!\!R^{n_1} \quad (17.27)$$

$$\tilde{y} \in \{0, 1\}^{n_2} \quad (17.28)$$

$$\tilde{y}_i = y_i \text{ for all } i \in \overline{SD}. \quad (17.29)$$

Note that the problem (17.24)–(17.29) always has a feasible solution, namely the current incumbent. We suggest the following simple rule that controls the tabu state and thus the repair problem that has to be solved: Variables which were free variables during one of the last t iterations are tabu for a number of iterations (the tabu tenure). Moreover, this tabu list is cleared whenever a new best solution has been determined. The purpose of this type of tabu list is different from standard TS implementations: It avoids the selection of the same variables from SD and reduces the number of variables which have to be considered when solving the repair problem. Especially if the size s of SD is too big with respect to an acceptable computation time, the tabu list controls the selection of different sets of free variables in a natural way. In addition to this tabu state we also use a *tabu set* which stores ‘setup patterns’ of the binary variables (i.e. a coding of the values of the binary variables) in a hash table. Before a call to the repair algorithm is made, a look-up in the hash table determines whether the corresponding setup pattern has already been investigated.

In the following we will restrict our attention to two possible repair algorithms. The first algorithm is similar to the reduced-costs-based greedy algorithm described in the case of LR above. Here the basic difference to the pure Lagrangean approach is the definition of the repair problem. In LTS the tabu state can be used to define the repair problem (17.24)–(17.29). This avoids the repeated solution of the same repair problem, which is often observed in practice, e.g. [23]. A different type of repair algorithm can be implemented by

a tree-search procedure such as branch and bound. In this case, the variables in \overline{SD} are fixed to their values and the remaining problem is optimized over the free variables in SD . In case s is too big, this may be computationally prohibitive. One may, therefore, restrict the number of free variables by choosing only $r < s$ free variables according to their reduced costs and their tabu status as above. The remaining $s - r$ variables must then be fixed to the value they attain in the incumbent. This assures that the optimization problem over the free variables has a feasible solution.

The basic flow of the LTS algorithm can now be summarized. It does not contain any ‘advanced’ features, such as diversification and intensification. These can be implemented for improved performance.

[1] (Initialization) Initialize the Lagrangean multipliers π and input a solution (x, y) . Set tabu list empty.

[2] (Iteration) WHILE stopping criterion not met DO:

- (a) (Obtain a new solution of the LR) Obtain a new solution of the LR by performing *call* iterations of the multiplier update algorithm (for example, subgradient optimization). Input the solution, i.e. the reduced costs π and the corresponding decision variables $(x(\pi), y(\pi))$.
- (b) (Determine SD) Compute the symmetric difference SD .
- (c) (Call repair algorithm) Call the repair algorithm with arguments $y, y(\pi), \pi$, and SD . Let the new solution be (x', y') .
- (d) (Perform move and adjust tabu list) IF $c(x', y') < c(x, y)$, then let $(x, y) = (x', y')$ and clear tabu list. ELSE set variables in SD tabu.

Due to the acceptance rule 2d the LTS is a standard hill-climbing procedure that does not allow the objective function value of the current solution to deteriorate. Therefore, since the number of free variables in SD must be limited to r , local optima where more than r binary variables attain different values from the current solution cannot be reached (assuming no intermediate local optima). However, this acceptance rule consistently outperformed other acceptance rules in computational tests.

When the repair algorithm is implemented as a branch and bound procedure several design parameters regarding the implementation of the algorithm have to be fixed:

- The variable selection rule determines the selection of the next binary variable to branch on. This rule can be implemented as follows: Sort the binary variables according to increasing reduced cost value as in the greedy algorithm and select them for branching according to this sequence.
- Branching is done in the standard way by fixing the selected binary variable to zero and one.

- The bounding rule can be based on the current value of the Lagrangean relaxation. It can be amended by other logical rules. Since the number of free variables in the optimization is small, bounding rules with low-complexity computation requirements should be used. The motivation for this is that it is faster to branch when the possible tree-size is strongly limited than to perform advanced bounding at each node.
- New upper bounds are calculated whenever all the variables in the tree have been fixed.
- Variable fixing based on the Lagrangean bound can be incorporated into the branch and bound algorithm or be performed outside the procedure.

Example continued: Capacitated warehouse location

A LTS for the CWLP can be based on the LR of the CWLP described above. Here we will use subgradient optimization to update the Lagrangean multipliers π . Based on these implementation decisions the LTS is called every *call* subgradient iterations with the binary solution of the current LR $y(\pi)$ as an input. First the symmetric difference SD is computed. Only variables which are not tabu are added to SD . If the size of SD is too big, elements are removed using the selection rule based on reduced costs as described above.

Whenever SD is not empty, the branch and bound phase is activated. Here we can use the solution of the current LR to compute bounds during the branch and bound phase. This follows from the fact that the fixing of a variable can easily be incorporated into the solution algorithm for the LR. A transportation problem is solved whenever all binary variables have been fixed, i.e. when a terminal node in the branch and bound tree has been reached. Furthermore, nodes can be fathomed whenever the corresponding partial solutions do not provide enough open capacity or contain too few or too many open warehouses. The minimal and maximal number of open warehouses is updated during the algorithm using the method described in [3]. We have also included the variable reduction techniques described there.

17.5 LTS AND LAGRANGEAN DECOMPOSITION

Often a mixed-binary program decomposes naturally into several structured subproblems. In this case better bounds can be obtained by making copies of the decision variables and relaxing the equality constraints. This technique is known as Lagrangean decomposition, variable splitting, or layering strategies [22], [12], [15] and lends itself naturally to LTS. Let $y^p(\pi)$ be the vector of decision variables in the solution of the p -th subproblem, $p \in \{1, \dots, P\}$ of the LR. We define a voting coefficient for variable $i \in \{1, \dots, n_2\}$ as the sum of variable values in the subproblems, i.e. $v_i := \frac{1}{P} \sum_{p=1}^P y^p(\pi)$. A value of zero (one) indicates that the variable should be fixed to zero (one). Values close to 0.5 point to contradictions. In order to obtain one value, a voting threshold $0 \leq \bar{v} \leq 0.5$ is defined. Using the voting threshold, a partial solution from the LR can be constructed by setting $y_i(\pi) = 1$ if $v_i \geq 1 - \bar{v}$ and $y_i(\pi) = 0$ if

$v_i \leq \bar{v}$. No decision about the value of $y_i(\pi)$ can be made if $\bar{v} < v_i < 1 - \bar{v}$. The symmetric difference SD now contains the indices of the binary variables with $y_i \neq y_i(\pi)$ as well as $\bar{v} < v_i < 1 - \bar{v}$. Tabu conditions can be implemented in the usual way. The selection of integer variables based on reduced costs in the repair algorithm cannot be applied in this case since there is no single LR master problem but rather a set of independent problems, each with its reduced costs. In this case the tree-search type of repair algorithm is better suited since it allows conflicting variable assignments in the subproblems to be resolved by optimization. We, therefore, select the free variables in this approach in the order of decreasing magnitude of the Lagrangean multipliers, i.e. $|\pi_i|$ with the idea that large multipliers indicate conflicting values in different subproblems.

A Lagrangean decomposition approach for the CWLP has been developed by Barcelo et al. in [2]. They decompose the CWLP into one semi-assignment problem and a knapsack problem similar to the one we solve here. It is shown in [8] that the optimal value of the Lagrangean dual does not improve if Lagrangean decompostion is used instead of LR in this case. However, from a computational point of view it might give better results. A problem where more than two independent subproblems can be recognized and solved stems from lotsizing in production planning [14]. Four independent subproblems can be identified and solved: a mixed-binary knapsack problem, an elementary shortest path problem, a dynamic lotsizing problem, and a Leontief production planning problem. A LTS algorithm based on the decomposition described above has been developed. The binary variables in this model correspond to setup variables on a set of machines. The repair algorithm, thus, has to decide on the production quantities and the setup sequence over a number of periods on each machine. This was done by first fixing an initial sequence corresponding to the variables which do not belong to the symmetric difference and then inserting variables based on the magnitude of the Lagrangean multipliers as discussed above. Large-scale problems with up to 300 products, 10 machines, and 10 periods could be solved within a reasonable amount of time. These results go far beyond the capabilities of other approaches and show that LTS is a good alternative when complex models have to be solved.

17.6 COMPUTATIONAL RESULTS FOR THE CAPACITATED WAREHOUSE LOCATION PROBLEM

In order to demonstrate the effectiveness of LTS a computational study has been performed using the set of benchmark instances for the CWLP from the OR-library [4]. The size of the instances range from 16 warehouses and 50 customers up to 100 warehouses and 1000 customers. All these instances have been solved to optimality by the parallel branch and bound algorithm given in [3]. The most effective heuristic is the Lagrangean heuristic described in [5]. It does use a different LR than we use here and also employs local search to improve the incumbent solutions.

We used subgradient optimization to optimize the Lagrangean dual. The exponentially smoothed update rule described in [1] was used to determine

subgradients. The weighting coefficient was set to $\alpha = 0.5$. This rule outperformed the ones described in [5, 23]. The repair algorithm was called every $call = 30$ iterations. All other ingredients are identical to [5]. The transportation problems were solved exactly using the CPLEX network optimization callable library [18]. In the branch and bound phase we limited the number of free variables to seven.

The computational study focused on the impact of the tabu tenure and the number of iterations on the quality of the solutions. More specifically, we wanted to

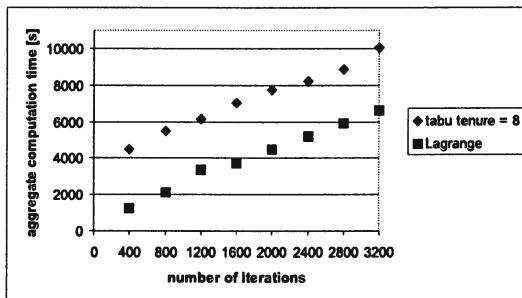
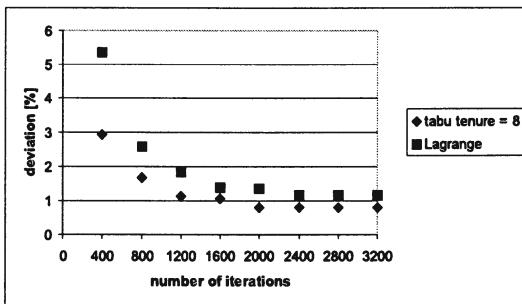
- compare LTS to the standard Lagrangean heuristic (using the settings above);
- study the impact of the number of iterations on the quality of the solution and the computation time;
- study the impact of the tabu tenure on quality and computation time; and
- compare the results to the best available heuristic of Beasley [5].

All computational tests were performed on a Pentium 300 MHz PC with 128 MB RAM using the Microsoft NT 4.0 operating system. The algorithms were coded in C++ using the Microsoft Visual C++ compiler, release 6.0, with target set to ‘release’.

Our first observation was that the ‘small’ instances IV-1-XII-4 cannot be used to discriminate the performance of the algorithms since they are too easy to solve. Even the pure Lagrangean heuristic with 1,200 iterations solved all but three of the instances to optimality. The deviation from optimality measured as $100(z_{UB} - z_{Opt})/z_{Opt}$ in percent was 0.44 for XI-4 and 0.06 for IX-4 and XII-4, respectively. Here z_{UB} is the best upper bound and z_{Opt} the value of the optimal solution. Computation times are very short, the longest time being about 6 seconds for the instance XII-4. These deviation results are slightly better than the ones given in [5]. Computation times cannot be compared due to different hardware. Thus, in the following we restrict ourselves to the instances A-1-C-4 with 100 warehouses and 1000 customers. It should, however, be noted that this set of 12 instances is too small to draw any statistically significant conclusions. Still, averaging over all instances will only bias the results and does not provide any new information.

In order to show the relationship between quality and computation times, the standard Lagrangean algorithm and the LTS with a tabu tenure set to 8 were compared for different numbers of iterations. The average deviation from the lower bound in percent defined as $100(z_{UB} - z_{LB})/z_{LB}$ (z_{LB} is the Lagrangean lower bound) over the 12 instances (Excess) and the total computation times (Agg. Time) are given for both algorithms in Table 17.1. The aggregate computation time and the average excess are depicted in Figures 17.1 and 17.2. As can be expected, LTS outperforms the Lagrangean heuristic with respect to the quality of the solutions on an iteration count basis. For the test instances

Number of Iterations	Lagrangean heuristic		LTS	
	Excess [%]	Agg. Time [s]	Excess [%]	Agg. Time [s]
400	5.36	1,228	2.94	4,497
800	2.57	2,117	1.66	5,499
1200	1.82	3,380	1.13	6,166
1600	1.38	3,725	1.06	7,062
2000	1.34	4,477	0.81	7,765
2400	1.15	5,222	0.81	8,229
2800	1.15	5,915	0.81	8,876
3200	1.15	6,627	0.79	10,032

Table 17.1 Performance of Lagrangean heuristic and LTS.**Figure 17.1** Aggregate time as a function of the number of iterations.**Figure 17.2** Average excess as a function of the number of iterations.

the average excess of the Lagrangean heuristic was 1.53 times the excess of LTS and one iteration of LTS took 2.04 times as long as one iteration of the Lagrangean heuristic. Both algorithms show a sub-linear growth of computation time with the number of iterations. For the Lagrangean heuristic this can easily

be explained by the fact that the effort per iteration is basically constant and is reduced when some warehouses are fixed open or closed when the bounds converge. In the case of LTS the behavior is more difficult to explain due to the repeated call of the branch and bound algorithm and its close link to the tabu state of the variables. If we assume that the size of the repair problem remains constant on average, then the sub-linear behavior of the computation time can be attributed both to the increasing number of fixed variables and the use of the tabu set described above.

The improvement of LTS over the Lagrangean heuristic on an iteration count basis can be expected. Since one iteration of the LTS takes longer than one iteration of the Lagrangean heuristic, the question is whether LTS pays off on a time-equalized basis. We, therefore, plotted the deviation from the lower bound as a function of computer time in Figure 17.3. It can be observed that

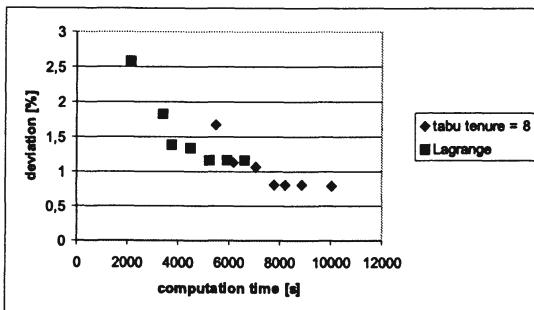


Figure 17.3 Average excess as a function of the aggregate computation time.

LTS is only efficient if very high-quality solutions are required. For lower-quality solutions it is better to run the Lagrangean heuristic for about twice as many iterations as the LTS. After 2000 iterations the subgradient optimization has converged and the Lagrangean heuristic does not provide new and better solutions. The same does basically hold for LTS. However, LTS computes significantly better results and is the method of choice if improved solutions are sought.

We now focus on the effect of the tabu tenure on the computation time and the quality of the solutions. As noted above the tabu list is used in a very different way from standard implementations of TS. It is, thus, difficult to predict the results stemming from runs with different tabu tenures. Our basis for a comparison is the total computational time and the average excess for the test instances with a fixed number of iterations. Figure 17.4 depicts the results when the number of iterations was fixed to 1600. Several observations can be made: First, there is no clear relationship between the tabu tenure and the two criteria. The plots for other iteration counts showed very different results. Secondly, computation time was largest for a tabu tenure equal to zero. This

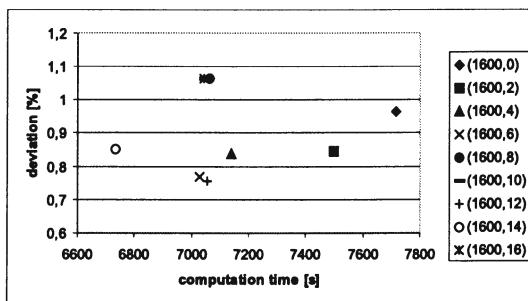


Figure 17.4 Average excess and aggregate computation time as a function of tabu tenure.

results from the larger size of the repair problems and was also observed for other iteration counts. The third observation cannot be inferred from the figure: A further increase of the tabu tenure (above 16 in this case) did not change the results. This follows from the fact that the tabu list was erased whenever a new best solution was found. The results can thus be summarized as follows: Tabu tenures between zero and some (instance-dependent) upper bound do influence the quality of the solutions but any good value cannot be fixed a priori. However, on average, tabu tenures greater than zero lead to a reduction in computation times and to an increased quality. Eventually we compare the

Instance	LTS		Beasley's heuristic	
	Excess [%]	Time [s]	Excess [%]	Time [s]
A-1	0	314	0.241	73.65
A-2	0	371	0.454	65.49
A-3	0.64	473	0	29.87
A-4	0	217	0	16.58
B-1	0	360	0.4	131.84
B-2	2.64	774	0.143	111.78
B-3	2.06	1235	0.907	96.14
B-4	0	920	0	48.92
C-1	0	486	0	105.74
C-2	0.39	762	0	83.42
C-3	0.65	1099	0.443	86.89
C-4	0.71	779	0	59.11

Table 17.2 Performance of LTS and Beasley's heuristic.

quality of the solutions of the instances A-1–C-4 with the results published in [5]. We used a tabu tenure of 16 and a fixed number of 2,000 iterations. All other settings remained as above. We also give the computational time of our algorithm which cannot be compared in useful way to the ones given in [5] since very different computer architectures are involved. Table 17.2 shows the

instances, the computational times in seconds and the deviation from the lower bound in percent for both algorithms. The table shows that Beasley's heuristic gives slightly better results for these instances. However, our algorithm does not use as many variable fixing techniques and does not employ any local search. Interestingly, the 'difficult' instances are different for each algorithm. This might be attributed to the different type of relaxation.

17.7 CONCLUSIONS

This paper has introduced the Lagrangean Tabu Search as a new metaheuristic for solving difficult combinatorial optimization problems. The algorithm combines ideas from Lagrangean relaxation and Tabu Search and has the advantage of providing lower and upper bounds on the quality of the solutions. The basis of LTS is the solution of the Lagrangean dual by some iterative technique, such as subgradient optimization. Using the information generated during that process, new neighbor solutions according to the Tabu Search framework are generated. This neighborhood forms the basis for the solution of a so-called repair problem which can be optimized by a heuristic or an optimization algorithm such as branch and bound. Implementation strategies both for a greedy-type algorithm and a branch and bound procedure have been outlined.

The implementation of LTS for solving the Capacitated Warehouse Location Problem has been discussed in detail. The computational results show that LTS is competitive with the best available algorithms for this problem – although we did not put any effort into tuning of parameters or the implementation of advanced TS techniques such as intensification and diversification. The implementation of such advanced techniques within the framework of LTS is straightforward since frequency information can easily be obtained during the solution process. This information can be used in combination with the reduced cost information to fix variables temporarily or permanently. The computational results also showed that it is difficult to determine a 'good' tabu tenure *a priori*. This indicates that it may be useful to use randomly controlled, variable-length tabu lists. The study of these concepts is a fruitful area of future research.

Acknowledgments: I would like to thank Mr. Birger Funke and Mr. Stefan Irnich who read earlier drafts of this paper and the two anonymous referees. Their suggestions helped to correct mistakes and improve the presentation.

References

- [1] B.M. Baker and J. Sheasby. Accelerating the Convergence of Subgradient Optimisation. *European Journal of Operational Research*, 117:136–144, 1999.
- [2] J. Barcelo, E. Fernandez, and K.O. Jørnsten. Computational Results From a New Lagrangean Relaxation Algorithm for the Capacitated Plant Location Problem. *European Journal of Operational Research*, 53:38–45, 1991.

- [3] J.E. Beasley. An Algorithm for Solving Large Capacitated Warehouse Location Problems. *European Journal of Operational Research*, 33:314–325, 1988.
- [4] J.E. Beasley. OR-Library: Distributing Test Problems by Electronic Mail. *Journal of the Operational Research Society*, 41:1069–1072, 1991.
- [5] J.E. Beasley. Lagrangean Heuristics for Location Problems. *European Journal of Operational Research*, 65:383–399, 1993.
- [6] J.E. Beasley. Lagrangean Relaxation. In: *Modern Heuristic Techniques for Combinatorial Problems*, C.R.R. Reeves, editor, pages 243–303. Blackwell, 1993.
- [7] A. Caprara, M. Fischetti, and P. Toth. A Heuristic Algorithm for the Set Covering Problem. *Operations Research*, 47:730–743, 1999.
- [8] G. Cornuejols, R. Sridharan, and J.M. Thizy. A Comparison of Heuristics and Different Relaxations for the Capacitated Plant Location Problem. *European Journal of Operational Research*, 50:280–297, 1991.
- [9] M.L. Fisher. The Lagrangian Relaxation Method for Solving Integer Programming Problems. *Management Science*, 26:1–17, 1981.
- [10] A.M. Geoffrion. Lagrangean Relaxation for Integer Programming. *Mathematical Programming Study*, 2:82–114, 1974.
- [11] F. Glover. Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers and Operations Research*, 5:533–549, 1986.
- [12] F. Glover and D. Klingman. Layering Strategies for Creating Exploitable in Linear and Integer Programs. *Mathematical Programming*, 40:165–181, 1988.
- [13] F. Glover and M. Laguna. *Tabu Search*. Kluwer, 1997.
- [14] T. Grünert. *Multi-Level Sequence-Dependent Dynamic Lotsizing and Scheduling*. Shaker, Aachen, 1998.
- [15] M. Guignard and S. Kim. Lagrangean Decomposition: A Model Yielding Stronger Lagrangean Bounds. *Mathematical Programming*, 39:215–228, 1987.
- [16] M. Held and R.M. Karp. The Traveling-Salesman Problem and Minimum Spanning Trees. *Operations Research*, 21:1138–1162, 1970.
- [17] J.P. Hiriart-Urruty and C. Lemarechal. *Convex Analysis and Minimization Algorithms II: Advanced Theory and Bundle Methods*. Springer, 1993.
- [18] Ilog Inc. Cplex Callable Library 6.5.2. Reference Manual, 1999.

- [19] S.K. Jacobsen. Heuristics for the Capacitated Plan Location Model. *European Journal of Operational Research*, 12:253–261, 1988.
- [20] B.M. Khumawala. An Efficient Heuristic Procedure for the Capacitated Warehouse Location Problem. *Naval Research Logistics Quarterly*, 21:609–623, 1974.
- [21] A.A. Kuehn and M.J. Hamburger. A Heuristic Program for Locating Warehouses. *Management Science*, 9:643–666, 1963.
- [22] G.L. Nemhauser and L.A. Wolsey. *Integer and Combinatorial Optimization*. Wiley, 1988.
- [23] R. Sridharan. A Lagrangian Heuristic for the Capacitated Plant Location Problem with Side Constraints. *Journal of the Operational Research Society*, 42:579–585, 1991.

18 A GIDS METAHEURISTIC APPROACH TO THE FLEET SIZE AND MIX VEHICLE ROUTING PROBLEM

Anthony Fu-Wha Han and Yuh-Jen Cho

Department of Transportation Engineering and Management
National Chiao Tung University
1001 Ta Hsueh Road, Hsinchu 300, Taiwan
afhan@cc.nctu.edu.tw
yjcho@chu.edu.tw

Abstract: This paper presents a new metaheuristic approach, i.e., the GIDS (Generic Intensification and Diversification Search), as well as its performance for solving the FSMVRP (Fleet Size and Mix Vehicle Routing Problem). The GIDS integrates the use of some recently developed generic search methods such as TA (Threshold Accepting) and GDA (Great Deluge Algorithm), and the meta-strategies of intensification and diversification for intelligent search. The GIDS includes three components: (1) MIC, multiple initialization constructor; (2) GSI, generic search for intensification; (3) PSD, perturbation search for diversification. A bank of twenty FSMVRP benchmark instances was tested by several different implementations of GIDS. All programs were coded in UNIX C and implemented on a SPARC 10 SUN workstation. Results are very encouraging. We have updated the best-known solutions for two of the twenty benchmark instances; the average deviation from the twenty best solutions is merely 0.598%.

18.1 INTRODUCTION

The fleet size and mix vehicle routing problem (FSMVRP), an extension of classical vehicle routing problem, determines both the vehicle fleet compositions and the routes to serve a number of customers with respective demand from a central depot. Due to the NP-hard complexity of FSMVRP, most existing methods for FSMVRP are heuristics. Golden et al. [17] adopted four modified savings methods that were derived from the conventional savings algorithm [3],

and proposed two giant tour partition methods, SGT and MGT. Gheysens et al. [11] showed a two-phase generalized assignment based approach. Desrochers and Verhoog [4] extended their matching based savings algorithm (MBSA) initially designed for the VRP to solve the FSMVRP. Salhi and Rand [28] proposed a perturbation procedure which can be considered as a two-phased local search heuristic. Osman and Salhi [26] modified the Salhi and Rand's perturbation procedure by relaxing the restriction of utilizing large-sized vehicle.

Since the 1980's, vehicle routing related researches have trended toward the era of metaheuristics [12]. A metaheuristic guides subordinate (classical) heuristics combining concepts derived from artificial intelligence, and biological, mathematical, natural and physical sciences to improve their performance in the process of search [25]. Liu and Shen [23] adopted several insertion-based savings heuristics followed by a composite improvement scheme for the FS-MVRP with time window constraints. The composite improvement scheme consisted of a perturbation procedure intending to move to a poor neighbor and an improvement procedure purely searching a better solution. Osman and Salhi [26] applied the tabu search method [15], one of the most popular metaheuristics, to solve the FSMVRP. This tabu search method used the 1-interchange (compound-moves) mechanism to generate neighborhood. Another tabu search metaheuristic for FSMVRP was proposed by Gendreau et al. [10]. This tabu search algorithm utilized GENIUS, a generalized insertion heuristic, as the subordinate search engine and embedded within an adaptive memory procedure. Several strategies, such as aspiration criterion, post-optimization and fleet change, were incorporated in the process of tabu search. The authors have generated very good results and have found many of the best-known solutions of the benchmark test instances of FSMVRP.

In this paper, we present a new metaheuristic approach, the Generic Intensification and Diversification Search (GIDS), which combines the use of generic search methods, such as TA (Threshold Accepting) [6] and GDA (Great Deluge Algorithm) [7], with the concepts of intensification and diversification strategies for solving FSMVRP. The generic search method, the term first coined by Fisher [9], can be considered as a general class of metaheuristics, which allows the search to move to a non-improving solution under certain conditions. This paper would also report the results of several different implementation procedures of GIDS for a test bank of twenty benchmark FSMVRP instances. The following of this paper is organized as follows. Section 18.2 is devoted to a sketch about the ideas and conceptual framework of GIDS. Section 18.3 describes the implementation design of our proposed metaheuristic for FSMVRP. Section 18.4 reports the contents of benchmark instances and parameters setting for testing GIDS. Section 18.5 analyzes the performance of different stages in GIDS that tested on twenty FSMVRP benchmark instances, and compares our best solution with that of other heuristics. Finally, conclusions are given in Section 18.6.

18.2 THE GIDS METAHEURISTIC: IDEAS AND FRAMEWORK

18.2.1 Some basic strategies of existing metaheuristics

The GIDS attempts to implement the ideas of many existing meta-strategies with a new framework, which does not require the memory of search history. We designed several new heuristics, i.e., the PUS (Proportional Usage Savings) and HCP (Han-and-Cho Perturbation), for its application to the FSMVRP.

In general, a metaheuristic is guided by some intelligent strategies to get away from trapped at some local optima, though the myopic move from one incumbent solution to its neighborhood solution is basically the same as conventional local search methods. The idea of “accept poor neighbor” that allows the selection of non-improving solutions under certain conditions may be the most famous meta-strategy. For example, the tabu search applies the tabu list to prevent the algorithm from cycling. Metaheuristics which adopt this “accept poor neighbor” strategy is not only limited to tabu search [12] but also shared by tabu threshold [15], simulated annealing [22], threshold accepting [6], great deluge algorithm [7], record-to-record travel [7], etc.

In addition to the idea of “accept poor neighbor”, better search results can be found by using some macroscopic approaches, such as intensification and diversification strategies of tabu search. The concepts of these approaches generally include ideas of “keep good parts” and “start from different points”. Specifically, the intensification strategy encourages the search process to choose historically found elite solutions and to initiate a return to search them more thoroughly. Examples of metaheuristic methods that adopted the intensification idea of “keep of good parts” include genetic algorithms [16] and scatter search [13]. On the other hand, the diversification strategy encourages the search process to examine unvisited regions and to generate solutions that differ in various significant ways from those seen before. Methods such as greedy random adaptive search procedure (GRASP) [8] and jump search [29] are some examples, which have applied the ideas of “start from different points”.

Metaheuristics that generate different neighborhoods during the search process belong to the category of “explore unvisited space” and may use different techniques to achieve their purpose. One example is the variable neighborhood search [24], which applies a sequence of different local search heuristics to change the search space. Other examples include the noising method [2], search space smoothing [18], and flip-flop method [19], which alter the search space by disturbing the cost functions. More details of these metaheuristics could be referred to Glover and Laguna [15], Osman and Kelly [25], and Reeves [27].

As one of the most important metaheuristics, tabu search integrates both the intensification and diversification meta-strategies to guide the search process and performs well in solving many combinatorial optimization problems. However, an efficient implementation of tabu search in general relies on a good memory structure of the search history. We thus present the GIDS as an alternative metaheuristic framework, which can be implemented efficiently without complicate memory structures.

18.2.2 The conceptual framework of the GIDS

As Battiti and Tecchioli [1] pointed out, in the process of searching for the global optimum of a combinatorial optimization problem, three phenomena may occur and must be conquered. Specifically, these three search syndromes are: (1) local optima which tend to trap a local search heuristic; (2) limited cycles which may lead the search process repeat a sequence of solutions; and (3) chaotic attractors which may contract or distort the solution space so that the search process will visit only a limited part of the solution space. Therefore, the effectiveness of a metaheuristic will rely heavily on its ability to deal with these search syndromes. The intensification and diversification strategies as mentioned earlier, when applied appropriately, should provide effective tools for intelligent search.

The conceptual structure of the GIDS is basically built around the ideas of how to overcome the aforementioned Battiti and Tecchioli's search syndromes with efficient and yet easy-to-apply heuristic algorithms. Accordingly, GIDS integrates the use of some recently developed generic search methods [9], and the meta-strategies of intensification and diversification. As shown in Figure 18.1, the GIDS system includes three major components: (1) Multiple Initialization Constructor (MIC), (2) Generic Search for Intensification (GSI), and (3) Perturbation Search for Diversification (PSD). Each component, together with its implementation procedures, is designed to overcome a specific syndrome. Note that though GIDS shares some ideas of intensification and diversification strategies from tabu search, the implementation of GIDS is very different to tabu search. We applied generic search methods and the idea of variable neighborhood search [24] to implement the intensification strategy. Additionally, the strategies of "explore unvisited space" and "start from different points" are adopted for diversification purpose.

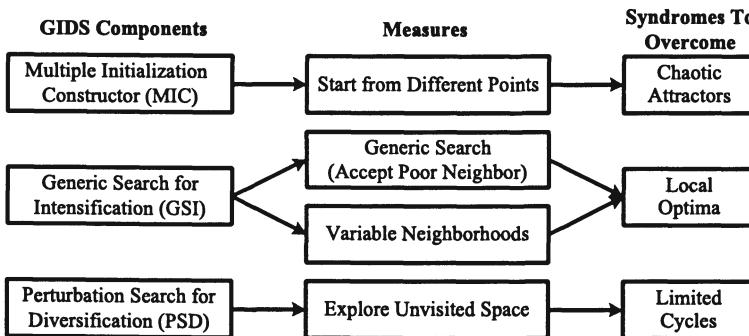


Figure 18.1 The conceptual framework of GIDS.

18.3 IMPLEMENTATION DESIGN OF GIDS FOR FSMVRP

The three components of GIDS each takes a modular design. The MIC includes two modules, weighted initialization (WI) and neighborhood search (NS); the GSI includes two generic search modules, G1 and G2; and the PSD includes the module of cost perturbation (CP). As shown in Figure 18.2, the GIDS iteratively runs through these modules with two loops, which are designed respectively for intensification search and diversification search purposes. Details of the implementation design of these five modules are described next.

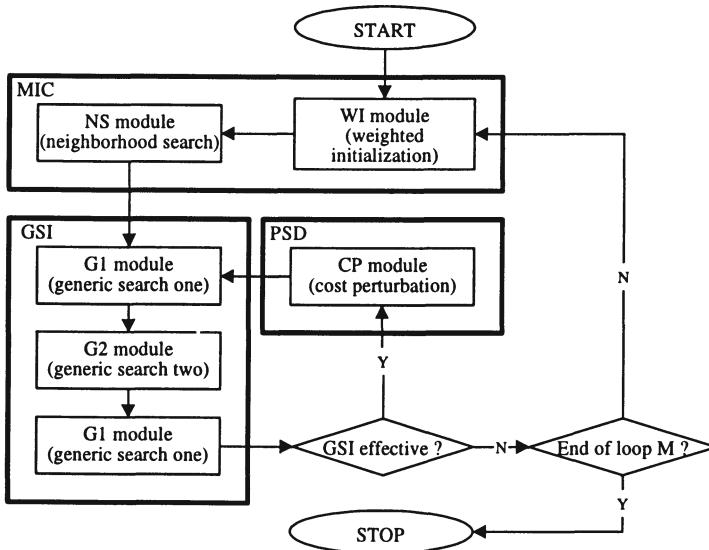


Figure 18.2 The implementation design of GIDS for FSMVRP.

18.3.1 Weighted Initialization (WI) module

The WI module provides the tool for systematically generating various start points. Specifically, the WI module assigns a different weight to the modified savings methods adopted in the WI module. We adopted four modified savings methods to build the WI module. Three of them are the conventional modified savings proposed by Golden et al. [17], i.e., combined savings (CS), optimistic opportunity savings (OOS), and realistic opportunity savings (ROS). In addition to these conventional methods, we designed a new modified savings heuristic, i.e., proportional usage savings (PUS), to exploit the potential savings using the vehicle type with least average fixed cost. Table 18.1 summarizes these four modified savings formulas, where the notations are defined as follows:

C_{ij} : the travel cost of link (i, j) ,

$S_{ij} = C_{i0} + C_{0j} - C_{ij}$, is the Clarke and Wright savings value of travel cost by connecting link (i, j) ,

Z_i : the load of the vehicle route which covers node i ,

$F(Z)$: the fixed cost of the smallest vehicle that can fulfil the demand of Z ,

$P(Z)$: the capacity of the smallest vehicle that can fulfil the demand of Z ,

$F'(Z)$: the fixed cost of the largest vehicle that has a capacity less than or equal to Z ,

$$W = P(Z_i + Z_j) - P(\max\{Z_i, Z_j\}),$$

$$\delta(W) = 0, \text{ if } W = 0; \delta(W) = 1, \text{ if } W > 0, \text{ and}$$

$U(Z) = F(Z) \cdot [Z/P(Z)]$, is the proportional fixed cost of the smallest vehicle that can fulfil the demand of Z .

Method	Savings formula	
CS	$CS_{ij} = S_{ij} + F(Z_i) + F(Z_j) - F(Z_i + Z_j)$	(18.1)
OOS	$OS_{ij} = CS_{ij} + F[P(Z_i + Z_j) - Z_i - Z_j]$	(18.2)
ROS	$RS_{ij} = CS_{ij} + \delta(W) \cdot F'[P(Z_i + Z_j) - Z_i - Z_j]$	(18.3)
PUS*	$US_{ij} = S_{ij} + U(Z_i) + U(Z_j) - U(Z_i + Z_j)$	(18.4)

* First proposed in this work.

Table 18.1 Summary of savings methods.

Table 18.2 shows how the WI method regulates the weights of travel cost and vehicle cost in savings formula of WI modules, where $w(m)$ is the weight of iteration m , and

$$w(m) = \begin{cases} 0.5 & \text{for } M = 1 \\ (m-1)/(M-1) & \text{for } M \geq 2 \end{cases}, m = 1, \dots, M. \quad (18.5)$$

M is a predefined number of restart iterations. Based on the updated (weighted) savings, the WI module would accordingly generate a new initial solution, and to repeat the whole GIDS once again. Both sequential and parallel savings were tested for each method in Table 18.2.

18.3.2 Neighborhood search (NS) module

The NS module has two local search heuristics, i.e., SRP (Salhi-and-Rand perturbation) and HCP (Han-and-Cho perturbation). Both SRP and HCP are modified from the perturbation method proposed by Salhi and Rand [28]. The SRP has seven refinement procedures, which includes reduction, reallocation, combining, sharing, swapping, relax/comb, and relax/share, to improve an initial solution. These refinement procedures can be regarded as variants of traditional inter-route exchange methods. The HCP module is a simplified version of

Method	Weighted savings formula
CS	$CS_{ij}(m) = w(m) \cdot S_{ij} + [1 - w(m)][F(Z_i) + F(Z_j) - F(Z_i + Z_j)] \quad (18.6)$
OOS	$OS_{ij}(m) = CS_{ij} + [1 - w(m)]\{F[P(Z_i + Z_j) - Z_i - Z_j]\} \quad (18.7)$
ROS	$RS_{ij}(m) = CS_{ij} + \delta(W) \cdot [1 - w(m)]\{F'[P(Z_i + Z_j) - Z_i - Z_j]\} \quad (18.8)$
PUS	$US_{ij}(m) = w(m) \cdot S_{ij} + [1 - w(m)][U(Z_i) + U(Z_j) - U(Z_i + Z_j)] \quad (18.9)$

Table 18.2 Summary of weighted savings methods.

SRP, and contains four refinement procedures: reallocation, relax/share, swapping, and relax/comb. Two more variant procedures, i.e., SRPX and HCPX, are also designed based on SRP and HCP respectively, to allow for adjusting vehicle sizes while inserting or exchanging nodes. Such a design of four different search methods is to provide us with alternative tools for a more flexible search with different neighborhoods while implementing GSI or PSD.

There are two ways to implement a local neighborhood search method: the best-improvement and the first-improvement. The best-improvement is to find the best solution in the whole neighborhood of the incumbent solution. The first-improvement is to find the first solution, among the neighborhood, which will be accepted by the associated rule. Both implementations were tested for various NS modules; the better ones were adopted accordingly.

18.3.3 Generic search modules: G1 and G2

In GSI, both strategies of "accept poor neighbor" and "variable neighborhoods" are embedded in the procedure. Deterministic generic search methods, such as threshold accepting (TA) and great deluge algorithm (GDA), are adopted because their capabilities to overcome the trap of local optima. In these generic search methods, generic rules of accepting inferior neighboring solutions are applied to guide the subordinate NS modules as mentioned earlier. For better intensification, we applied two different generic search modules, G1 and G2, in the GSI process. G1 is more elaborated than G2, and serves as the major search engine for intensification in the GSI subsystem. During the implementation of GSI, G1 is implemented twice with G2 sandwiched in between. This idea is similar to that of variable neighborhood search [23].

18.3.4 Cost perturbation (CP) module

The PSD subsystem includes the module of cost perturbation (CP). The CP module is based on the strategy of changing search space by perturbing the cost function. In CP, we applied the flip-flop (FF) method [19] by assigning a minus sign to the cost function and then go to repeat the NS module. The original minimization problem now essentially becomes a maximization problem. According CP tends to move from a solution of GSI to a drastic different

solution, which may be distant from the neighborhoods previously searched by the GSI process of G1 and G2 (see Figure 18.3).

Figure 18.3 gives an abstract presentation of the search movement in GIDS, where each of the small circles denotes the neighborhood of a local or neighborhood search (NS), and each of the big circles denotes the region explored by a generic search module.

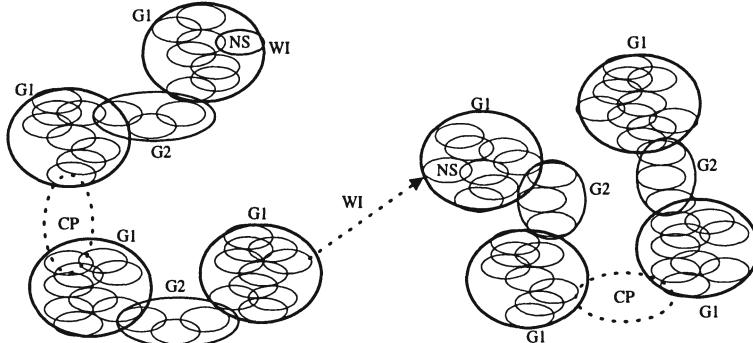


Figure 18.3 An abstract presentation of GIDS search process.

18.4 APPLICATION OF GIDS TO FSMVRP

18.4.1 Benchmark instances and evaluation criteria

We use the set of twenty benchmark instances compiled by Golden et al. [17] for evaluation and comparison analysis of GIDS for FSMVRP. The size of these instances varies from 12 to 100 points not including the depot as shown in Table 18.3. The best-known solutions in the published literature are also summarized in the table. Computer programs of GIDS for FSMVRP were coded in UNIX C language and ran at the SPARC 10 SUN Workstation. The criteria selected to measure the performance of the proposed metaheuristic are the quality of solution and the computational effort. The quality of solution calculates the percentage of deviation between the best-known solution and the solutions found by GIDS. The computational effort is measured by seconds of computer's CPU time. To avoid confusion of accuracy measure due to different round-off errors on different computational platforms, in our implementation we adopted the all-integer format for the input cost data of all the test instances.

18.4.2 Parameter settings and experimental design

As mentioned earlier, the GIDS consists of three phases: MIC, GSI, and PSD. Implementation of different heuristics modules requires specific settings of related parameters. The related heuristics and parameters selected for computational testing of GIDS in three phases are listed in Table 18.4. As shown in the table, parameters of GDA include initial water level (L) and rain down speed

No.	Size	Veh. [†]	Published best-known solutions		
			Cost	Method	Reference
1	12	3	602	ROM	Desrochers (1991) [4]
2	12	3	722	SRP	Salhi & Rand (1993) [28]
3	20	5	960	TS2	Gendreau et al. (1999) [10]
4	20	3	6436	TS2	Gendreau et al. (1999) [10]
5	20	5	1007	TS2	Gendreau et al. (1999) [10]
6	20	3	6515	TS2	Gendreau et al. (1999) [10]
7	30	5	7298	VRP solution [†]	Golden et al. (1984) [17]
8	30	4	2347	IBS	Liu & Shen (1999) [23]
9	30	5	2209	SRP	Salhi & Rand (1993) [28]
10	30	4	2363	TS1	Osman & Salhi (1996) [26]
11	30	4	4755	TS1	Osman & Salhi (1996) [26]
12	30	6	4092	SRP	Salhi & Rand (1993) [28]
13	50	6	2405	TS2	Gendreau et al. (1999) [10]
14	50	3	9115	TS2	Gendreau et al. (1999) [10]
15	50	3	2583	TS2	Gendreau et al. (1999) [10]
16	50	3	2741	TS2	Gendreau et al. (1999) [10]
17	75	5	1747	TS2	Gendreau et al. (1999) [10]
18	75	6	2377	TS2	Gendreau et al. (1999) [10]
19	100	3	8663	TS2	Gendreau et al. (1999) [10]
20	100	3	4076	TS2	Gendreau et al. (1999) [10]

[†]: types of vehicles.

[†]: best-known solution comes from the result of original VRP.

Table 18.3 Summary of FSMVRP benchmark test instances.

(S); and parameters of TA include initial threshold (T) and length of threshold sequence (K).

Those notations given in the fourth column of Table 18.4 indicate the detailed combinations of heuristic modules implemented at each stage. For example, “CS_s” denotes the sequential implementation of the combined savings (CS) module. Similarly, “SRP_b” is SRP with best-improvement; and “CP(FF_SRP)” represent the CP module that executes FF based on best-improvement SRP. Every kind of implementing combination is executed to all of the twenty instances.

18.5 PERFORMANCE ANALYSIS AND EVALUATION

18.5.1 Accuracy performance of different phases in GIDS

In this section we would like to compare the accuracy performance of different modules during the implementation of GIDS. Figure 18.4 shows the trend of average accuracy performance against the cumulative CPU time at different stages during the implementation process of GIDS based on WI(PUS_p), NS(HCP), G1(GDA_HCP), G2(TA_HCP) and CP(FF_HCP). Though the trend in Figure 18.4 represents a diminishing return of marginal contributions for subsequent stages of the implementation of the GIDS, the contribution of each subsequent module still can be identified. In the first iteration of GIDS ($M = 1$),

Phases	Modules	Heuristics (parameters)	Implementing combinations
I (MIC)	WI	sequential: CS, OOS, ROS, PUS parallel: CS, OOS, ROS, PUS ($M = 1, \dots, 10$)	CS_S, PUS_p, etc.
	NS	best-improvement: SRP, HCP	SRP_b, HCP_b
II (GSI)	G1	GDA ($L = X_0 \times 1.2, S = X_0 \times 0.01$)* TA ($T = X_0 \times 0.2, K = 10$)	G1(GDA_SRPA), G1(GDA_HCP), G1(TA_SRPA) and G1(TA_HCP)
	G2	best-improvement: HCP, SRP first-improvement: HCPX, SRPX	
		TA ($T = X_0 \times 0.2, K = 30$) GDA ($L = X_0 \times 1.2, S = X_0 \times 0.01$)	G2(TA_SRPA), G2(TA_HCP), G2(GDA_SRPA) and G2(GDA_HCP)
	CP	FF + first-improvement: SRP, HCP	CP(FF_SRPA) and CP(FF_HCP)
III (PSD)			

* X_0 is the objective value of the initial solution.

Table 18.4 Parameter settings of GIDS for FSMVRP.

the average percentage error reduces from 12.912% of WI module to 5.550% of NS module. Such results at the initial solution stage can be considered fair to good. The second phase of GSI further reduces the average percentage error from 5.550% to 3.297%, which represents a significant improvement of 40% error reduction. This result indicates the effectiveness of the GSI procedure. Then, for the PSD phase, the average percentage error reduces from 3.297% to 2.202% which again presents a more than 30% error reduction. As shown in Figure 18.4, the average computer time to obtain this result in the first iteration ($M=1$) of GIDS, is about 10 CPU seconds. The marginal contribution of the consecutive MIC runs, i.e., $M = 2, 3, \dots, 10$, tends to be less significant. Still, the overall average deviation reduces from 2.202% for $M=1$, to 1.471% for $M=10$. The average computer time for ten iterations ($M=10$) is about 58 CPU seconds. All these results imply an overall consistent and effective system structure of our proposed GIDS.

Note that both the TA and GDA generic search algorithms used in our GSI component involve pre-settings of algorithmic parameters such as T, K, L and S . Since it is not our purpose to strike out the best-fitted parameters for the benchmark instances, we simply applied the settings as listed in Table 18.4 to explore the applicability of our proposed GIDS to the FSMVRP. Based on the results of these settings, we also obtained the following findings. First, for those eight WI heuristics tested, our proposed PUS performs better than other three conventional weighted savings methods both in sequential and parallel implementations. Second, for the NS heuristics, our proposed HCP, which is much easier to implement than SRP, yielded as good results as SRP. Third, for generic search heuristics, GDA slightly performs better than TA both in G1 and G2 modules.

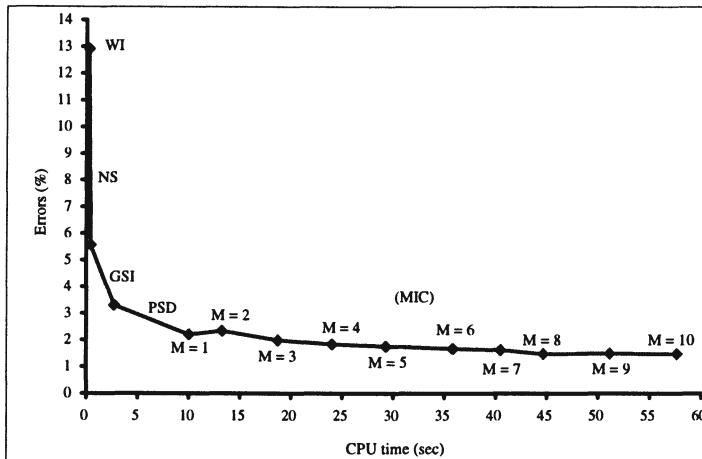


Figure 18.4 Trend of consistent improvement of GIDS.

18.5.2 Comparison of GIDS with other heuristics

Table 18.5 summarizes the best performance of GIDS for all twenty instances. The table also compares the computational results of GIDS with that of five other well-performed FSMVRP heuristics: MGT+Or [17], ROM [4], SRPP [28], TS1 [26], IBS [23] and TS2 [10]. To avoid round-off errors due to different computer implementations, we applied integer costs in our study as those in [4, 17, 23, 28]. For consistency, we converted the real values of TS2 results into integer numbers based on the tour configurations of their solutions given in [10]. The results of those published heuristics are listed and compared with our results in Table 18.5. As shown in the table, GIDS generated fairly good results; the average deviation of our best results from the published best results is merely 0.598%.

As compared to the published best-known solutions, using GIDS we have found two new best-known solutions of the test instances No. 7 and No. 10. The boldfaced numbers in the last column of Table 18.5 indicate new best solutions found by GIDS. The details of our best results for the test instances No. 7 and No. 10 are given in the Appendix. More detailed results including the cost, the route composition, and the route configuration for those instances can be found at the web site of <http://www.tem.nctu.edu.tw/~network> developed by the authors.

18.6 CONCLUSIONS

In this paper, we proposed a new metaheuristic, the GIDS, which combined generic search methods and meta-strategies of intensification and diversification for solving the fleet size and mixed vehicle routing problem (FSMVRP).

No	Size	Total cost						Updated best known sol.	
		MGT+O	ROM	SRPP	TS1	IBS	TS2 [†]	GIDS [†]	
1	12	622	606	614	602	602	—	602 <0.000>	602
2	12	722	730	722	722	722	—	722 <0.000>	722
3	20	966	990	1003	971.24	972	961.03 (960)	960 <0.000>	960
4	20	6930	6547	6447	6445.10	6445	6437.33 (6436)	6436 <0.000>	6436
5	20	1013	1040	1015	1009.15	1014	1007.05 (1007)	1007 <0.000>	1007
6	20	6974	6517	6516	6516.56	6516	6516.46 (6515)	6515 <0.000>	6515
7	30	7389	7421	7402	7310	7313	—	7284 <-0.192>	7284
8	30	2367	2387	2367	2348	2347	—	2347 <0.000>	2347
9	30	2220	2231	2209	2209	2215	—	2209 <0.000>	2209
10	30	2370	2393	2377	2363	2368	—	2356 <-0.296>	2356
11	30	4763	4862	4819	4755	4778	—	4756 <0.021>	4755
12	30	4136	4254	4092	4092	4101	—	4092 <0.000>	4092
13	50	2438	2525	2493	2471.07	2465	2408.41 (2405)	2452 <1.954>	2405
14	50	9132	9155	9153	9125.65	9132	9119.03 (9115)	9145 <0.329>	9115
15	50	2640	2622	2623	2606.72	2608	2586.37 (2583)	2593 <0.387>	2583
16	50	2822	2809	2765	2745.01	2809	2741.50 (2741)	2766 <0.912>	2741
17	75	1783	1877	1767	1762.05	1806	1749.50 (1747)	1810 <3.606>	1747
18	75	2432	2489	2439	2412.56	2416	2381.43 (2377)	2427 <2.103>	2377
19	100	8720	8700	8751	8685.71	8684	8675.16 (8663)	8679 <0.185>	8663
20	100	4195	4248	4187	4188.73	4148	4086.76 (4076)	4196 <2.944>	4076
Average =									0.598

[‡]: numeral in parentheses is the cost of the integral solution.

[†]: numeral in brackets is the percentage error of our result with respect to the best-known solution.

Table 18.5 Comparison of computational results with published heuristics.

The GIDS is composed of three components, i.e., multiple initialization constructor (MIC), generic search for intensification (GSI) and perturbation search for diversification (PSD), which is designed to overcome three searching syndromes. As compared to the well-known tabu search, GIDS shares some similar ideas in the concepts of intensification and diversification but does not require complicated memory scheme for computer implementation.

We implemented the GIDS for a testing bank of twenty benchmark instances, and found good results comparable to other first class metaheuristics approaches, tabu search in particular. The average CPU time of GIDS implementation for one iteration ($M=1$) over the twenty tested instances is about 10 seconds, and the average deviation is 2.20%. The average deviation reduces to 1.47% for ten iterations ($M = 10$) of GIDS, and the CPU time is about 58 seconds. Among the twenty benchmark instances tested, GIDS has found new best solutions for two instances. The average deviation of our best results with respect to the best published results is merely 0.598%. Such results imply that the GIDS may provide an efficient tool for real world FSMVRP and related problems.

Acknowledgments: This research was supported by the National Science Council (NSC-87-2211-E-009-024) of Taiwan, R.O.C.

References

- [1] B. Battiti and G. Tecchiolli. The Reactive Tabu Search. *ORSA Journal on Computing*, 6:126–140, 1994.
- [2] I. Charon and O. Hudry. The Noising Method: A New Method for Combinatorial Optimization. *Operations Research Letters*, 14:133–137, 1993.
- [3] G. Clarke and J.W. Wright. Scheduling of Vehicles From a Central Depot to a Number of Delivery Points. *Operations Research*, 12:568–589, 1964.
- [4] M. Desrochers and T.W. Verhoog. A New Heuristic for the Fleet Size and Mix Vehicle Routing Problem. *Computers and Operations Research*, 18:263–274, 1991.
- [5] M. Dorigo, V. Maniezzo, and A. Colomi. The Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Transactions on Systems, Man, and Cybernetics*, 26:29–41, 1996.
- [6] G. Dueck and T. Scheuer. Threshold Accepting: A General Purpose Optimization Algorithm Appearing Superior to Simulated Annealing. *Journal of Computational Physics*, 90:161–175, 1990.
- [7] G. Dueck. New Optimization Heuristics: The Great Deluge Algorithm and the Record-to-Record Travel. *Journal of Computational Physics*, 104:86–92, 1993.
- [8] T.A. Feo and M.G.C. Resende. A Probabilistic Heuristic for a Computationally Difficult Set Covering Problem. *Operations Research Letters*, 8:67–71, 1989.
- [9] M.L. Fisher. Vehicle Routing. In: *Network Routing*, M. Ball, T. Magnanti, C. Monma, and G. Nemhauser, editors, pages 1–33, Elsevier, 1995.
- [10] M. Gendreau, G. Laporte, C. Musaraganyi, and E.D. Taillard. A Tabu Search Heuristic for the Heterogeneous Fleet Vehicle Routing Problem. *Computers and Operations Research*, 26:1153–1173, 1999.
- [11] F.G. Gheysens, B.L. Golden, and A. Assad. A New Heuristic for Determining Fleet Size and Composition. *Mathematical Programming Studies*, 26:233–236, 1986.
- [12] F. Glover. Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers and Operations Research*, 13:533–549, 1986.
- [13] F. Glover. Genetic Algorithms and Scatter Search: Unsuspected Potentials. *Statistics and Computing*, 4:131–140, 1994.
- [14] F. Glover. Tabu Thresholding: Improved Search by Non-Monotonic Trajectories, *ORSA Journal on Computing*, 7:426–442, 1995.

- [15] F. Glover and M. Laguna. *Tabu Search*. Kluwer, 1997.
- [16] D.E. Goldberg and R. Lingle. Alleles, Loci and Travelling Salesman Problem. In: *Proceedings of an International Conference on Genetic Algorithms and their Applications*, J.J. Grefenstette, editor, pages 154–159, 1985.
- [17] B.L. Golden, A. Assad, L. Levy, and F. Gheysens. The Fleet Size and Mix Vehicle Routing Problem. *Computers and Operations Research*, 11:49–66, 1984.
- [18] J. Gu and X. Huang. Efficient Local Search with Search Space Smoothing: A Case Study of the Traveling Salesman Problem. *IEEE Transactions on Systems, Man, and Cybernetics*, 24:728–736, 1994.
- [19] A.F. Han and G.C. Chen. Applications of Noising Method and Flip-flop Method to Travelling Salesman Problem (in Chinese). Working Paper, Department of Transportation Engineering and Management, National Chiao Tung University, Taiwan, 1996.
- [20] A.F. Han and Y.J. Cho. Hybrid Heuristic Methods for Complicate Vehicle Routing Problems: Applications to FSMVRP (in Chinese). Report No. NSC-87-2211-E-009-024, Department of Transportation Engineering and Management, National Chiao Tung University, Taiwan, 1998.
- [21] J.J. Hopfield and D.W. Tank. Neural Computation of Decisions in Optimization Problems. *Biological Cybernetics*, 52:141–152, 1985.
- [22] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671–680, 1983.
- [23] F. Liu and S. Shen. The Fleet Size and Mix Vehicle Routing Problem with Time Windows. *Journal of the Operational Research Society*, 50:721–732, 1999.
- [24] N. Mladenović and P. Hansen. Variable Neighborhood Search. *Computers and Operations Research*, 24:1097–1100, 1997.
- [25] I.H. Osman and J.P. Kelly. Metaheuristics: An Overview. In: *Metaheuristics: Theory and Applications*, I.H. Osman and J.P. Kelly, editors, pages 1–21, Kluwer, 1996.
- [26] I.H. Osman and S. Salhi. Local Search Strategies for the Vehicle Fleet Mix Problem. In: *Modern Heuristic Search Methods*, V.J. Rayward-Smith, I.H. Osman, C.R. Reeves, and G.D. Smith, editors, pages 131–153, Wiley, 1996.
- [27] C.R. Reeves (editor). *Modern Heuristics Techniques for Combinatorial Problems*, Wiley, 1993.
- [28] S. Salhi and G.K. Rand. Incorporating Vehicle Routing into the Vehicle Fleet Composition Problem. *European Journal of Operational Research*, 66:313–330, 1993.

- [29] S. Tsubakitani and J.R. Evans. An Empirical Study of a New Metaheuristics for the Traveling Salesman Problem. *European Journal of Operational Research*, 104:113–128, 1998.

APPENDIX:

NEW BEST RESULTS OF TEST INSTANCES NO. 7 AND NO. 10

(1) Best solution of the instance 7

Nodes: 30	Cost matrix	Published best-known solution: 7298
Types of vehicle: A, B, C, D, E		Best solution found by GIDS: 7284

Route 1 (A): 0 - 02 - 0
Route 2 (B): 0 - 24 - 28 - 0
Route 3 (B): 0 - 01 - 12 - 18 - 0
Route 4 (B): 0 - 21 - 08 - 25 - 0
Route 5 (B): 0 - 03 - 07 - 14 - 09 - 15 - 16 - 11 - 05 - 06 - 22 - 0
Route 6 (C): 0 - 20 - 30 - 0
Route 7 (C): 0 - 04 - 29 - 23 - 0
Route 8 (C): 0 - 13 - 27 - 26 - 17 - 0
Route 9 (D): 0 - 10 - 19 - 0

(2) Best solution of the instance 10

Nodes: 30	Cost matrix	Published best-known solution: 2363
Types of vehicle: A, B, C, D		Best solution found by GIDS: 2356

Route 1 (B): 0 - 02 - 01 - 17 - 0
Route 2 (D): 0 - 10 - 19 - 0
Route 3 (D): 0 - 21 - 30 - 25 - 0
Route 4 (D): 0 - 24 - 29 - 28 - 0
Route 5 (D): 0 - 08 - 26 - 27 - 18 - 0
Route 6 (D): 0 - 23 - 22 - 03 - 04 - 06 - 05 - 16 - 11 - 13 - 07 - 15 - 09 - 14 - 12 - 20 - 0

Detailed results of other test instances can be found at our web site at
<http://www.tem.nctu.edu.tw/~network>.

19

DEVELOPMENTS OF VARIABLE NEIGHBORHOOD SEARCH

Pierre Hansen¹ and Nenad Mladenović^{1,2}

¹GERAD and École des Hautes Études Commerciales
3000, chemin de la Côte-Sainte-Catherine
Montréal, Canada, H3T 2A7
pierreh@crt.umontreal.ca

²Mathematical Institute, Serbian Academy of Science
Kneza Mihajla 35, 11000 Belgrade, Yugoslavia
nenad@crt.umontreal.ca

Abstract: Proposed just a few years ago, Variable Neighborhood Search (VNS) is a new metaheuristic for combinatorial and global optimization, based upon systematic change of neighborhood within a possibly randomized local search. Its development has been rapid. After reviewing the basic scheme of VNS, several extensions aimed at solving large problem instances are surveyed. Issues in devising a VNS heuristic are discussed. Finally, recent applications are briefly summarized, with special attention given to VNS-based computer-aided discovery of conjectures in graph theory.

19.1 INTRODUCTION

Metaheuristics, or general frameworks for building heuristics to solve a variety of problems, are usually based upon a central idea, or analogy. Then, they are developed, extended in various directions and possibly hybridized. The resulting heuristics often get complicated, and use many parameters. This may enhance their efficiency but obscures the reasons of their success.

Variable Neighborhood Search, a metaheuristic invented just a few years ago [59, 61] is based upon a simple, but then (and to some extent still now) little explored principle: systematic change of neighborhood within the search. Its development has been rapid, with more than two dozen papers already published or to appear. Several extensions have been made, mainly to allow

solution of large problem instances. In all of them, an effort has been made to keep the simplicity of the basic scheme. Also, hybrids, while they certainly hold potential, have been left on the back-burner. In contrast, applications have been numerous and successfull. Moreover, they have given some insight on issues involved in the design of efficient VNS heuristics.

In this paper, we survey these developments. The basic rules of VNS are recalled in the next section. Then, recent extensions designed for solving large problem instances are presented, in Section 19.3. Issues in devising efficient VNS heuristics are discussed in Section 19.4. Applications are reviewed in Section 19.5. A new VNS-based approach to computer-aided discovery of conjectures in graph theory is discussed in Section 19.6.

19.2 BASIC SCHEMES

Let us denote with \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., $k_{max} = 1$.) Steps of the basic VNS are presented on Figure 19.1. They make use of a local search routine discussed later.

Initialization. Select the set of neighborhood structures \mathcal{N}_k , for $k = 1, \dots, k_{max}$, that will be used in the search; find an initial solution x ; choose a stopping condition;

Repeat the following sequence until the stopping condition is met:

- (1) Set $k \leftarrow 1$;
- (2) Repeat the following steps until $k = k_{max}$:
 - (a) *Shaking.* Generate a point x' at random from the k^{th} neighborhood of x ($x' \in \mathcal{N}_k(x)$);
 - (b) *Local search.* Apply some local search method with x' as initial solution; denote with x'' the so obtained local optimum;
 - (c) *Move or not.* If this local optimum is better than the incumbent, move there ($x \leftarrow x''$), and continue the search with \mathcal{N}_1 ($k \leftarrow 1$); otherwise, set $k \leftarrow k + 1$;

Figure 19.1 Steps of the basic VNS.

The stopping condition may be e.g. maximum CPU time allowed, maximum number of iterations, or maximum number of iterations between two improvements. Often successive neighborhoods \mathcal{N}_k will be nested. Observe that point x' is generated at random in step 2a in order to avoid cycling, which might occur if any deterministic rule was used.

As a local optimum within some neighborhood is not necessarily one within another, change of neighborhoods can be performed fruitfully during the local search phase too. In some cases, as when applying VNS to graph theory, the use of many neighborhoods in the local search is crucial. This local search is

then called Variable Neighborhood Descent (VND) and its steps are presented on Figure 19.2.

Initialization. Select the set of neighborhood structures \mathcal{N}'_k , for $k = 1, \dots, k'_{\max}$, that will be used in the descent; find an initial solution x (or apply the rules to a given x);

Repeat the following sequence until no improvement is obtained:

- (1) Set $k \leftarrow 1$;
- (2) Repeat the following steps until $k = k'_{\max}$:
 - (a) *Exploration of neighborhood.* Find the best neighbor x' of x ($x' \in \mathcal{N}'_k(x)$);
 - (b) *Move or not.* If the solution thus obtained x' is better than x , set $x \leftarrow x'$ and $k \leftarrow 1$; otherwise, set $k \leftarrow k + 1$;

Figure 19.2 Steps of the basic VND.

19.3 SOLVING LARGE PROBLEM INSTANCES BY VNS

While the basic VNS is clearly useful for approximate solution of many combinatorial and global optimization problems, it remains difficult or long to solve very large instances. As often, size of problems considered is limited in practice by the tools available to solve them more than by the needs of potential users of these tools. Hence, improvements appear to be highly desirable. Moreover, when heuristics are applied to really large instances their strengths and weaknesses become clearly apparent.

Four improvements of the basic VNS are considered in this section:

Reduced VNS where the local search step 2b of the basic VNS is simply skipped, aims at increasing effectiveness (or speed), at the possible cost of an increase in solution value. RVNS is useful for very large instances for which local search is costly. It is akin to a Monte-Carlo method, but more systematic. Its relationship to the Monte-Carlo method is the same as that of basic VNS to Multistart local search. When applied to the p -median problem, RVNS gave equally good solutions than the *Fast Interchange* heuristic of Whittaker [71] in 20 to 40 times less time [47].

Variable neighborhood decomposition search [47] extends basic VNS into a two-level VNS scheme based upon decomposition of the problem. Its steps are presented on Figure 19.3.

Note that the only difference between the basic VNS and VNDS is in step 2b: instead of applying some local search method in the whole solution space \mathcal{S} (starting from $x' \in \mathcal{N}_k(x)$), in VNDS we solve at each iteration a subproblem in some subspace $V_k \subseteq \mathcal{N}_k(x)$ with $x' \in V_k$. When the local search used in this step is also VNS, the two-level VNS-scheme arises.

Initialization. Select the set of neighborhood structures \mathcal{N}_k , for $k = 1, \dots, k_{max}$, that will be used in the search; find an initial solution x ; choose a stopping condition;

Repeat the following sequence until the stopping condition is met:

- (1) Set $k \leftarrow 1$;
 - (2) Repeat the following steps until $k = k_{max}$:
 - (a) *Shaking.* Generate a point x' at random from the k^{th} neighborhood of x ($x' \in \mathcal{N}_k(x)$); in other words, let y be a set of k solution attributes present in x' but not in x ($y = x' \setminus x$).
 - (b) *Local search.* Find the local optimum in the space of y either by inspection or by some heuristic; denote the best solution found with y' and with x'' the corresponding solution in the whole space S ($x'' = (x' \setminus y) \cup y'$);
 - (c) *Move or not.* If the solution thus obtained is better than the incumbent, move there ($x \leftarrow x''$), and continue the search with \mathcal{N}_1 ($k \leftarrow 1$); otherwise, set $k \leftarrow k + 1$;
-

Figure 19.3 Steps of the basic VNDS.

Skewed VNS. The third extension 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 can then degenerate, to some extent, into the multistart heuristic (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 [40]. Its steps are presented in Figure 19.4.

SVNS makes use of a function $\rho(x, x'')$ to measure distance between the incumbent solution x and the local optimum found x'' . The distance used to define the \mathcal{N}_k , as in the above examples, could be used also for this purpose. The parameter α must be chosen in order to accept exploring valleys far from x when $f(x'')$ is larger than $f(x)$ but not too much (otherwise one will always leave x). A good value 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 large value for α when $\rho(x, x'')$ is small. More sophisticated choices for a function $\alpha\rho(x, x'')$ could be made through some learning process.

Parallel VNS. Several ways for parallelizing VNS have recently been proposed by López et al. [57] and Crainic et al. [19] in solving the p -Median problem. López et al. [57] have tested three of them: (i) parallelize local search; (ii) augment the number of solutions drawn from the current neighborhood and do local search in parallel from each of them and (iii) do the same as (ii) but updating the information about the best solution found. The second

Initialization. Select the set of neighborhood structures \mathcal{N}_k , for $k = 1, \dots, k_{max}$, that will be used in the search; find an initial solution x and its value $f(x)$; set $x_{opt} \leftarrow x$, $f_{opt} \leftarrow f(x)$; choose a stopping condition and a parameter value α ;

Repeat the following until the stopping condition is met:

- (1) Set $k \leftarrow 1$;
 - (2) Repeat the following steps until $k = k_{max}$:
 - (a) *Shaking.* Generate a point x' at random from the k^{th} neighborhood of x ;
 - (b) *Local search.* Apply some local search method with x' as initial solution; denote with x'' the so obtained local optimum;
 - (c) *Improve or not.* If $f(x'') < f_{opt}$ set $f_{opt} \leftarrow f(x'')$ and $x_{opt} \leftarrow x''$;
 - (d) *Move or not.* If $f(x'') - \alpha\rho(x, x'') < f(x)$ set $x \leftarrow x''$ and $k \leftarrow 1$; otherwise set $k \leftarrow k + 1$;
-

Figure 19.4 Steps of the Skewed VNS.

version gave the best results. Crainic et al. [19] show that assigning different neighborhoods to each processor without interrupting their work as soon as an improved solution is found gives very good results, summarized in Table 19.1. Optimal solutions (when known) were systematically reproduced and better solution than previously found obtained in 4 cases out of 5.

19.4 ISSUES IN DEVISING A VNS HEURISTIC

When using more than one neighborhood structure in the search, the following problem specific questions have to be answered:

- (i) how many neighborhoods should be used and which?
- (ii) what should be their order in the search?
- (iii) what strategy should be used in changing neighborhoods?

Application of the VNS metaheuristic for solving each particular problem is based on answers to these questions. In this section we give some suggestions about how to deal with them.

Selection of neighborhood structures. As mentioned above, neighborhood structures, in general, can be induced from different metrics introduced into the solution space. However, there are easy ways to do so. We next list some possible selection strategies:

- (a) *Selection of existing heuristics.* For many combinatorial problems a few (and sometimes many) local search heuristics already exist. They usually differ one from another in their, possibly implicit, definition of neighborhood solutions. All we have to do is to make some reasonable selection of

<i>p</i>	<i>Best known</i>	Objective values				% Deviation				Time (seconds)			
		5	10	20	40	5	10	20	40	5	10	20	40
10	101249.45	101249.47	101249.45	101249.45	101249.45	0.00	0.00	0.00	0.00	408.12	242.02	123.33	64.67
20	57857.54	57857.55	57857.54	57857.54	57857.54	0.00	0.00	0.00	0.00	717.78	415.35	230.33	143.25
30	44013.00*	44013.00	44013.05	44013.07	44013.07	0.00	0.00	0.00	0.00	1012.06	606.15	307.27	180.89
40	35002.02*	35002.02	35002.14	35002.14	35002.14	0.00	0.00	0.00	0.01	947.33	570.48	296.94	203.53
50	29089.71	29089.71	29089.77	29089.77	29089.78	0.00	0.00	0.00	0.00	1396.63	808.54	424.06	267.76
60	25160.40*	25160.40	25160.47	25185.84	25197.33	0.00	0.00	0.10	0.15	1278.55	762.35	402.27	260.68
70	22125.46	22125.46	22125.51	22125.50	22125.51	0.00	0.00	0.00	0.00	1508.26	907.68	492.72	304.03
80	19870.45*	19870.45	19884.38	19872.32	19884.67	0.00	0.07	0.01	0.07	1245.73	720.35	462.36	293.94
90	17987.91	17987.91	17987.93	17987.94	17987.93	0.00	0.00	0.00	0.00	1281.16	809.35	439.35	299.57
100	16551.20	16552.50	16552.50	16564.42	16552.49	0.01	0.01	0.08	0.01	3734.35	2057.54	1105.61	605.40
Average	36890.85	36892.27	36894.80	36903.29	0.00	0.01	0.02	0.04	1353.00	789.98	428.42	262.37	

Table 19.1 Parallel VNS with a different number of processors (5, 10, 20 and 40) for the *p*-Median in $n = 1400$ TSP-Lib test problem; maximum number of iteration is set to 60 on SUN Sparc 10; values proved to be optimal are boldfaced, while * denotes new best known solution.

such local searches and apply one after another in the search. This leads to a VND heuristic.

- (b) *Changing parameter(s) of existing methods.* Some local search heuristics are supplied with parameters that have a great influence on the size of the neighborhood. In other words, the cardinality of the complete neighborhood can be a function of parameters whose values are estimated before each run of the code. Instead of fixing them in advance we can systematically change their values (within reasonable limits). In that way, for each parameter value the different solutions in the vicinity of the current one are generated. This simple VNS scheme has been used for solving Traveling salesman problem in [61], where the so-called GENIUS local search [30] has been modified.
- (c) *Use of k -interchange moves.* The easiest and probably the most natural way to generate neighborhoods is by using k -interchange moves. In 0-1 problems this corresponds to the Hamming distance; if the solution of a problem is represented by a set, then this move is defined by the symmetric difference metric, and so on. For example, well-known moves that belong to this class are k -opt, k -reallocation, k -relocation, k -substitution, etc. This approach has been used in solving the Traveling salesman [43], the p -Median [42], the Weighted maximum satisfiability [40], the Bilinear Programming [43], the Multi-source Weber [11] and the Minimum sum-of-squares clustering [25, 45] problems.
- (d) *Breaking up neighborhoods.* A natural neighborhood can be split into several smaller ones, in order to get in a quicker way easy to obtain improvements. For instance, when applying the 2-opt move in the Traveling Salesman Problem, edges considered for introduction in the tour may first be limited to the 20%, then 40% and so on of smallest ones [46].

Ordering neighborhoods. In VND, a natural ordering of the neighborhoods is from smallest to largest, i.e., $|\mathcal{N}'_1(x)| \leq |\mathcal{N}'_2(x)| \leq \dots \leq |\mathcal{N}'_{k_{max}}(x)|$ in order to find quickly the most obvious improvements. In VNS, a natural ordering corresponds to increasing distance between the current point x and points x_k that belong to $\mathcal{N}_k(x)$, $k = 1, \dots, k_{max}$. When using the k -interchange selection rule, the following property holds as well: $|\mathcal{N}_1(x)| \leq |\mathcal{N}_2(x)| \leq \dots \leq |\mathcal{N}_{k_{max}}(x)|$ when k_{max} is much smaller than the size of x .

Selection in the neighborhood. Visiting all solutions from large neighborhoods would produce a non effective algorithm. Hence, some $V_k(x) \subseteq \mathcal{N}_k(x)$ should be generated at random, where $s_k = |V_k(x)|$ are VNS parameters. Setting $s_k = 1$ for all $k = 1, \dots, k_{max}$, the basic VNS is obtained. Alternatively, some selection criteria can be used to define $V_k(x)$ as e.g. considering the five best moves and choosing at random between them. (This strategy called *low-VNS* did not give good results for Weighted Maximum Satisfiability when used alone, but was a very useful component of skewed VNS for that problem [40]).

Search strategies. Two search strategies are usual in local searches: best improvement and first improvement. Note that the decision about search strategy in VNS should be made at two levels: first, when the neighborhood is changed and second, when different solutions in the same neighborhood are visited. Thus, there are four possible options. Note that the basic VNS uses a first improvement strategy (at the first decision level) while VND often uses a best improvement strategy (at the second decision level).

Descent and descent-ascent VNS. If there is no improvement, the decision has to be made whether or not to accept an ascent move when the neighborhood and/or solution in the same neighborhood are changed. Thus, four possibilities again occur: (*descent, descent*); (*descent, descent-ascent*); (*descent-ascent, descent*); and (*descent-ascent, descent-ascent*). For example, the choice of (*descent, descent-ascent*) means that a move will not be made in VNS if the solution obtained by descent-ascent local search is not better than the incumbent. The basic VNS uses the (*descent, descent*) option. Skewed VNS uses the (*descent-ascent, descent*) options.

Forward and backward VNS. When a first improvement search strategy is chosen, the order in which the neighborhoods \mathcal{N}_k will be used can play an important role in the quality of the final solution obtained. An order of the \mathcal{N}_k nondecreasing in k defines *Forward VNS*; it starts with $k := 1$, when there is no improvement in $\mathcal{N}_k(x)$, $k := k + 1$, and if a better solution is found, $k := 1$ again. A nonincreasing order of \mathcal{N}_k in k defines *Backward VNS*: start with $k := k_{\max}$, set $k := k - 1$ in case of unsuccessful search and $k := k_{\max}$ again in the case of success. Forward and Backward VNS are special cases of an extended version: introduce k_{\min} and k_{step} , two parameters that control the change of neighborhood process, i.e., in the previous algorithm instead of $k \leftarrow 1$ set $k \leftarrow k_{\min}$ and instead of $k \leftarrow k + 1$ set $k \leftarrow k + k_{\text{step}}$; if $k_{\text{step}} < 0$, backward VNS is obtained. Clearly, basic VNS uses $k_{\min} = k_{\text{step}} = 1$. Also, one may go from a backward VNS, at the outset when no good solution is yet found, to a forward VNS when a presumably near-optimal local optimum has been detected.

Intensification and diversification. Usual questions in local search heuristics are how to intensify the search in some attractive areas (*Intensification*) and how to find some previously unexplored regions (*Diversification*) [34]. It is easy to see that both these functions can be achieved in a natural way by changing VNS parameters (k_{\max} , k_{\min} , k_{step} , s_k , etc.) and choosing different search strategies and options described above.

In fact, the basic VNS scheme with k -interchange moves has imbedded intensification and diversification strategies: one first explores thoroughly small close neighborhoods until they give no further improvements, then one proceeds to further larger neighborhoods which are more lightly explored.

19.5 RECENT APPLICATIONS

Traveling salesman problem. Given n cities with intercity distances, the traveling salesman problem (TSP) is to find a minimum cost tour x (i.e., a permutation of the cities which minimizes the sum of the n distances between adjacent cities in the tour). It is a classical NP-hard problem. In [12] a new local search heuristic, the so-called k -hyperopt is proposed. Then a VND heuristic uses 2-opt, 2-hyperopt and 3-hyperopt neighborhoods in descent, while a VNS heuristic uses k -hyperopt for shaking and 2-hyperopt for a local search. The new methods are compared with Iterated local search and Multistart 2-opt. Results are reported on standard test instances. It appears that tours obtained with those given by the iterated Lin-Kernighan heuristic [56] are comparable in terms of tour quality, but CPU time is larger.

p -center problem. The p -Center problem consists in locating p facilities and assigning clients to them in order to minimize the maximum distance between a client and the facility to which he (or she) is allocated (i.e., the closest facility). This model is used for example in locating fire stations or ambulances, where the distance from the facilities to their farthest allocated client should be minimum. In [60] basic Variable Neighborhood Search and Tabu Search (e.g., the so called Chain-substitution Tabu Search, [62]) heuristics are presented. Both methods use the 1-interchange (or vertex substitution) neighborhood structure. It is shown how this neighborhood can be used even more efficiently than for solving the p -Median problem. Based on the same computing time, results of a comparison between Multistart 1-interchange, Chain interchange TS (with one and two Tabu lists) and VNS are reported on standard test problem instances from the literature (see Table 19.2). It appears that both TS and VNS outperform the multistart approach.

Multiprocessor scheduling problem with communication delays. The Multiprocessor Scheduling Problem with Communication Delays consists in finding a static schedule of an arbitrary task graph onto a homogeneous multiprocessor system, such that the makespan (i.e., the time when all tasks are finished) is minimum. The task graph contains a precedence relation as well as communication delays (or data transferring time) between tasks if they are executed on different processors. The multiprocessor architecture is assumed to contain identical processors with a given distance matrix (i.e., the number of processors along the shortest path) between each two processors [21]. For solving this NP-hard problem [70], a basic Variable Neighborhood Search heuristic is developed in [22], where a k -swap neighborhood structure is used for shaking and a reduced 1-swap for local search. Results on random graphs with up to 300 tasks and 8 processors are reported in Tables 19.3 and 19.4. Two types of task graphs are generated: (i) with known optimal solution on the 2-dimensional hypercube (i.e. with 4 processors) and given density as proposed in [54] (see Table 19.3); (ii) with given density 0.2, 0.4, 0.5, 0.6 and 0.8 (Table 19.4). The basic VNS is compared with Multistart local search (MLS),

p known	Best	Objective values				% Deviation			Time (seconds)			
		M-I	VNS	TS-1	TS-2	M-I	VNS	TS-1	TS-2	M-I	VNS	TS-1
50	307.48	329.66	317.00	307.48	307.88	7.21	3.10	0.00	0.13	762.15	578.81	981.91
100	215.67	234.10	220.06	219.59	215.67	8.55	2.04	1.82	0.00	994.28	570.60	587.98
150	174.83	195.21	174.83	174.83	177.23	11.66	0.00	0.00	1.37	602.95	52.99	984.27
200	157.00	181.18	157.88	157.00	157.09	15.40	0.56	0.00	0.06	238.56	747.00	661.67
250	137.54	165.47	140.98	137.54	138.92	20.31	2.50	0.00	1.00	640.69	103.72	952.57
300	123.33	158.38	123.33	124.60	123.44	28.42	0.00	1.03	0.09	985.02	786.96	892.71
350	118.02	146.82	118.02	118.83	118.07	24.40	0.00	0.69	0.04	354.46	264.59	783.24
400	107.65	137.57	107.65	114.11	113.22	27.79	0.00	6.00	5.17	723.01	904.88	518.61
450	101.51	135.09	101.51	101.98	102.14	33.08	0.00	0.46	0.62	267.58	674.55	973.90
500	94.37	124.33	94.37	97.20	96.14	31.75	0.00	3.00	1.88	351.76	937.70	732.09
Average						20.86	0.82	1.30	1.04	592.05	562.18	806.89
												806.74

Table 19.2 The p -Center for $n = 3038$ TSP-Lib test problem; maximum time allowed for each instance is set to 1,000 seconds on a SUN Sparc 10.

n	f_{opt}	% Deviation				Time (seconds)					
		CP	LS	VNS	TS	Init	LS	VNS	TS	MLS	PSGA
50	600	51.67	36.67	25.17	36.67	29.00	34.33	0.00	0.23	5.99	0.05
100	800	74.50	65.75	56.88	67.62	66.75	66.88	0.00	5.25	39.36	23.42
150	1000	83.50	60.90	10.50	60.90	77.20	78.60	0.01	22.69	346.49	32.03
200	1200	86.92	83.25	71.67	83.25	78.33	83.50	0.01	27.60	190.21	16.96
250	1400	87.00	86.43	83.50	85.50	84.86	84.29	0.02	19.03	721.65	94.72
300	1600	88.62	88.62	81.62	85.50	81.94	88.19	0.03	34.44	1392.77	1667.78
Average		78.70	70.27	54.89	69.91	69.68	72.63	0.01	18.21	449.41	305.83
											249.39
											121.89

Table 19.3 Multiprocessor scheduling on random test instances with known optimal solution.

Tabu search (TS) [22] and a Genetic algorithm [2] (PSGA). Initial solutions for all methods are obtained by modification of the well-known constructive heuristic *critical path* (CP) [67]. Starting from it a *swap* local search is run (LS).

n	Best (av.)	% Deviation						Time (seconds)			
		CP	LS	VNS	TS	MLS	PSGA	VNS	TS	MLS	PSGA
50	584.67	3.56	0.34	0.03	0.33	0.18	0.48	0.54	0.99	0.34	1.36
100	1207.73	7.38	1.05	0.07	0.57	0.77	2.03	26.40	22.15	7.32	19.36
200	2435.03	10.68	1.67	0.07	0.93	1.20	4.02	246.11	168.87	80.93	187.34
300	3599.83	14.15	2.37	0.09	1.22	1.85	6.63	1235.44	934.11	439.95	1015.84

Table 19.4 Multiprocessor scheduling: average results for each n over 30 random tests.

It appears that for both types of random test instances VNS performs best. However, the reasons why the best solution found by VNS is still more than 50% above the optimal one (in Table 19.3) should be the subject of further investigation.

Route-median problem. Given a set of cities with intercities distances, the Route-Median problem consists in choosing a subset of cities that are included in a cycle and allocating the remaining ones each to the closest chosen city in order to minimize the length of the route with an upper bound on the sum of the distances from the cities not in the route to the cities to which they are assigned. An exact branch and cut solution method is described in [55]. A metaheuristic approach that combines VNS and TS and uses several neighborhood structures is proposed in [66].

Continuous min-max problem. The so-called multi-level Tabu search (MLTS) heuristic has been proposed in [53] for solving a continuous min-max optimization problem of spread spectrum radar polyphase code design [24]:

$$\min_{x \in X} f(x) = \max\{F_1(x), \dots, F_{2m}(x)\} \quad (19.1)$$

$$X = \{(x_1, \dots, x_n) \in R^n | 0 \leq x_j \leq 2\pi, j = 1, \dots, n\},$$

where $m = 2n - 1$ and

$$F_{2i-1}(x) = \sum_{j=i}^n \cos \left(\sum_{k=|2i-j-1|+1}^j x_k \right), \quad i = 1, \dots, n$$

$$F_{2i}(x) = 0.5 \sum_{j=i+1}^n \cos \left(\sum_{k=|2i-j|+1}^j x_k \right), \quad i = 1, \dots, n-1$$

$$F_{m+i}(x) = -F_i(x), \quad i = 1, \dots, n.$$

Improvements obtained with the basic VNS in both solution quality and computing time on the same set of test problems are reported in [63]. Different neighborhoods are derived from the Euclidean distance. A random point is selected from such neighborhoods in the *Shaking* step; then the gradient (feasible direction) local search with a given step size is performed on the functions that are active (functions that have a maximum value in the current point); this step is repeated until the number of active functions in the current point is equal to n (with some tolerance). Some comparative results between MLTS and VNS are given in Table 19.5. It appears that VNS outperforms MLTS on all test instances.

n	known	Objective function values				% Deviation				t_{max} (sec.)	
		Best		Average in 10		Best in 10		Average in 10			
		MLTS	VNS	MLTS	VNS	MLTS	VNS	MLTS	VNS		
2	0.385162	0.385315	0.385175	0.385210	0.385162	0.04	0.00	0.01	0.00	10	
4	0.056166	0.058659	0.056240	0.057307	0.056166	4.44	0.13	2.03	0.00	40	
6	0.456169	0.542915	0.456358	0.457406	0.456169	19.02	0.04	0.27	0.00	120	
8	0.387091	0.430552	0.395468	0.413070	0.387091	11.23	2.16	6.71	0.00	240	
10	0.410478	0.607798	0.473535	0.456855	0.410478	48.07	15.36	11.30	0.00	360	
15	0.485701	0.785917	0.677801	0.748820	0.485701	61.81	39.55	54.17	0.00	660	
20	0.690931	0.933160	0.857995	0.805040	0.690931	35.06	24.18	16.52	0.00	960	
Average						25.67	11.63	13.00	0.00		

Table 19.5 A continuous min-max optimization problem in R^n : Spread spectrum radar polyphase code design; t_{max} is the maximum time allowed for a single run.

Fuzzy clustering problem. The Fuzzy Clustering Problem (FCP) is an important one in pattern recognition. It consists in assigning (allocating) a set of patterns (or entities) to a given number of clusters such that each of them belongs to one or more clusters with different degrees of membership. The fuzzy clustering problem was initially formulated by Dunn [23] as a mathematical programming problem and later generalized by Bezdek [9]. The most popular heuristic for solving FCP is the so-called Fuzzy C-means (F-CM) method [13]. It alternatively finds membership matrices and centroids until there is no more improvement in the objective function value. A new descent local search heuristic called Fuzzy J-means (F-JM) is proposed in [8] where the neighborhood is defined by all possible centroid-to-entity relocations (see also [45, 11]). This ‘integer’ solution is then moved to a continuous one by one alternate step, i.e., by finding centroids with given memberships. Fuzzy VNS rules are applied as well (F-JM is used as local search subroutine). Some preliminary results are given in Table 19.6.

Quadratic assignment problem. The Quadratic Assignment Problem can be described as follows: Given two $n \times n$ matrices A and B , find a permutation π^* minimizing the sum of the $a_{ij} \cdot b_{\pi_i \pi_j}$. A parameter free basic VNS (k_{max} is set to n) is suggested in [69], where two new methods based on Fast Ant

<i>m</i>	Best known	Objective function values				% Deviation			Running time		
		Initial	F-CM	F-JM	VNS	F-CM	F-JM	VNS	F-CM	F-JM	VNS
2	128.90	2190.18	128.90	128.90	128.90	0.00	0.00	0.00	0.02	0.62	3.08
3	60.51	314.92	60.51	60.55	60.51	0.00	0.08	0.00	0.05	0.61	3.07
4	41.61	167.48	49.57	41.62	41.61	19.11	0.01	0.00	0.07	1.23	6.22
5	32.73	115.16	32.93	32.80	32.73	0.61	0.20	0.00	0.16	3.38	17.74
6	24.73	89.18	30.00	24.80	24.73	21.32	0.29	0.00	0.16	3.22	16.10
7	20.39	74.05	22.17	20.48	20.39	8.74	0.47	0.00	0.17	3.26	16.31
8	17.53	62.89	18.04	17.53	17.53	2.93	0.03	0.00	0.33	1.55	7.75
9	15.40	55.24	16.84	15.40	15.40	9.41	0.03	0.00	0.24	3.35	17.58
10	13.47	49.13	14.44	13.48	13.47	7.20	0.13	0.00	0.16	4.54	22.88
Average						7.70	0.14	0.00	0.15	2.42	12.30

Table 19.6 Fuzzy clustering for Iris data ($n = 150$); m is a given number of clusters; the initial solution is obtained using [51].

systems (FANT) and Genetic-Descent hybrid (GDH) are also proposed. All three methods use the same simple descent local search procedure. On a series of *structured* instances from the literature, results of similar quality for all three methods are reported, despite the fact that VNS does not use memory nor parameters in the search. Moreover, when the number of calls of the descent procedure increases, the quality of the solution obtained by VNS improves fastest. The % error was reduced by 2.801%, 1.686% and 1.970% for VNS, FANT and GDH respectively, when the number of calls increased from 10 to 1000.

Oil pipeline design problem. Brimberg *et al* [10] consider a given set of offshore platforms and onshore wells producing known (or estimated) amounts of oil to be connected to a port. Connections may take place directly between platforms, well sites and the port or may go through connection points at given locations. The configuration of the network and sizes of pipes used must be chosen in order to minimize construction costs. This problem is expressed as a mixed integer program, and solved both heuristically by Tabu search and Variable neighborhood search methods and exactly by a branch-and-bound method. Tests are made with data from the South Gabon oil field and randomly-generated problems. Again, VNS gives best solutions and TS is close (see Table 19.7).

Phylogeny problem. The phylogeny (or evolutionary tree) problem, in its simplest form, can be formulated as follows: given A , an $n \times m$ 0-1 matrix; construct a tree with minimum number of ‘evolutionary’ steps, that satisfies the following conditions: (i) each node of the tree corresponds to one n -dimensional 0-1 vector, where vectors given by rows of A should be associated to leaves; (ii) degrees of all nodes that are not leaves are equal to three; (iii) two nodes are connected by an edge if their corresponding vectors differ in only one variable. The trees that satisfies (i)-(iii) and the criterion used are called phylogeny and

Dens.	Best known	% Deviation					Running time				
		MST	RA	LS	TS	VNS	RA	LS	TS	VNS	t_{max}
0.10	19335.4	14.55	11.79	0.50	0.00	0.00	0.60	4.90	5.50	8.20	49.30
0.20	14519.6	11.37	8.62	0.38	0.20	0.00	0.70	9.80	89.50	31.90	98.10
0.30	11450.0	11.59	7.71	0.08	0.00	0.00	0.60	26.90	29.30	266.20	269.00
0.40	9790.1	14.28	11.00	0.00	0.00	0.00	0.40	41.00	41.00	41.00	409.90
0.50	9282.3	13.60	13.00	0.22	0.22	0.00	0.40	56.70	56.70	285.80	567.10
0.60	8489.4	16.57	15.49	0.03	0.00	0.00	0.10	78.40	86.50	78.40	783.80
0.70	8365.2	18.78	14.18	0.35	0.00	0.00	1.00	100.10	275.60	100.10	1000.90
Average		14.39	11.68	0.22	0.06	0.00	0.54	45.40	83.44	115.94	454.01

Table 19.7 Pipeline design on random 150-node problem.

parsimony respectively. The leaves of an evolutionary tree represent groups of species, populations of distinct species, etc. (denoted by taxons as well), while interior nodes represent hypothetical (unknown) ancestors. In Andreatta and Ribeiro [3] a VND method that uses three neighborhoods is developed, and compared with the best among three local searches used alone. It appears that VND is much faster and gives solutions with better quality. In the same paper, the authors compared three metaheuristic approaches to the phylogeny problem: GRASP, VNS and tabu search.

Arc routing problem. In arc routing problems the aim is to determine a least cost traversal of all edges or arcs of a graph, subject to some side constraints. In the capacitated arc routing problem (CARP) edges have a non-negative weight and each edge with a positive weight must be traversed by exactly one of several vehicles starting and ending their trip at a depot, subject to the constraint that total weight of all edges serviced by a vehicle cannot exceed its capacity. In [29] the VNS heuristic developed in [58] for the undirected CARP was compared with a tabu search heuristic [49]. The conclusions are as follows: (i) on small test instances ($7 \leq n \leq 27$), TS and VNS perform similarly (the deviation from optimality is identical and computing times are about 20% smaller for VNS); (ii) on larger test instances VNS is better than TS in terms of solution quality, but also faster (average deviation from optimality and computing times on the 270 instances were 0.71% and 349.81 seconds for TS and 0.54 and 42.50 seconds for VNS).

Degree-constrained minimum spanning tree problem. This problem consists in finding a minimum spanning tree such that the degree of each vertex of the tree is less than or equal to a given integer b_i , for all i . Note that for $b_i = 2$ the problem becomes the Hamiltonian path problem. In Ribeiro and Souza [65] three neighbourhood structures based on the k -edge exchange operator, $k = 1, 2, 3$ (or k -elementary tree transformations) are used in developing local search procedures within a VND heuristic. VND is compared with known methods from the literature, (i.e., a genetic search, problem space search heuristics and simulated annealing) on the same test instances. VND was better and much

faster on almost all test instances. For example, VND succeeded in finding optimal solutions for all problems already solved exactly (in the so-called STR class) in less than one second of processing time, while the best among three other heuristics, i.e., the problem space search heuristic, found suboptimal solutions within up to 300 seconds.

Multicommodity location with balancing requirements. This problem consists in finding locations of depots that collect empty containers in order to satisfy customer requests, while minimizing the total operating costs. For this problem an exact and a few heuristic methods have been proposed in the literature. In Gendron et al. [32] only two neighborhoods are used in developing a VND heuristic, (i.e., add/drop and swap facilities, the same two that they used in their previous paper for solving this problem by Tabu search,[31]). Although these authors used only two neighborhoods and in Step 2b of VND did not use $k \leftarrow 1$, the results obtained are very interesting: VND was much faster than Tabu search (about two orders of magnitude); the gap never exceeded 3.7% and was less than 1% in most cases.

Simple plant location problem. The well-known simple plant location problem (SPLP) is to locate facilities among a given set of cities in order to minimize the sum of set-up costs and distribution cost from the facilities to a given set of users, whose demand must be satisfied. The case where all fixed-costs are equal and the number of facilities is fixed is the p -median problem. In work in progress, VNS heuristics for the p -median are extended to the SPLP. In addition to interchange moves, opening and closing of facilities are considered. First results are given in Table 19.8.

Maximum clique problem. Let $G = (V, E)$ denote a graph with vertex set $V = \{v_1, v_2, \dots, v_n\}$ and edge set $E = \{e_1, e_2, \dots, e_m\}$. A set $C \subseteq V$ of vertices is a *clique* if any two of them are adjacent. A clique C is *maximum* if it has the largest possible number of vertices. Finding the maximum clique is a classical problem in graph theory. A basic VNS heuristic that combines greedy with the simplicial vertex test in its descent step is proposed and tested on standard test problems from the literature in [48]. Despite its simplicity, the proposed heuristic outperforms most of the well-known approximate solution methods (see Table 19.9). One exception is the recent Reactive Local Search (RLS) heuristic of Battiti and Protasi [6] which obtains results of equal quality in a quite different way.

Application of VNS to several other combinatorial problems is currently under study. Hertz and Zufferey [50] solve the **graph colouring problem** by VNS; Potvin and Soriano are currently working on VNS methods for a **network loading problem** from the telecommunications domain (where fiber optic cables must be installed at least cost on a telecommunications network to satisfy service demands); Rodriguez et al. are implementing VNS rules on different variants of **Route-Median Problem**.

<i>m</i>	<i>n</i>	<i>Optimum</i>		<i>% Deviation</i>				<i>Time (seconds)</i>			
		<i>p</i>	<i>Value</i>	<i>LS</i> ₂	<i>LS</i> _{<i>n</i>/2}	<i>LS</i> _{<i>n</i>}	VNS	RVNS	<i>LS</i> ₂	VNS	RVNS
50	16	11	932615.75	0.00	0.00	1.15	0.00	0.00	0.00	0.02	0.04
50	16	9	977799.38	0.00	0.00	3.00	0.00	0.00	0.00	0.02	0.04
50	16	5	1010641.44	0.18	0.06	5.59	0.00	0.00	0.00	0.02	0.04
50	16	4	1034977.00	0.00	1.29	11.10	0.00	0.00	0.00	0.03	0.04
50	25	14	796648.44	0.11	0.11	3.22	0.00	0.11	0.00	0.03	0.04
50	25	11	854704.19	0.00	0.00	8.20	0.00	0.00	0.00	0.05	0.05
50	25	7	893782.12	0.14	0.75	14.66	0.00	0.03	0.00	0.04	0.04
50	25	4	928941.75	0.00	2.42	25.79	0.00	0.00	0.00	0.07	0.04
50	50	15	793439.56	0.11	2.70	18.97	0.00	0.00	0.00	0.15	0.05
50	50	11	851495.31	0.00	5.87	32.51	0.00	0.00	0.00	0.15	0.06
50	50	7	893076.69	0.11	7.31	44.96	0.00	0.11	0.00	0.20	0.05
50	50	4	928941.75	0.00	16.14	65.15	0.00	0.00	0.00	0.17	0.06
1000	100	4	17156454.00	0.00	74.34	240.27	0.00	1.50	1.15	9.22	1.00
1000	100	7	12979072.00	0.00	34.92	165.88	0.00	0.71	0.91	16.19	1.43
1000	100	9	11505594.00	0.03	36.00	120.98	0.00	2.38	0.84	11.04	0.71
Average				0.05	12.13	50.76	0.00	0.32	0.19	2.49	0.25

Table 19.8 Simple plant location problem in OR-Lib test instances [7]; LS_p denotes initial value for p ; VNS and RVNS stop when the number of iterations without improvement reaches 50 and 1000 respectively; k_{max} for VNS was set to the value p obtained from the local search; $k_{max} = 2$ for RVNS.

Pr. No.	Pr. Name	Best Known	GA	CBH	TS		RLS		VNS	
					av.	(best, worst)	av.	(best, worst)	av.	(best, worst)
1.	C125.9	34*			0.00	0.00	0.00		0.00	
2.	C250.9	44*			6.82	1.82	0.00		0.00	
3.	C500.9	57			9.62	1.05 (0.00-3.51)	0.00		0.00	
4.	C1000.9	68			11.76	4.41 (2.94-7.35)	0.00		0.00	
5.	C2000.9	78			18.18	6.67 (5.13-7.69)	0.54 (0.00-1.28)		1.03 (0.00-2.56)	
6.	DSJC500.5	14*			7.14	7.14	7.14		7.14	
7.	DSJC1000.5	15			6.67	5.33 (0.00-6.67)	0.00		0.0	
8.	C2000.5	16			6.25	6.25	0.00		0.00	
9.	C4000.5	18				7.78 (5.56-11.11)	0.00		0.00	
10.	MANN_a27	126*	0.00		3.97	0.79	0.00		0.00	
11.	MANN_a45	345*	0.58	0.58		0.87	0.41 (0.00-0.58)		0.14 (0.00-0.29)	
12.	MANN_a81	1100				0.36	0.18		0.06 (0.00-0.18)	
13.	brock200_2	12*	8.33	0.00		8.33	0.00		5.83 (0.00-8.33)	
14.	brock200_4	17*	5.88	5.88		5.88	0.00		0.58 (0.00-5.88)	
15.	brock400_2	29*	17.24	17.24	15.17 (13.80-17.24)		10.12 (0.00-13.80)		5.51 (0.00-13.80)	
16.	brock400_4	33*	27.27	27.27	26.67 (24.24-27.27)		1.75 (0.00-24.24)		0.00	
17.	brock800_2	21 (24)	20.83	20.83	15.00 (12.50-16.67)			12.50	12.50	
18.	brock800_4	21 (26)	26.92	26.92		23.08		19.23		19.23

Table 19.9 Maximum clique problem. % Deviation from the best known solutions for 18 hard DIMACS challenge instances for the following methods: GA - Genetic algorithm [1]; CBH - Continuous based heuristic [33]; TS - Tabu search [68]; RLS - Reactive local search [6] and VNS.

Several papers on applications of VNS already discussed in previous surveys [43, 44] have appeared. They include VNS heuristics for the Multi-source Weber problem [10] and for the minimum sum-of-squares clustering problem [17, 27].

19.6 COMPUTER-AIDED DISCOVERY IN GRAPH THEORY

Recall that a graph invariant is a variable defined on the family of all graphs (or on an appropriate subfamily) and the value of which does not depend on the numbering of vertices or edges. Numerous problems in graph theory can be viewed, or involve, optimization of invariants on a possibly constrained family of graphs. Therefore, VNS can be used for approximate solution of such problems. This led to the development of the AutoGraphiX (AGX) system and to a series of papers, next reviewed, on its principles and applications.

As explained by Caporossi and Hansen [18], AGX uses the basic schemes of both VNS and VND. The descent is done by first drawing a graph G at random (with a given number of vertices and edges), computing its value for the invariant under study and then examining the effect on this value of applying some elementary transformation to G : removal or addition of an edge, displacement of an edge, detour, i.e., replacement of an edge between two vertices by a path of length 2 joining them through some non-adjacent vertex, and so on. Neighborhoods so defined are ranked from the smallest to the largest. The best move is determined in each of them in turn and, if it improves the value of the invariant studied, the corresponding change is made in G . After a local optimum has been found, neighborhoods are defined using the Hamming distance, i.e., by considering removal or addition of edges to G , and VNS is applied. The only parameter is the maximum number of edges to be removed or added.

AGS can be applied to the following problems: (a) find a graph satisfying given constraints; (b) find a graph with optimal or near optimal values for an invariant subject to constraints; (c) refute a conjecture; (d) suggest a conjecture (or sharpen one); (e) suggest a proof. For instance, three conjectures of Fajtlowicz' system GRAFFITI [26, 27] are refuted in [18], several strengthened and one of these proved, for the particular case of trees, exploiting knowledge of moves needed to find local optima.

Automated conjecture finding. Study of a set of extremal or near-extremal graphs G obtained with AGX taking the numbers n of vertices and m of edges as parameters often suggests conjectures. Moreover, there can be corroborated or refuted by performing various changes interactively. However, it would be preferable (and a major goal of Artificial Intelligence) to have an entirely automated system. Caporossi and Hansen [17] outline three ways to automate conjecture finding: (a) a numerical method, which exploits the mathematics of Principal Component Analysis in order to find a basis of affine relations between graph invariants; (b) a geometric method which consists in finding the convex hull of the points representing extremal or near-extremal graphs in invariants space, with a 'gift-wrapping' algorithm. Hyperplanes of this convex hull correspond to conjectures; (c) a geometric approach, which consists in recognizing the families of graphs to which belong the extremal ones and using known relations between graph invariants in those families to find new relations.

Chemical graph theory. 1. Energy. The π -electronic energy E is defined in chemical graph theory as the sum of absolute values of the eigenvalues of the adjacency matrix [36]. It has been extensively studied by chemists and mathematicians. Caporossi, Cvetkovic, Gutman and Hansen [14] found with AGX several simple, but as yet unknown relations, including the following:

$$E \geq 2\sqrt{m} \quad \text{and} \quad E \geq \frac{4m}{n},$$

which were easily proved.

Chemical graph theory. 2. Randić index. The Randić index [64] and its generalizations are probably the most studied invariants of chemical graph theory. Assign to each edge of a graph G a weight equal to the inverse of the geometric mean of the degrees of its vertices. Then, the Randić index of G is the sum of all such weights. Caporossi, Gutman and Hansen [1999] use AGX to find extremal trees for this index. Maximum values are obtained by paths and minimal ones by two kinds of caterpillars and one of modified caterpillars with an appended 4-star. This last result was proved by an original way of using linear programming. Indeed, variables are associated with the numbers of edges with given degrees of end vertices (in chemical graphs these degrees are bounded by 4). This proof technique has already been used in three other papers of chemical graph theory.

Araujo and de la Peña [5] present several bounds on the Randić index of chemical trees in terms of this index for general trees and of the ramification index, which is the sum for all vertices of the excess of their degree over 2. Hansen and Melot [41] apply AGX to correct some of the results of Araujo and de la Peña, and obtain much strengthened, and best possible, versions of the others.

Chemical graph theory. 3. Polyhenes with maximum HOMO-LUMO gap. Using AGX, Fowler, Hansen and Caporossi [28] study fully conjugated acyclic π systems with maximum HOMO-LUMO gap. In the simplest Hückel model, this property of the eigenvalue spectrum of the adjacency matrix of the corresponding chemical tree corresponds to reactivity. AGX gives for all even n a ‘comb’, i.e., a path on $n/2$ vertices to each vertex of which is appended a single pendant edge. From this, the conjecture that the maximum gap tends to $2\sqrt{2} - 2$ when n goes to infinity can be deduced.

Trees with maximum index. Trees are bipartite and hence bicolorable, say in black and white. Color-constrained trees have given numbers of black and white vertices. Cvetković, Simić, Caporossi and Hansen [20] use AGX to study color-constrained trees with extremal index, or largest eigenvalue of the adjacency matrix. Six conjectures are obtained, five of which are proved.

Applying the numerical method for automated conjecture finding to the extremal trees found gave the conjecture

$$\alpha = \frac{1}{2}(m + n_1 + D - 2r),$$

where α is the stability number, n_1 the number of pendant vertices, D the diameter and r the radius of the tree. It is unlikely that a conjecture with so many invariants would be obtained without computer-aid.

Trees with palindromic Hosoya polynomial. The Hosoya (or distance) polynomial of a graph is defined by

$$H(G) = a_0 + a_1x + \cdots + a_Dx^D,$$

where $a_0 = n$, $a_1 = m$ and $a_k (k = 3, \dots, D)$ is the number of pairs of vertices at distance k . Gutman [35] and, after an extensive computer search, Gutman, Estrada and Ivanciu [37] conjectured that there is no tree with a palindromic Hosoya polynomial, i.e., such that $a_0 = a_D, a_1 = a_{D-1}$ and so on. Caporossi, Dobrynin, Gutman and Hansen [15] used AGX to refute this conjecture. All counter-examples have D even. This case is easier to satisfy than that where D is odd. Indeed, in the former case, there is a free coefficient in the center, and in the latter not. Then, define the distance to the palindrome condition as

$$z_p = \sum_{k=0}^{\lfloor \frac{D}{2} \rfloor} |a_k - a_{D-k}|.$$

AGX gives for all $n = 10$ to $n = 50$ trees with $z_p = \lceil \frac{n}{2} \rceil$ (and higher values, due to border effects for $n \leq 10$). The conjecture

$$z_p \geq \lceil \frac{n}{2} \rceil$$

appears to be hard to prove. (P. H. offers a gallon of Canadian whisky, or of maple syrup, for the first proof or disproof of this conjecture.)

GRAFFITI 105. The transmission of distance of a vertex of a graph G is the sum of distances from that vertex to all others. Conjecture 105 of GRAFFITI [27] is that in all trees the range of degrees does not exceed the range of transmission of distances. Aouchiche, Caporossi and Hansen [4] use AGX to get hints on how to prove this conjecture and study variants of it. Minimizing range of transmission minus range of degrees always gives stars. It is then easy to prove that the conjecture holds with equality for stars only. Moreover, one gets the conjectures: (a) if T is not a star then range of transmission minus range of degree is at least $\lceil \frac{n}{2} \rceil - 2$, and (b) if T has maximum degree $D \leq \lceil \frac{n}{2} \rceil$ then range of transmission minus range of degree is not less than $n - 3D - 1$.

These results are easily proved, and, with a little more work, the extremal graphs characterized.

These examples illustrate the help the VNS-based system AGX can bring to discovery in graph theory. Many further ones are given in the cited papers.

Acknowledgments: Work supported by NSERC grant #GP0105574, FCAR grant #95ER1048 and a grant from CETAI - HEC.

References

- [1] C. Aggarwal, J. Orlin, and R. Tai. Optimized Crossover for the Independent Set Problem. *Operations Research*, 45:226–234, 1997.
- [2] I. Ahmad and M. Dhodhi. Multiprocessor Scheduling in a Genetic Paradigm. *Parallel Computing*, 22:395–406, 1996.
- [3] A. Andreatta and C. Ribeiro. Heuristics for the Phylogeny Problem. To appear in: *Journal of Heuristics*.
- [4] M. Aouchiche, G. Caporossi, and P. Hansen. Variable Neighborhood Search for Extremal Graphs, 8. Variations on Graffiti 105. In preparation, 2001.
- [5] O. Araujo and J.A. de la Peña. Some Bounds on the Connectivity Index of a Chemical Graph. *Journal of Chemical Information and Computer Science*, 38:827–831, 1998.
- [6] R. Battiti and M. Protasi. Reactive Local Search for the Maximum Clique Problem. *Algorithmica*, 29:610–637, 2001.
- [7] J.E. Beasley. A Note on Solving Large p -Median Problems. *European Journal of Operational Research*, 21:270–273, 1985.
- [8] N. Belacel, P. Hansen, and N. Mladenović. Fuzzy J-Means: A New Heuristic for Fuzzy Clustering. *Les Cahiers du GERAD*, G-2001-14, 2001.
- [9] J.C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithm*. Plenum, 1981.
- [10] J. Brimberg, P. Hansen, K.-W. Lih, N. Mladenović, and M. Breton. An Oil Pipeline Design Problem. *Les Cahiers du GERAD* G-2000-73, Montréal, 2000.
- [11] J. Brimberg, P. Hansen, N. Mladenović, and É. Taillard. Improvements and Comparison of Heuristics for Solving the Multisource Weber Problem. *Operations Research*, 48:444–460, 2000.
- [12] E.K. Burke, P. Cowling, and R. Keuthen. New Local and Variable Neighborhood Search Heuristics for a Sequencing Problem in Printed Circuit Board Assembly Applied to the Travelling Salesman Problem. Technical Report, University of Nottingham, 1999.
- [13] R.L. Canon, J.C. Bezdek, and J.V. Dave. Efficient Implementation of the Fuzzy C-Means Clustering Algorithm. *IEEE Transaction on Pattern Recognition and Machine Intelligence*, 8:248–255, 1986.

- [14] G. Caporossi, D. Cvetković, I. Gutman, and P. Hansen. Variable Neighborhood Search for Extremal Graphs. 2. Finding Graphs with Extremal Energy. *Journal of Chemical Information and Computer Science*, 39:984–996, 1999.
- [15] G. Caporossi, A.A. Dobrynin, I. Gutman, and P. Hansen. Trees with Palindromic Hosoya Polynomials. *Graph Theory Notes of New-York*, 37:10–16, 1999.
- [16] G. Caporossi, I. Gutman, and P. Hansen. Variable Neighborhood Search for Extremal Graphs. 4. Chemical Trees with Extremal Connectivity Index. *Computers and Chemistry*, 23:469–477, 1999.
- [17] G. Caporossi and P. Hansen. Finding Relations in Polynomial Time. In: *Proceedings of the XVI International Joint Conference on Artificial Intelligence*, pages 780–785, Stockholm, 1999.
- [18] G. Caporossi and P. Hansen. Variable Neighborhood Search for Extremal Graphs. 1. The AutoGraphiX System. *Discrete Mathematics*, 212:29–44, 2000.
- [19] T. Crainic, M. Gendreau, P. Hansen, N. Hoeb, and N. Mladenović. Parallel Variable Neighborhood Search for the p -Median. In preparation.
- [20] D. Cvetković, S. Simić, G. Caporossi, and P. Hansen. Variable Neighborhood Search for Extremal Graphs. 3. On the Largest Eigenvalue of Color-Constrained Trees. To appear in: *Linear and Multilinear Algebra*, 2001.
- [21] T. Davidović. Scheduling Heuristic for Dense Task Graphs. *Yugoslav Journal of Operations Research*, 10:113–136, 2000.
- [22] T. Davidović, P. Hansen, and N. Mladenović. Variable Neighborhood Search for Multiprocessor Scheduling with Communication Delays. In preparation.
- [23] J.C. Dunn. A Fuzzy Relative of the ISODATA Process and its Use in Detecting Compact Well-Separated Clusters. *Journal of Cybernetics*, 3:32–57, 1974.
- [24] M.L. Dukić and Z.S. Dobrosavljević. A Method of a Spread-Spectrum Radar Polyphase Code Design. *IEEE Journal on Selected Areas in Communication*, 8:743–749, 1990.
- [25] O. du Merle, P. Hansen, B. Jaumard, and N. Mladenović. An Interior Point Algorithm for Minimum Sum-of-Squares Clustering. *SIAM Journal on Scientific Computing*, 21:1485–1505, 2000.
- [26] S. Fajtlowicz. On Conjectures of Graffiti. *Discrete Mathematics*, 72:113–118, 1988.

- [27] S. Fajtlowicz. Written on the wall. Version 09-2000 (regularly updated file accessible via e-mail from clarson@math.uh.edu), 2000.
- [28] P. Fowler, P. Hansen, G. Caporossi, and A. Soncini. Variable Neighborhood Search for Extremal Graphs 7. Polyenes with Maximum HOMO-LUMO Gap. *Les Cahiers du GERAD*, G-2001-15, Montréal, 2001.
- [29] G. Ghiani, A. Hertz, and G. Laporte. Recent Algorithmic Advances for Arc Routing Problems. *Les Cahiers du GERAD*, G-2000-40, Montréal, 2000.
- [30] M. Gendreau, A. Hertz, and G. Laporte. New Insertion and Postoptimization Procedures for the Traveling Salesman Problem. *Operations Research*, 40:1086–1094, 1992.
- [31] B. Gendron, J-Y. Potvin, and P. Soriano. Tabu Search with Exact Neighbor Evaluation for Multi-Commodity Location with Balancing Requirements. *INFOR*, 37:255–270, 1999.
- [32] B. Gendron, J-Y. Potvin, and P. Soriano. Variable Neighborhood Descent for Multi-Commodity Location with Balancing Requirements. In: *Extended abstracts of the III Metaheuristics International Conference*, pages 217–221, Angra do Reis, 1999.
- [33] L.E. Gibbons, D.W. Hearn, and P.M. Pardalos. Continuous Based Heuristic for the Maximum Clique Problem. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 26:103–124, 1996.
- [34] F. Glover and M. Laguna. *Tabu Search*. Kluwer, 1997.
- [35] I. Gutman. A Contribution to the Study of Palindromic Graphs. *Graph Theory Notes of New York*, New York Academy of Science, XXIV:51–56, 1993.
- [36] I. Gutman and O.E. Polansky. *Mathematical Concepts in Organic Chemistry*. Springer, 1986.
- [37] I. Gutman, E. Estrada, and O. Ivanciu. Some Properties of the Wiener Polynomial of Trees. *Graph Theory Notes of New York*, New York Academy of Sciences, XXXVI:7–13, 1999.
- [38] I. Gutman and O. Miljković. Molecules with Smallest Connectivity Index. *Match*, 41:57–70, 2000.
- [39] I. Gutman, O. Miljković, G. Caporossi, and P. Hansen. Alkanes with Small and Large Connectivity Index. *Chemical Physics Letters*, 306:366–372, 1999.
- [40] P. Hansen, B. Jaumard, N. Mladenović, and A. Parreira. Variable Neighborhood Search for Weighted Maximum Satisfiability Problem. *Les Cahiers du GERAD*, G-2000-62, Montréal, 2000.

- [41] P. Hansen and H. Melot. Variable Neighborhood Search for Extremal Graphs. 6. Analyzing Bounds on the Connectivity Index. To appear in: *Les Cahiers du GERAD*, Montréal, 2001.
- [42] P. Hansen and N. Mladenović. Variable Neighborhood Search for the p -Median. *Location Science*, 5:207–226, 1997.
- [43] P. Hansen and N. Mladenović. An Introduction to Variable Neighborhood Search. In: *Metaheuristics, Advances and Trends in Local Search Paradigms for Optimization*, S. Voss, S. Martello, I. Osman, and C. Roucairol, editors, pages 433–458, Kluwer, 1999.
- [44] P. Hansen and N. Mladenović. Variable Neighborhood Search: Principles and Applications. *European Journal of Operational Research*, 130:449–467, 2001.
- [45] P. Hansen and N. Mladenović. J-MEANS: A New Local Search Heuristic for Minimum Sum-of-Squares Clustering. *Pattern Recognition*, 34:405–413, 2001.
- [46] P. Hansen, N. Mladenović, M. Labb  , and M. Gendreau. Variable Neighborhood Search for the Travelling Salesman Problem. In preparation, 2001.
- [47] P. Hansen, N. Mladenović, and D. Perez-Brito. Variable Neighborhood Decomposition Search. *Les Cahiers du GERAD*, G-99-53, Montréal, 1998. To appear in: *Journal of Heuristics*.
- [48] P. Hansen, N. Mladenović, and D. Uroševi  . Variable Neighborhood Search for the Maximum Clique. *Les Cahiers du GERAD*, G-2001-08, 2001.
- [49] A. Hertz, G. Laporte, and M. Mittaz. A Tabu Search Heuristic for the Capacitated Arc Routing Problem. *Operations Research*, 48:129–135, 2000.
- [50] A. Hertz and N. Zufferey. Variable Neighborhood Search for Graph Colouring Problem. In preparation, 2001.
- [51] C.-T. Hsieh, C.-C. Chin, and K.-M. Shen. Generalized Fuzzy Kohonen Clustering Networks. *IEICE Transactions on Fundamentals*, E81-A:2144–2150, 1998.
- [52] D. Johnson and M. Trick (editors). *Cliques, Coloring and Satisfiability: Second DIMACS Implementation Challenge*. American Mathematical Society, 1996.
- [53] V. Kova  evi  , M.   angalovi  , M. A  i  , D. Dra  i  , and L. Ivanovi  . Tabu Search Methodology in Global Optimization. *Computers and Mathematics with Applications*, 37:125–133, 1999.
- [54] Y.-K. Kwok and I. Ahmad. Efficient Scheduling of Arbitrary Task Graphs to Multiprocessors using a Parallel Genetic Algorithm. *J. Parallel and Distributed Computing*, 47:58–77, 1997.

- [55] M. Labb  , G. Laporte, I. Rodriques, and J. Salazar. Median Cycle Problems. SMG Report, Universit   Libre de Bruxelles, 1999.
- [56] S. Lin and B.W. Kernighan. An Effective Heuristic for the Traveling Salesman Problem. *Operations Research*, 21:498–516, 1973.
- [57] F.G. Lopez, B.M. Batista, J.A. Moreno P  rez, and J.M. Moreno Vega. The Parallel Variable Neighborhood Search for the p -Median Problem. Research Report, University of La Laguna, 2000.
- [58] M. Mittaz. *Probl  mes de cheminements optimaux dans des r  seaux avec contraintes associ  es aux arcs*. Ph.D. Thesis, Department of Mathematics, cole Polytechnique F  d  rale de Lausanne, 1999.
- [59] N. Mladenovi  . A Variable Neighborhood Algorithm — A New Metaheuristic for Combinatorial Optimization. Abstracts of papers presented at Optimization Days, Montr  al, page 112, 1995.
- [60] N. Mladenovi  , M. Labb  , and P. Hansen. Solving the p -Center Problem by Tabu Search and Variable Neighborhood Search. SMG Report, R-00-17, Universit   Libre de Bruxelles, 2000.
- [61] N. Mladenovi   and P. Hansen. Variable Neighborhood Search. *Computers and Operations Research*, 24:1097–1100, 1997.
- [62] N. Mladenovi  , J.P. Moreno, and J. Moreno-Vega. A Chain-Interchange Heuristic Method. *Yugoslav Journal of Operations Research*, 6:41–54, 1996.
- [63] N. Mladenovi  , J. Petrovi  , V. Kova  evi  -Vujci  , and M. Čangalovi  . Solving Spread Spectrum Radar Polyphase Code Design Problem by Tabu Search and Variable Neighborhood Search. To appear in: *European Journal of Operational Research*.
- [64] M. Randi  . On Characterization of Molecular Branching. *Journal of the American Chemical Society*, 97:6609–6615, 1975.
- [65] C. Ribeiro and C. Souza. Variable Neighborhood Descent for the Degree-Constrained Minimum Spanning Tree Problem. To appear in: *Discrete Applied Mathematics*.
- [66] I. Rodriguez, M. Moreno-Vega, and J. Moreno-Perez. Heuristics for Routing-Median Problems. SMG Report, Universit   Libre de Bruxelles, 1999.
- [67] G.C. Sih and E.A. Lee. A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures. *IEEE Transactions on Parallel and Distributed Systems*, 4:175–187, 1993.
- [68] P. Soriano and M. Gendreau. Tabu Search Algorithms for The Maximum Clique Problem. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 26:221–244, 1996.

- [69] E. Taillard and L. Gambardella. Adaptive Memories for the Quadratic Assignment Problem. Technical Report I-87-97, IDSIA, Lugano, 1999.
- [70] J.D. Ullman. NP-Complete Scheduling Problems. *Journal of Computers and Systems Science*, 10:384–393, 1975.
- [71] R. Whittaker. A Fast Algorithm for the Greedy Interchange for Large-Scale Clustering and Median Location Problems. *INFOR*, 21:95–108, 1983.

20 ANALYZING THE PERFORMANCE OF LOCAL SEARCH ALGORITHMS USING GENERALIZED HILL CLIMBING ALGORITHMS

Sheldon H. Jacobson

Department of Mechanical and Industrial Engineering
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801-2906, USA
shj@uiuc.edu

Abstract: Generalized hill climbing algorithms provide a well-defined framework to model how local search algorithms address intractable discrete optimization problems. Monte Carlo search, simulated annealing, threshold accepting, and tabu search, among others, can all be formulated as particular generalized hill climbing algorithms. Moreover, generalized hill climbing algorithms provide a structure for classifying and studying a large body of stochastic and deterministic local search algorithms. In particular, the generalized hill climbing algorithm framework can be used to develop a general Markov chain model for the application of local search algorithms to intractable discrete optimization problems. Moreover, the generalized hill climbing algorithm framework has been used to evaluate the finite-time performance of local search algorithms. This analysis is of particular interest to practitioners for whom traditional convergence results are often of limited interest and value. This paper presents a survey of recent results on how the generalized hill climbing algorithm framework has been used to model and gain insight into the performance of local search algorithms. Many of these results provide tools for comparing and evaluating different types of local search algorithms using common performance measures.

20.1 BACKGROUND AND MOTIVATION

Discrete optimization problems are defined by a finite set of solutions together with a real-valued objective function value assigned to each such solution (see Garey and Johnson [22] for a formal definition). The ultimate goal when addressing such problems is to find solutions that globally optimize (minimize) the objective function. To provide a mathematical framework for these problems, various complexity classes have been formulated. Informally, the class *P* (*Polynomial*) contains problems that can be solved in polynomial time in the size of the problem instance. The class *NP* (*Non-deterministic Polynomial*) contains decision problems for which it can be checked in polynomial time (in the size of the problem instance) whether or not a guessed potential solution solves the (decision) problem. The hardest problems in the class *NP* are said to be *NP*-complete (Garey and Johnson [22]). The seminal paper by Cook [10] established *SATISFIABILITY* to be *NP*-complete.

A significant amount of research has been done since Cook's initial result. In particular, there are numerous problems that have been proven to be *NP*-complete (see Garey and Johnson [22] for an extensive list). Tremendous advances in computer technology have redefined the barrier between small and large problem instances that can be solved in a reasonable amount of computing time (see Johnson and Nemhauser [36], Nemhauser [43]). A large number of techniques and algorithms have been developed to obtain approximate or near-optimal solutions to *NP*-complete problems (e.g., see Balas and Vazacopoulos [4] for job shop scheduling problems, Ceria et al. [6] for set covering problems). In addition, special cases of *NP*-complete problems that are solvable in polynomial time have been identified (e.g., see Coullard et al. [11] for certain matching problems). Specialized algorithms for *NP*-complete problems, as well as local search strategies, provide valuable tools for obtaining good (near-optimal) solutions in a reasonable amount of computing time.

Constructing algorithms that find near-optimal solutions for all instances of a particular discrete optimization problem is a formidable task. Some algorithms tend to be *problem-specific*, focusing on exploiting particular problem instance characteristics (e.g., Hochbaum and Pathria [27] for p-center problems, Jacobson et al. [28] and Kumar et al. [37] for flexible assembly system design problems). Other algorithms tend to be *problem-independent* (i.e., generic), offering a means to find reasonable solutions to a wide variety of problems (e.g., the nested partitions method of Shi and Olafsson [46]). The objective of all these algorithms is to find the best possible solution using a limited amount of computing resources (e.g., executing in polynomial time in the size of the problem instance).

Local search algorithms such as *simulated annealing* (Eglese [14], Fleischner [17], Aarts et al. [1]), *genetic algorithms* (Liepins and Hilliard [39], Mühlenbein [42]), and *tabu search* (Glover and Laguna [23], Hansen and Jaumard [24]), are designed to overcome the trappings of local optima. Simulated annealing mimics the annealing process for crystalline solids, where a solid is cooled very slowly from an elevated temperature, with the hope of relaxing towards

a low-energy state. Genetic algorithms emulate the evolutionary behavior of biological systems to create subsequent generations that guide the search towards optimal/near-optimal solutions. Tabu search uses memory to guide the search towards optimal/near-optimal solutions, by dynamically managing a list of forbidden moves. There are numerous variations of tabu search that have been developed, including probabilistic tabu search, aspiration criteria, and candidate list strategies, among others (see Glover and Laguna [23]). Other local search algorithms, such as threshold accepting (Dueck and Scheuer [13]) and the noising method (Charon and Hudry [7]), also attempt to intelligently traverse the solution space in search of optimal/near-optimal solutions.

Developing theoretical properties of local search algorithms can be challenging. Tovey [50] presents an excellent discussion of the issues that must be addressed when applying local search algorithms, including the selection of the neighborhood function. Tovey also develops theoretical results that contrast the worst-case performance versus the average-case performance of such algorithms. Yannakakis [51] presents a detailed exposition on the complexity of finding local optima, including the definition of the class of polynomial-time local search algorithms (PLS) and the complete problems in this class (PLS-complete). This classification framework provides a well-defined structure by which the difficulty of local search problems can be analyzed. Other articles in Aarts and Lenstra [2] provide a summary of recent contributions in the area of local search algorithms and their applications.

The challenges when selecting and applying local search algorithms are to determine under what conditions they converge, if they converge at all, *a priori* which algorithm will be most effective over a finite number of iterations (i.e., a limited computing budget), which algorithm is most appropriate for a particular class of problems, and which initial parameter settings and stopping criteria are most appropriate for particular classes of problems and/or particular local search algorithms.

The literature contains several empirical comparisons of local search algorithms for specific classes or types of problems. However, there are few theoretical results in the literature (see Tovey [50] and Yannakakis [51]) evaluating and comparing the performance of different local search algorithms. Even more significantly, there does not exist a general framework under which different local search algorithms can be classified and studied. The objective of this paper is to introduce such a well-defined structure using the generalized hill climbing (GHC) algorithm framework, and to present a survey of recent results on how this structure has been used to evaluate and compare different local search algorithms.

The paper is organized as follows. Section 20.2 introduces the GHC algorithm framework. It also shows that several well-known local search algorithms can be formulated as particular GHC algorithms. Section 20.3 presents convergence results for GHC algorithms, while Section 20.4 discusses finite-time performance. Section 20.5 discusses measures that can be used to benchmark

the performance of GHC algorithms. Section 20.6 provides concluding comments.

20.2 GENERALIZED HILL CLIMBING ALGORITHMS

To introduce the GHC algorithm framework several definitions are needed. Define the *solution space*, Ω , to be the set of all solutions for a discrete optimization problem. Define an *objective function* $f : \Omega \rightarrow R$ that assigns a real value to each element of the solution space. Define a *neighborhood function* $\eta : \Omega \rightarrow 2^\Omega$, where $\eta(\omega) \subset \Omega$ for all $\omega \in \Omega$. The neighborhood function provides connections between solutions in the solution space, hence allows the solution space to be traversed or searched by moving between solutions. At each iteration of a GHC algorithm, a solution is generated among all neighbors of the current solution by a *neighborhood probability generating function* $g : \Omega \times \Omega \rightarrow [0, 1]$, where $g(\omega, \omega') = 0$ for all $\omega' \notin \eta(\omega)$, and for all $\omega \in \Omega$, $\sum_{\omega' \in \Omega} g(\omega, \omega') = 1$. Neighbors are said to be generated *uniformly* at each iteration of a GHC algorithm execution if, for all $\omega \in \Omega$, with $\omega' \in \eta(\omega)$,

$$\begin{aligned} g(\omega, \omega') &= P\{\omega' \text{ is selected as the neighbor of } \omega \text{ at a given iteration} \\ &\quad \text{of a GHC algorithm}\} \\ &= 1/|\eta(\omega)|. \end{aligned}$$

The ultimate goal when applying GHC algorithms is to identify the globally optimal solution ω^* (i.e., a global minimum ω^* such that $f(\omega^*) \leq f(\omega)$ for all $\omega \in \Omega$).

Generalized hill climbing algorithms (Jacobson et al. [29, 30]) provide a framework for modeling several discrete optimization local search algorithms. GHC algorithms allow inferior solutions to be visited enroute to globally optimal solutions. The GHC algorithm is summarized in pseudo-code form:

```

Set the iteration indices  $i = k = n = 1$ 
Set the outer loop counter bound  $K$  and the inner loop counter
bounds  $N(j)$ ,  $j = 1, 2, \dots, K$ 
Select an initial solution  $\omega(1) \in \Omega$ 
Define the hill climbing (random) variables  $R_j : \Omega \times \Omega \rightarrow R \cup \{-\infty, +\infty\}$ ,
 $j = 1, 2, \dots, K$ 
Repeat while  $k \leq K$ 
  Repeat while  $n \leq N(k)$ 
    Generate a neighboring solution  $\omega' \in \eta(\omega(i))$  with the
    neighborhood probability generating function and compute
     $\delta(\omega(i), \omega') = f(\omega') - f(\omega(i))$ 
    If  $R_k(\omega(i), \omega') \geq \delta(\omega(i), \omega')$ , then  $\omega(i+1) \leftarrow \omega'$ 
    If  $R_k(\omega(i), \omega') < \delta(\omega(i), \omega')$ , then  $\omega(i+1) \leftarrow \omega(i)$ 
     $n \leftarrow n + 1, i \leftarrow i + 1$ 
  Until  $n = N(k)$ 
   $n = 1, k \leftarrow k + 1$ 
Until  $k = K$ 
```

Note that the outer and inner loop counter bounds, K and $N(k)$, $k = 1, 2, \dots, K$, respectively, may all be fixed. Alternatively, K can be fixed and the $N(k)$, $k = 1, 2, \dots, K$, can be 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). Johnson and Jacobson [35] note that the solutions generated by the execution of a GHC algorithm can be modeled as a Markov chain.

A large body of local search algorithms can be formulated as particular GHC algorithms. For example, simulated annealing (Eglese [14]) can be formulated as a GHC algorithm by setting $R_k(\omega(i), \omega') = -t(k)ln(u)$ for all $\omega(i) \in \Omega$, $\omega' \in \eta(\omega(i))$, where $t(k)$ is a temperature parameter and u is distributed $U(0, 1)$. Note that by using the inversion method to generate exponential random variables (Law and Kelton [38]), $R_k(\omega(i), \omega')$ is an exponential random variable with mean $t(k)$. Moreover, for $f(\omega') - f(\omega(i)) > 0$, the acceptance probability can be written as

$$\begin{aligned} P\{R_k(\omega(i), \omega') \geq f(\omega') - f(\omega(i))\} &= P\{-t(k)ln(u) \geq f(\omega') - f(\omega(i))\} \\ &= P\{u \leq e^{-[f(\omega') - f(\omega(i))]/t(k)}\} \\ &= e^{-[f(\omega') - f(\omega(i))]/t(k)}, \end{aligned}$$

which is how simulated annealing is typically described.

Threshold accepting (Dueck and Scheuer [13]) can be formulated as a GHC algorithm by setting $R_k(\omega(i), \omega') = Q_k$ for all $\omega(i) \in \Omega$, $\omega' \in \eta(\omega(i))$, where Q_k is a non-negative real constant. Moreover, for $f(\omega') - f(\omega(i)) > 0$, the acceptance probability can be written as

$$P\{R_k(\omega(i), \omega') \geq f(\omega') - f(\omega(i))\} = P\{Q_k \geq f(\omega') - f(\omega(i))\},$$

which is how threshold accepting is typically described.

A variation of recency based tabu search (the most commonly used short-term memory) can be formulated as a GHC algorithm by setting

$$\begin{aligned} R_k(\omega(i), \omega') &= \begin{cases} +\infty & \text{for } \omega' \notin \Lambda \\ -\infty & \text{for } \omega' \in \Lambda \end{cases} \\ &= I\{\omega' \notin \Lambda\}/[1 - I\{\omega' \notin \Lambda\}] - I\{\omega' \in \Lambda\}/[1 - I\{\omega' \in \Lambda\}] \end{aligned}$$

for all $\omega(i) \in \Omega$, $\omega' \in \eta(\omega(i))$, where Λ is a tabu list of solutions (Glover and Laguna [23]) and $I\{\cdot\}$ is a 0-1 indicator function. Note that this variation of tabu search involves selecting the neighboring solution using the neighborhood probability generating function. If the best non-tabu neighboring solution is selected, then $R_k(\omega(i), \omega') = +\infty$. When recency based tabu search is used, attributes (e.g., properties) that are encountered in solutions recently visited are listed as tabu-active, causing solutions containing these attributes to be forbidden moves. Note that this keeps solutions that have been recently visited from being revisited. Additionally, solutions that share the tabu-active attributes are also not visited. Other variations of tabu search (e.g., probabilistic tabu search, aspiration criteria implementations, strategic oscillations,

candidate list strategies, variable depth strategies) can be formulated within the GHC algorithm framework by either modifying the definition of the tabu list or by replacing the $+\infty$ and $-\infty$ values by appropriate random variables.

Ordinal hill climbing (Sullivan and Jacobson [48]) is a particular GHC algorithm, where the hill climbing component is based on ordinal information. Ordinal hill climbing incorporates the ordinal optimization structure (Ho et al. [26]), which focuses on finding good solutions rather than trying to find the very best solution, into the GHC algorithm framework. The ordinal hill climbing algorithm is summarized in pseudo-code form:

Select a set of M initial solutions $D(0) \subset \Omega$.

Set the iteration number $k = 0$.

Define the hill climbing (random) variables $R_j : \Omega^M \rightarrow \mathbb{Z}^+, j = 1, 2, \dots$

where

- i) $\sum_{m=1}^M P\{R_j = m\} = 1, j = 1, 2, \dots,$
- ii) $P\{R_0 = 1\} = 1,$
- iii) $R_j \rightarrow M$ with probability one as $j \rightarrow +\infty$.

Repeat

Order the solutions in $D(k)$ from smallest to largest objective function values.

Generate R_k .

Keep the best (smallest objective function values) R_k solutions from the set $D(k)$. Call this set E_1 .

Generate one or more neighbors of each of these R_k , to obtain $M - R_k$ new solutions. Call this set E_2 .

Set $D(k + 1) \leftarrow E_1 \cup E_2$.

$k \leftarrow k + 1$.

Until stopping criterion is met.

Sullivan and Jacobson [48] study the relationship between ordinal hill climbing and genetic algorithms. They show that genetic algorithms that incorporate reproduction, crossover, and mutation can be modeled as specific ordinal hill climbing algorithm formulations by defining different rules that create the set E_2 (offspring) from E_1 (parents). In particular, the hill climbing variable R_k of GA determines the number of parents carried over from the population $D(k)$ to the set E_1 . Once the set E_1 has been determined, its members can be mated (e.g., according to a crossover rule) to produce $M - R_k$ offspring. Therefore, the GA concept of mating serves as the neighborhood structure in the OHC algorithm framework. This set of $M - R_k$ offspring, E_2 , together with the set E_1 becomes the next population (i.e., $D(k + 1) = E_1 \cup E_2$).

The GHC algorithm framework includes an unlimited number of local search algorithms for intractable discrete optimization problems. For example, by setting

$$R_k(\omega(i), \omega') = \lceil \ln(1 - u_i) / \ln(1 - p_k) \rceil$$

for all $\omega(i) \in \Omega$, $\omega' \in \eta(\omega(i))$, where p_k is a sampling parameter, and $\{u_i\}$ are independent and identically distributed $U(0, 1)$ random variables, the resulting GHC algorithm is called *geometric accepting*, since R_k is distributed geometric with mean $1/p_k$ (Law and Kelton [38]). By setting

$$R_k(\omega(i), \omega') = -t(k)[\ln(u_1) + \ln(u_2)]$$

for all $\omega(i) \in \Omega$, $\omega' \in \eta(\omega(i))$, where $t(k)$ is a temperature parameter, and u_1 and u_2 are independent and identically distributed $U(0, 1)$ random variables, the resulting GHC algorithm is called *Erlang accepting*, since R_k is distributed Erlang with mean $2t(k)$ (Law and Kelton [38]). By setting

$$R_k(\omega(i), \omega') = e(k) + [-2(\ln(u_1))]^{1/2} \sin(2\pi u_1)v(k)$$

for all $\omega(i) \in \Omega$, $\omega' \in \eta(\omega(i))$, where u_1 and u_2 are independent and identically distributed $U(0, 1)$ random variables, and $e(k)$ and $v(k)$ are temperature parameters, the resulting GHC algorithm is called *normal accepting*, since R_k is distributed normal with mean $e(k)$ and variance $[v(k)]^2$ (Law and Kelton [38]). In fact, using the GHC algorithm framework and defining R_k to be any random variable results in a unique local search algorithm. The two criteria to consider when selecting such random variables are assessing how difficult it is to generate observations from this random variable and assessing the effectiveness of such random variables when they are implemented (i.e., the effectiveness of the resulting GHC algorithm in reaching globally optimal solutions).

20.3 CONVERGENCE RESULTS

Asymptotic convergence conditions for GHC algorithms can be used to obtain a theoretical understanding of the performance of GHC algorithms. There is a large body of literature on the convergence properties and convergence conditions for simulated annealing (e.g., Hajek [25], Mitra et al. [41], Schuur [45]). To date, there are a limited number of articles that report convergence results for tabu search (see the discussion of probabilistic tabu search in Faigle and Kern [15]). The convergence theory for simulated annealing depends heavily on its exponential hill climbing random variable, and the Markov chain structure inherent in the algorithm's design (i.e., reversibility or detailed balance). Johnson and Jacobson [34] exploit this structure by providing sufficient convergence conditions for a class of GHC algorithms. In particular, if the acceptance probability $P\{R_k(\omega(i), \omega') \geq f(\omega') - f(\omega(i))\}$ can be written as

$$\min[1, P\{\tilde{R}_k(\omega^*, \omega') \geq f(\omega') - f(\omega^*)\}/P\{\tilde{R}_k(\omega^*, \omega(i)) \geq f(\omega(i)) - f(\omega^*)\}], \quad (20.1)$$

where ω^* is a globally optimal solution, and \tilde{R}_k are non-negative (random) variables, then the resulting GHC algorithm converges in probability to the set of globally optimal solutions. Johnson and Jacobson [34] identify several GHC algorithms (including simulated annealing) that satisfy (20.1). Moreover, they give a general expression that defines the acceptance probability in terms of the

distribution functions for $\tilde{R}_k(\omega^*, \omega(i))$ and $\tilde{R}_k(\omega^*, \omega')$. The major limitation of their approach is that the globally optimal objective function value must be known *a priori*. There are several applications where this is the case (e.g., see Yücesan and Jacobson [52], Jacobson and Yücesan [31]). For practical implementations, knowing a bound for the globally optimal objective function value is sufficient to effectively implement these convergent GHC algorithms.

Johnson and Jacobson [35] circumvent the restrictions imposed by Markov chain theory to develop general *sufficient* convergence conditions for GHC algorithms. Their conditions are based on a path argument that quantifies the algorithm's ability to traverse the solution space specifically between global and local optima. To describe these results, note that the objective function, f , together with the neighborhood function, η , allow the solution space, Ω , to be decomposed into three mutually exclusive and exhaustive sets:

- i) a set of γ *globally optimal solutions*, G ,
- ii) a set of λ *locally (but not globally) optimal solutions*, $L \equiv L(\eta)$, and
- iii) a set of φ *hill solutions*, H ,

with $\Omega = G \cup L \cup H$, $G \cap L = \emptyset$, $G \cap H = \emptyset$, and $L \cap H = \emptyset$ (Johnson and Jacobson [35]).

For a discrete optimization problem (with neighborhood function η), the elements of L and G always exist, but are generally unknown (if they were known, the problem would be solved). The iterations of a GHC algorithm can be classified and grouped based on the type of solutions visited. A *micro iteration* moves the current solution either to an immediate neighbor or back to itself. A *macro iteration* is a set of micro iterations that moves the algorithm from any element of $L \cup G$ to any element of $L \cup G$ (including itself), passing only through elements of H . The micro iterations between macro iterations define the *paths* between elements of $L \cup G$.

Using this structure, Johnson and Jacobson [35] obtain the following sufficient convergence condition.

Theorem 1 (Johnson and Jacobson [35]): *For an instance of a discrete optimization problem, with solution space Ω , objective function f , and neighborhood function η , and a particular GHC algorithm (with hill climbing random variable R_k), then under mild restrictions on the neighborhood function and the neighborhood probability generating function (i.e., the solution space is connected, all solutions are reachable from each other with positive probability for all k , the neighborhood probability generating function is strictly positive for any neighbor for all k , and the limit of the long run stationary probabilities exist), if*

- i) $\sum_{k=1}^{+\infty} P_k(\text{Min_Path}) = +\infty$,
- ii) $\sum_{k=1}^{+\infty} P_k(\text{Max_Path}) < +\infty$, and

$$\text{iii) } \sum_{k=1}^{+\infty} P_k(\text{Max_Prod}) < +\infty,$$

where $P_k(\text{Min_Path}) \equiv \min\{P_k(j \rightarrow i) | j \in L, i \in L \cup G, P_k(j \rightarrow i) > 0\}$, $P_k(\text{Max_Path}) \equiv \max\{P_k(i \rightarrow j) | i \in G, j \in L\}$, $P_k(\text{Max_Prod}) \equiv \max\{\delta_j(k)P_k(j \rightarrow q) | j, q \in L, q \neq j\}$, $P_k(j \rightarrow i)$ is the probability of moving from solution $j \in L \cup G$ to solution $i \in L \cup G$ along a path, $k = 1, 2, \dots$, and $\delta_j(k)$ is the long run stationary probability of being at solution j at macro iteration k , then the GHC algorithms converges in probability to elements in G (i.e., $\sum_{j \in G} \delta_j(k) \rightarrow 1$ as $k \rightarrow +\infty$) provided $R_k(\omega(i), \omega') \rightarrow_P 0$ as $k \rightarrow +\infty$ for all $\omega(i) \in \Omega$, $\omega' \in \eta(\omega(i))$.

Johnson and Jacobson [35] relate the three conditions in Theorem 1 to properties of R_k (such as the rate at which R_k approaches zero) to provide specific convergent GHC algorithm formulations. One limitation of this result is that the sufficient conditions are based solely on the rates (how quickly or how slowly) three different path probabilities approach zero as the number of iterations approaches infinity, without considering interactions between the paths. For example, these conditions may be violated, yet the *relative rates* at which these probabilities approach zero may be sufficient to guarantee convergence.

To exploit this observation, Sullivan and Jacobson [49] develop necessary and sufficient convergence conditions based on ratios of probabilities. To obtain their result, at each macro iteration k fixed, they define a homogeneous discrete time Markov chain, with *micro iteration transition matrix*

$$P_m^k = \begin{bmatrix} P_m^k(G, G) & P_m^k(G, L) & P_m^k(G, H) \\ P_m^k(L, G) & P_m^k(L, L) & P_m^k(L, H) \\ P_m^k(H, G) & P_m^k(H, L) & P_m^k(H, H) \end{bmatrix}$$

at macro iteration k , where $P_m^k(U, V)$, $U, V \in G \cup L \cup H$ are $|U| \times |V|$ matrices representing the micro iteration transition probabilities from the elements in set U to the elements in set V . Note that if the micro iteration transition Markov chain is aperiodic and irreducible, then at macro iteration k , the macro iterations define an inhomogeneous Markov chain, with *macro iteration transition matrix*

$$P_M^k = \begin{bmatrix} P_m^k(G, H)(I - P_m^k(H, H))^{-1}P_m^k(H, G) + P_m^k(G, G) \\ P_m^k(L, H)(I - P_m^k(H, H))^{-1}P_m^k(H, G) + P_m^k(L, G) \\ P_m^k(G, H)(I - P_m^k(G, H))^{-1}P_m^k(H, L) + P_m^k(G, L) \\ P_m^k(L, H)(I - P_m^k(H, H))^{-1}P_m^k(H, L) + P_m^k(L, L) \end{bmatrix}.$$

To obtain necessary and sufficient convergence conditions for a GHC algorithm to converge, define Π_ω^k to be the long run stationary probability of a GHC algorithm being at solution $\omega \in L \cup G$ at macro iteration k . Lemma 1 provides upper and lower bounds on the sum of the long run stationary probabilities of being at a solution in L at macro iteration k .

Lemma 1 (*Sullivan and Jacobson [49]*): For a GHC algorithm execution at macro iteration k (fixed), with macro iteration transition matrix P_M^k ,

$$\begin{aligned} \min_{\omega \in G} \sum_{\sigma \in L} P_M^k(\omega, \sigma) / [\min_{\omega \in G} \sum_{\sigma \in L} P_M^k(\omega, \sigma) + \max_{\sigma \in L} \sum_{\omega \in G} P_M^k(\sigma, \omega)] &\leq \sum_{\omega \in L} \Pi_\omega^k \\ &\leq \max_{\omega \in G} \sum_{\sigma \in L} P_M^k(\omega, \sigma) / [\max_{\omega \in G} \sum_{\sigma \in L} P_M^k(\omega, \sigma) + \min_{\sigma \in L} \sum_{\omega \in G} P_M^k(\sigma, \omega)]. \end{aligned}$$

These bounds are used to obtain necessary and sufficient convergence conditions for a GHC algorithm to converge in probability to G .

Theorem 2 (*Sullivan and Jacobson [49]*): For a GHC algorithm execution at macro iteration k (fixed), with macro iteration transition matrix P_M^k ,

a) *(Necessary Condition)*

If $\Pi_\omega^k \rightarrow 0$ as $k \rightarrow +\infty$, for all $\omega \in L$,
then $[\min_{\omega \in G} \sum_{\sigma \in L} P_M^k(\omega, \sigma)] / [\max_{\sigma \in L} \sum_{\omega \in G} P_M^k(\sigma, \omega)] \rightarrow 0$ as $k \rightarrow +\infty$,

b) *(Sufficient Condition)*

If $[\max_{\omega \in G} \sum_{\sigma \in L} P_M^k(\omega, \sigma)] / [\max_{\omega \in G} \sum_{\sigma \in L} P_M^k(\omega, \sigma) + \min_{\sigma \in L} \sum_{\omega \in G} P_M^k(\sigma, \omega)] \rightarrow 0$ as
 $k \rightarrow +\infty$,
then $\Pi_\omega^k \rightarrow 0$ as $k \rightarrow +\infty$, for all $\omega \in L$.

Theorem 2 uses the macro iteration structure and the macro iteration transition matrix to obtain conditions that can be used to determine when a GHC algorithm does or does not converge.

20.4 FINITE-TIME PERFORMANCE RESULTS

Convergence results provide theoretical support for applying GHC algorithms. Practitioners, however, are interested in an algorithm's *finite-time* performance. Finite-time performance measures for GHC algorithms provide useful information that can be used to help determine how to select the hill climbing (random) variables $R_k(\omega(i), \omega')$ for classes of discrete optimization problems when a limited computing budget is available.

There have been relatively few results on finite-time performance measures for simulated annealing. Mitra et al. [41] present objective function bounds over a finite horizon. Chiang and Chow [8] and Mazza [40] investigate the statistical properties of the first visit time to a global optima which provides insight into the time-asymptotic properties of the algorithm as the outer loop counter, $k \rightarrow +\infty$. Catoni [5] investigates optimizing a finite-horizon cooling schedule to maximize the number of visits to a global optimum after a finite number of iterations. Desai [12] focuses on finite-time performance by incorporating size-asymptotic information supplied by certain eigenvalues associated with the transition matrix. Desai [12] 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 related to the number of steps that the algorithm has taken.

Srichander [47] examines the finite-time performance of simulated annealing using spectral decomposition of matrices. Srichander [47] shows that for the final solution of the simulated annealing algorithm to converge to the global minimum with probability one, an annealing schedule on the temperature is not necessary. Srichander [47] also shows that annealing schedules on the temperature produce an inefficient algorithm in the sense that the number of function evaluations required to obtain a global minimum is very large. A modified simulated annealing algorithm is presented with an iterative schedule on the size of the neighborhood sets that leads to a more efficient algorithm. The performance of this algorithm on a real-world example is also reported.

Fleischer and Jacobson [18] present an entropy measure of the underlying Markov chains for simulated annealing, and use this measure to compare various finite-time simulated annealing implementations. They show that the higher this entropy measure, the better the finite-time performance for simulated annealing, as measured by the likelihood of terminating in a globally optimal solution.

To describe new performance measures that capture the finite-time performance of GHC algorithms, let \mathbf{A} denote a GHC algorithm applied to an instance of a discrete optimization problem, where K represents the *total* number of macro iterations obtained during the algorithm's execution. Assume that for Algorithm \mathbf{A} , $R_k(\omega(i), \omega) \geq 0$, $\omega(i) \in \Omega, \omega \in \eta(\omega(i))$, for all macro iterations $k = 1, 2, \dots, K$. At each macro iteration k , define the events

$$B_{\mathbf{A}}(k, G) = B(k, G) \equiv \{\mathbf{A} \text{ does not visit any element of } G \text{ after } k \text{ macro iterations}\} \quad (20.2)$$

and

$$B_{\mathbf{A}}(G) = B(G) \equiv \{\mathbf{A} \text{ does not visit any element of } G\}. \quad (20.3)$$

These two events are distinct in that $B(k, G)$, $k = 1, 2, \dots, K$, are for Algorithm \mathbf{A} executed over a finite number of macro iterations k , while $B(G)$ has no such limitation. The complementary events, $B^c(k, G)$ (termed the *finite global visit probability*) and $B^c(G)$ (termed the *global visit probability*), are defined as

$$B^c(k, G) \equiv \{\mathbf{A} \text{ visits at least one element of } G \text{ after } k \text{ iterations}\} \quad (20.4)$$

and

$$B^c(G) \equiv \{\mathbf{A} \text{ visits at least one element of } G\}. \quad (20.5)$$

The definition of $B(k, G)$ in (20.2) implies that $B(k, G) \supseteq B(k+1, G)$, for all macro iterations $k = 1, 2, \dots, K$, hence $\{B(k, G)\}$ is a telescoping, non-increasing sequence of events in k . Therefore, since the probability function is a continuous set function, then by the Monotone Convergence Theorem (Ross [44], p.44),

$$P\{B(K, G)\} \rightarrow P\{B(G)\} \text{ as } K \rightarrow +\infty,$$

where

$$B(G) = \bigcap_{k=1}^{+\infty} B(k, G).$$

After K macro iterations, Algorithm **A** yields K solutions, $\{\omega_1, \omega_2, \dots, \omega_K\} \subseteq L \cup G$. Define f^K to be the minimum objective function value among these K solutions and ω^K to be the associated solution (i.e., $f^K = f(\omega^K)$ with $\omega^K = \operatorname{argmin}\{f(\omega_k), k = 1, 2, \dots, K\}$). In practice, the best solution to date (i.e., ω^K) is reported. The key issue is whether $\omega^K \in G$. If $\omega^K \in G$, then Algorithm **A** would be terminated at (no later than) macro iteration K . On the other hand, if $\omega^K \notin G$, it would be desirable to determine whether Algorithm **A** will at some future macro iteration find any solution in G . In this case, it would also be desirable to determine the number of additional macro iterations required to visit such a solution. The probability that ω^K is an element of G , $P\{\omega^K \in G\}$, is given by $P\{B^c(K, G)\}$. If this probability is sufficiently close to one, then Algorithm **A** may be terminated. Therefore $P\{B^c(K, G)\}$ provides a quality measure for the solutions obtained after K macro iterations.

To establish the relationship between the convergence of a GHC algorithm and $P\{B(G)\}$, the following definition of convergence in probability is formally stated.

Definition 1 *GHC Algorithm **A** converges in probability to G if $P\{C(K, G)\} \rightarrow 1$ as $K \rightarrow +\infty$, where $C_A(K, G) = C(K, G) \equiv \{\mathbf{A} \text{ is at an element of } G \text{ at macro iteration } K\}$.*

Therefore, for a given (fixed) initial solution $\omega(0)$, if Algorithm **A** converges in probability to G (as $K \rightarrow +\infty$), then $P\{B^c(G)\} = 1$. Equivalently, if $P\{B^c(G)\} < 1$, then Algorithm **A** does not converge in probability to G (i.e., for all $\varepsilon > 0$, there exists a macro iteration K_0 such that $P\{C(K, G)\} \leq 1 - \varepsilon$ for all $K \geq K_0$).

In light of these observations, the *false negative problem* asks whether a GHC Algorithm **A** will eventually visit an element of G , given that the algorithm, after executing a finite number of macro iterations, has yet to visit an element of G . This question can be quantified by considering the *false negative probability* at macro iteration K , $P\{B^c(G)|B(K, G)\}$.

The false negative probability at macro iteration K provides a measure for the effectiveness of a GHC Algorithm **A**, namely the ability of Algorithm **A** to visit an element of G beyond macro iteration K . In particular, if $P\{B^c(G)\}$ is small, then one can use the false negative probability to assess whether a GHC algorithm will eventually visit an element of G ; if the false negative probability at macro iteration K is sufficiently close to zero, then the algorithm may be terminated.

The false negative probability can be used to derive necessary convergence conditions for GHC algorithms. The false negative probability is also used to measure the effectiveness of non-convergent (in probability) GHC algorithms. Without loss of generality, assume that $P\{B(0, G)\} = 1$ (i.e., all GHC algorithm

runs are initialized at an element of L , hence $\omega(0) \in L$). Furthermore, unless otherwise stated, assume that $P\{B^c(k, G)\} < 1$ for all macro iterations $k = 1, 2, \dots, K$.

For macro iteration k , define the event

$$R_A(k, G) = R(k, G) \equiv \{\textbf{A} \text{ visits any element of } G \text{ after } k \text{ macro iterations, given that } \textbf{A} \text{ has not visited any element of } G \text{ after } k - 1 \text{ macro iterations}\}, \quad (20.6)$$

where

$$r(k, G) \equiv P\{R(k, G)\} = P\{B^c(k, G)|B(k - 1, G)\} = P\{C(k, G)|B(k - 1, G)\}. \quad (20.7)$$

This probability can be used to quantify the false negative probability. Lemma 2 expresses the relationship between (20.2) and (20.7).

Lemma 2 (*Jacobson and Yücesan [32]*): *For GHC Algorithm **A**,*

$$(i) \quad P\{B(K, G)\} = \prod_{k=1}^K [1 - r(k, G)] \text{ for all macro iterations } K.$$

$$(ii) \quad P\{B(G)\} = \prod_{k=1}^{+\infty} [1 - r(k, G)].$$

Theorem 3 summarizes the relationship between the global visit probability, the false negative probabilities, $r(K, G)$, and convergence in probability to G .

Theorem 3 (*Jacobson and Yücesan [32]*): *For a GHC Algorithm **A**, consider the expressions*

$$(D1) \quad P\{C(K, G)\} \rightarrow 1 \text{ as } K \rightarrow +\infty \text{ (converges in probability to } G\text{),}$$

$$(D2) \quad P\{B^c(G)|B(K, G)\} = 1 \text{ for all macro iterations } K \text{ (visits } G \text{ with probability one),}$$

$$(D3) \quad P\{B^c(G)\} = 1 \text{ (visits } G \text{ with probability one),}$$

$$(D4) \quad \sum_{k=K+1}^{+\infty} r(k, G) = +\infty \text{ for all macro iterations } K.$$

Then, (D1) \Rightarrow (D2) \Leftrightarrow (D3) \Leftrightarrow (D4).

Theorem 3 provides three necessary conditions for the convergence of a GHC algorithm. The only restriction on how the GHC algorithm traverses the solution space is that $P\{B(K, G)\} > 0$ for all macro iterations $K = 1, 2, \dots$. This restriction means that the GHC algorithm is never guaranteed to visit any element of G for K finite (i.e., $P\{B^c(K, G)\} < 1$ for all macro iterations $K = 1, 2, \dots$).

Theorem 4 shows that for a GHC algorithm with $\sum_{k=1}^{+\infty} r(k, G) < +\infty$, then with probability one, only a finite number of the events $R(k, G)$, $k = 1, 2, \dots$, occur simultaneously, or equivalently, with probability zero, the $R(k, G)$, $k = 1, 2, \dots$, occur infinitely often.

Theorem 4 (Jacobson and Yücesan [32]): *Suppose that for a GHC Algorithm A, $\sum_{k=1}^{+\infty} r(k, G) < +\infty$ (hence, the algorithm does not converge in probability to G). Then $R(k, G)$, $k = 1, 2, \dots$, occur finitely often with probability one.*

Theorem 4 shows that if $\sum_{k=1}^{+\infty} r(k, G) < +\infty$, hence the GHC algorithm does not converge in probability to G , then for all $\varepsilon > 0$, there exists $K(\varepsilon) \in \mathbb{Z}^+$ such that $P\{\bigcup_{k=K(\varepsilon)}^{+\infty} R(k, G)\} \leq \sum_{k=K(\varepsilon)}^{+\infty} r(k, G) \leq \varepsilon$. This means that for some non-convergent (in probability) GHC algorithms, the probability that it will visit an element of G for the first time (at or beyond macro iteration K) can be made arbitrarily small if the macro iteration K is set sufficiently large. This result can be used to define a stopping condition for a GHC algorithm run. For example, if an improved solution has not been observed for some prespecified number of macro iterations, it may be feasible to terminate the GHC algorithm run.

These results can be used to assess the performance of GHC algorithms. To illustrate, the performance of Monte Carlo search, random-restart local search, and threshold accepting, are evaluated. For Monte Carlo search, Theorem 3 implies that the false negative probability is one (for all macro iterations) for Monte Carlo search. To see this, Monte Carlo search can be described as a GHC algorithm by setting $\eta(\omega) = \Omega$ for all $\omega \in \Omega$, and $R_k = +\infty$ for all macro iterations $k = 1, 2, \dots$. If $p(G) \equiv \gamma/(\gamma + \lambda)$, then $r(k, G) = p(G)$. Therefore, $P\{B(k, G)\} = [1 - p(G)]^k$ and the finite false negative probability approaches one as j approaches infinity. Moreover, from Theorem 3, $\sum_{j=k+1}^{+\infty} r(j, G) = \sum_{j=k+1}^{+\infty} p(G) = +\infty$ for all macro iterations k . This means that Monte Carlo search visits G with probability one as k approaches infinity. However, $P\{C(k, G)\} = p(G)$ for all macro iterations k , hence, Monte Carlo search does not converge in probability to G . More specifically, from Theorem 3, (D2), (D3), and (D4) all hold, but (D1) is not satisfied.

Random-restart local search combines Monte Carlo search and local search, by randomly selecting a new initial solution every time a local search algorithm terminates at a local optimum. The analysis for Monte Carlo search also shows that the false negative probability is one (for all macro iterations) for random-restart local search, by redefining $p(G)$ to be the probability that a randomly generated initial element of Ω will terminate at an element of G . Moreover, random-restart local search will not converge in probability to G (i.e., from Theorem 3, (D2), (D3), and (D4) all hold, but (D1) is not satisfied).

Threshold accepting is a particular GHC algorithm with $R_k(\omega(i), \omega') = Q_k, \omega(i) \in \Omega, \omega' \in \eta(\omega(i))$, for macro iteration k , where Q_k approaches zero as k approaches infinity. Therefore, there exists an $\varepsilon > 0$ sufficiently small and a

macro iteration k_0 finite such that $|Q_k| < \varepsilon$ and $P\{R_k(\omega(i), \omega') \geq \delta(\omega(i), \omega')\} = 0$ for all $\omega(i) \in L$, $\omega' \in \eta(\omega(i))$, and all $k \geq k_0$, hence (D4) in Theorem 3 does not hold. This implies that this particular form of threshold accepting does not converge in probability to G . However, if Q_k is set such that it does not approach zero, hence $r(k, G) \geq \delta$ for some $\delta > 0$ and for all macro iterations k , then (D4) in Theorem 3 may hold and the probability of a false negative is one at all macro iterations k . However, setting Q_k in this way may not be feasible in practice, since it requires full knowledge of the solution space (with respect to the depth of all the local and global optima; see Hajek [25]). Note that Althofer and Koschnick [3] develop a convergence result for threshold accepting by showing that simulated annealing belongs to the convex hull of threshold accepting. Although they do not show that threshold accepting asymptotically converges to a globally optimal solution, they do show that there always exists a set of Q_k values for threshold accepting that ensures convergence to within an ε -neighborhood of a globally optimal solution.

20.5 BENCHMARK PERFORMANCE RESULTS FOR GENERALIZED HILL CLIMBING ALGORITHMS

The false negative probability measure can be used to benchmark the performance of GHC algorithms. This analysis results in new necessary and sufficient convergence conditions, as well as provides a vehicle for developing new performance measures to establish benchmark performance criteria. These results are reported in Jacobson and Yücesan [33]. To describe these results, several additional definitions are needed.

Proposition 1 establishes the well-known relationship between convergence in probability to G , almost sure convergence to G , and visits G infinitely often.

Proposition 1 (*Jacobson and Yücesan [33]*): *Let \mathbf{A} be a GHC algorithm.*

- i) *If GHC Algorithm \mathbf{A} converges almost surely to an element of G (as the number of macro iterations approaches infinity), then Algorithm \mathbf{A} converges in probability to an element of G .*
- ii) *If GHC Algorithm \mathbf{A} converges in probability to an element of G , then Algorithm \mathbf{A} visits an element of G infinitely often.*

Proposition 2 provides a condition on the hill climbing random variables such that i) and ii) in Proposition 1 become the same.

Proposition 2 (*Jacobson and Yücesan [33]*): *Let \mathbf{A} be a GHC algorithm. Suppose that for the hill climbing random variables for \mathbf{A} , there exists a macro iteration K_0 such that $R_k(\omega, \omega') = 0$ with probability one for all $\omega \in \Omega$, $\omega' \in \eta(\omega)$, $k \geq K_0$. Then the GHC Algorithm \mathbf{A} converges almost surely to an element of G (as the number of macro iterations approaches infinity) if and only if it visits an element of G infinitely often.*

From Proposition 2, if for GHC Algorithm **A** there exists a macro iteration K_0 such that $R_k(\omega, \omega') = 0$ with probability one for all $\omega \in \Omega$, $\omega' \in \eta(\omega(i))$, $k \geq K_0$, then convergence almost surely to G and convergence in probability to G are identical. Therefore, if a GHC algorithm is designed to become deterministic local search after some finite number of iterations, then convergence in probability to G reduces to almost sure convergence to G .

The finite global visit probability, $P\{B^c(k, G)\}$, can be used to obtain necessary and sufficient convergence conditions for GHC algorithms. Recall that $P\{B(0, G)\} = 1$ (i.e., all GHC algorithm runs are initialized at an element of L , hence $\omega(0) \in L$).

The one-step macro iteration transition probability, $r(k, G)$, defined in (20.7) is used to obtain the necessary and sufficient convergence conditions. Theorem 5 provides necessary and sufficient conditions for a GHC algorithm to converge in probability to G .

Theorem 5 (*Jacobson and Yücesan [33]*): *Let **A** be a GHC algorithm. Then $P\{C(K, G)\} \rightarrow 1$ as $K \rightarrow +\infty$ (i.e., converges in probability to G) if and only if*

$$i) \sum_{K=1}^{+\infty} r(K, G) = +\infty \text{ and}$$

$$ii) P\{C^c(K, G)|B^c(K-1, G)\} \rightarrow 0 \text{ as } K \rightarrow +\infty.$$

Condition i) in Theorem 5 requires that $r(K, G)$ not converge to zero too quickly, as K approaches plus infinity. This means that the conditional probability that GHC Algorithm **A** visits an element of G for the first time at macro iteration K approaches zero sufficiently slowly such that the infinite summation diverges. Condition ii) requires that the conditional probability that GHC Algorithm **A** visits an element of G for either the second, or the third, or up to the K^{th} time, including the K^{th} macro iteration, approaches one as the number of macro iterations K approaches plus infinity.

If a GHC algorithm does not converge in probability to G , then conditions i) and ii) in Theorem 5 suggest three possible cases:

Case 1) $P\{C^c(K, G)|B^c(K-1, G)\} \rightarrow 0$ as $K \rightarrow +\infty$, but $\sum_{K=1}^{+\infty} r(K, G) < +\infty$.

In this case $\sum_{K=1}^{+\infty} r(K, G) < +\infty$ implies that $P\{B^c(G)\} < 1$. Therefore, $P\{C(K, G)\} \rightarrow P\{B^c(G)\} = 1 - \prod_{k=1}^{+\infty} [1 - r(k, G)] < 1$ as $K \rightarrow +\infty$.

Case 2) $\sum_{K=1}^{+\infty} r(K, G) = +\infty$, but $P\{C^c(K, G)|B^c(K-1, G)\} \not\rightarrow 0$ as $K \rightarrow +\infty$.

If $P\{C^c(K, G)|B^c(K-1, G)\} \geq \varepsilon > 0$ for all $K \geq K_0$ for some $K_0 \in \mathbb{Z}^+$, then $P\{C(K, G)\} \leq 1 - \varepsilon$ for all $K \geq K_0$.

Case 3) $\sum_{K=1}^{+\infty} r(K, G) < +\infty$ and $P\{C^c(K, G)|B^c(K - 1, G)\} \rightarrow 0$ as $K \rightarrow +\infty$.

If $P\{C^c(K, G)|B^c(K - 1, G)\} \geq \varepsilon > 0$ for all $K \geq K_0$ for some $K_0 \in Z^+$, then $P\{C(K, G)\} \leq (1 - \varepsilon)P\{B^c(G)\}$ for all $K \geq K_0$.

For case (1), $P\{C(K, G)\}$ converges to the global visit probability, as $K \rightarrow +\infty$. For case (2), $P\{C(K, G)\}$ is bounded above by one minus a strictly positive lower bound for $P\{C^c(K, G)|B^c(K - 1, G)\}$. For case (3), $P\{C(K, G)\}$ is bounded above by the global visit probability times one minus a strictly positive lower bound for $P\{C^c(K, G)|B^c(K - 1, G)\}$ (plus a value that can be made arbitrarily close to zero). These three cases illustrate the relationship between the global visit probability and the convergence in probability to an element of G for a GHC algorithm.

The conditions in Theorem 5 can be related to the convergence conditions for simulated annealing presented in Hajek [25], which show that simulated annealing converges in probability to an element in the set of global optima, G , if and only if $\sum_{k=1}^{+\infty} e^{-(d^*/t(k))} = +\infty$, where $t(k)$ is a non-increasing cooling schedule at iteration k (that approaches zero as $k \rightarrow +\infty$), and d^* is the maximum depth of all elements in L (i.e., the maximum gap in objective function value between an element of L and the solution in H that can reach another element of $L \cup G$ via local search, where the maximum is taken over all elements of L). This result assumes that the depth of all elements in G is infinity, hence once a global optimum is reached, simulated annealing cannot escape (with probability one) from such an element. Therefore, if GHC Algorithm A is simulated annealing, then under this assumption, ii) in Theorem 1 is always satisfied. Moreover, since the solution space is reachable, then at each macro iteration K , for K sufficiently large, there is a positive probability that the algorithm will need to escape from each element of L and move to an element of G . In particular, at each macro iteration K , for K sufficiently large, the conditional probability $r(K, G)$ has a component that includes the probability of escaping from the deepest local optimum. Therefore, using the law of total probability,

$$r(K, G) = \sum_{\omega \in L} r\{K, G|\omega \in L \text{ is visited at macro iteration } K - 1\} P\{\omega \in L \text{ is visited at macro iteration } K - 1\}. \quad (20.8)$$

Therefore, there exists a lower bound for $r(K, G)$ that is a linear function of $P\{\text{moving from the deepest element of } L \text{ to an element of } G\} = P\{\text{Accepting hill climbing moves out of the deepest element of } L \text{ to an element of } G\} = O(e^{-(d^*/t(K))})$, since the hill climbing random variable at macro iteration K is exponential with mean $1/t(K)$. Therefore, a sufficient condition for i) in Theorem 5 is $\sum_{K=1}^{+\infty} e^{-(d^*/t(K))} = +\infty$. A similar analysis shows that this infinite summation is also a necessary condition for i) in Theorem 5 (Jacobson and Yücesan [33]).

The finite global visit probability can also be used to compare different GHC algorithms. In particular, *random restart local search* (RR) can be used as a benchmark algorithm to compare the performance of various GHC algorithms. Other authors have compared simulated annealing with random restart local search. Ferreira and Zerovnik [16] develop bounds on the probability that simulated annealing obtains an optimal (or near-optimal) solution. They also show that random restart local search dominates simulated annealing, as measured by the probability of finding a globally optimal solution, as the number of restarts grows. Fox [20] notes that this result is only true if both the number of accepted and rejected moves are counted. He also provides a clever example to illustrate this point, and notes that comparing random restart local search and simulating annealing may not be prudent. Fox [19, 21] presents modifications of simulated annealing that circumvent this counting issue, hence yielding superior performing simulated annealing algorithm implementations. In particular, he develops the QUICKER algorithm to circumvent this problem. The QUICKER algorithm allows one to overcome the excessive number of rejections that occur with simulated annealing, particular in the later stages of the algorithm's execution when the temperature parameter has been reduced close to zero. Therefore, the QUICKER algorithm provides a unique way to improve the efficiency of simulated annealing, as well as allows simulated annealing to work with random restart local search, tabu search, and genetic algorithms (Fox [19]).

Let LS denote a single macro iteration (restart) of random restart local search. Random restart local search (RR) involves randomly selecting an initial solution (i.e., uniformly generated over the entire solution space, hence for all $\omega \in \Omega$, $P\{\text{Algorithm } LS \text{ is initialized at } \omega \in \Omega\} = 1/|\Omega|$), and applying local search until a local optimum is found. This process is repeated until K local optima are obtained. The best of these K solutions is then reported as the final solution. Therefore, each random restart corresponds to a single macro iteration. Using the GHC algorithm framework described in Section 20.2, the hill climbing (random) variables $R_k(\omega(i), \omega') = 0$ for all $\omega(i) \in \Omega$, $\omega' \in \eta(\omega(i))$, $k = 1, 2, \dots, K$, with $N(k)$ representing the number of micro iterations needed to reach the k^{th} element of $L \cup G$. In addition, after this element of $L \cup G$ is found, a new element of Ω is randomly (uniformly) generated to begin the next (inner loop) set of micro iterations.

To show how random restart local search can be used to compare different GHC algorithms, at each macro iteration (i.e., at each random restart), define the conditional probability

$$p(\omega) \equiv P\{\text{Algorithm } LS \text{ terminates in } G \mid \text{Algorithm } LS \text{ is initialized at } \omega \in \Omega\}.$$

Using $p(\omega)$ and the law of total probability, define

$$\begin{aligned}
P\{G - \text{stop}\} &\equiv P\{\text{Algorithm } LS \text{ terminates in } G\} \\
&= \sum_{\omega \in \Omega} p(\omega)^* P\{\text{Algorithm } LS \text{ is initialized at } \omega\} \\
&= [|G| + \sum_{\omega \in H} p(\omega)]/|\Omega|. \tag{20.9}
\end{aligned}$$

Similarly,

$$\begin{aligned}
P\{L - \text{stop}\} &\equiv P\{\text{Algorithm } LS \text{ terminates in } L\} \\
&= 1 - P\{G - \text{stop}\} \\
&= [|L| + \sum_{\omega \in H} (1 - p(\omega))]/|\Omega|. \tag{20.10}
\end{aligned}$$

Note that $P\{L - \text{stop}\}$ is defined by the neighborhood function and the neighborhood probability generating function. Therefore, if η_1 and η_2 are two neighborhood functions defined on a solution space Ω , where $\eta_1(\omega) \subseteq \eta_2(\omega)$ for all $\omega \in \Omega$, then $L(\eta_1) \supseteq L(\eta_2)$ and $P\{L - \text{stop}\}$ for η_1 is greater than or equal to $P\{L - \text{stop}\}$ for η_2 . In general, enriching the neighborhood function such that local optima are eliminated tends to decrease $P\{L - \text{stop}\}$.

Without loss of generality, assume that the neighborhood function is defined such that $P\{L - \text{stop}\} > 0$. If this is not the case, then $P\{G - \text{stop}\} = 1$, hence local search will always find a global optimum with every restart (i.e., $P\{(B_{RR}(K, G))^c\} = 1$ for all $K = 1, 2, \dots$). Under this assumption, Theorem 6 provides a convergence comparison between random restart local search and a GHC algorithm that does not visit G with probability one.

Theorem 6 (*Jacobson and Yücesan [33]*): *Let \mathbf{A} be a GHC algorithm that does not visit G with probability one. Let RR be a random restart local search algorithm. If the neighborhood function and the neighborhood probability generating function are defined on the solution space such that $0 < P\{L - \text{stop}\} < 1$, then there exists a macro iteration K_0 (which represents the number of restarts for the random restart local search algorithm) such that for all $K \geq K_0$,*

$$P\{(B_{RR}(K, G))^c\} \geq P\{(B_{\mathbf{A}}(K, G))^c\}.$$

Moreover, $K_0 \leq \ln[\alpha]/\ln[P\{L - \text{stop}\}]$, where $\prod_{k=1}^{+\infty} (1 - r(k, G)) \geq \alpha > 0$.

Theorem 6 shows that if a GHC algorithm does not visit G with probability one, then there exists a macro iteration beyond which random restart local search yields better results, as measured by the finite global visit probabilities at macro iteration K , $P\{(B_{\mathbf{A}}(K, G))^c\}$ and $P\{(B_{RR}(K, G))^c\}$ (i.e., the probabilities that the best solution visited to date (i.e., after K macro iterations) is in G , for algorithms \mathbf{A} and RR , respectively).

Theorem 7 provides sufficient conditions on the rate at which $P\{B_{\mathbf{A}}(K, G)\}$ converges to zero such that the performance of random restart local search and

a GHC algorithm can be compared. To described these conditions, define the non-negative value $\varphi_0 = -\ln(P\{L - \text{stop}\})$.

Theorem 7 (*Jacobson and Yücesan [33]*): *Let \mathbf{A} be a GHC algorithm that visits G with probability one. Let RR be a random restart local search algorithm. If the neighborhood function and the neighborhood probability generating function are defined on the solution space such that $0 < P\{L - \text{stop}\} < 1$, then*

- i) if $P\{B_{\mathbf{A}}(K, G)\} = O(e^{-\varphi K})$ for K large and $\varphi \geq \varphi_0$, then there exists a macro iteration K_0 such that for all $K \geq K_0$, $P\{(B_{RR}(K, G))^c\} \leq P\{(B_{\mathbf{A}}(K, G))^c\}$,
- ii) if $P\{B_{\mathbf{A}}(K, G)\} = O(e^{-\varphi K})$ for K large and $\varphi < \varphi_0$, then there exists a macro iteration K_0 such that for all $K \geq K_0$, $P\{(B_{RR}(K, G))^c\} \geq P\{(B_{\mathbf{A}}(K, G))^c\}$,
- iii) if $P\{B_{\mathbf{A}}(K, G)\} = o(e^{-\varphi K})$ for K large and $\varphi \geq \varphi_0$, then there exists a macro iteration K_0 such that for all $K \geq K_0$, $P\{(B_{RR}(K, G))^c\} \leq P\{(B_{\mathbf{A}}(K, G))^c\}$, and
- iv) if $1/P\{B_{\mathbf{A}}(K, G)\} = o(e^{\varphi K})$ for K large and $\varphi \leq \varphi_0$, then there exists a macro iteration K_0 such that for all $K \geq K_0$, $P\{(B_{RR}(K, G))^c\} \geq P\{(B_{\mathbf{A}}(K, G))^c\}$,

where $P\{B_{\mathbf{A}}(K, G)\} = O(e^{-\varphi K})$ for K large means that $P\{B_{\mathbf{A}}(K, G)\}e^{\varphi K} \rightarrow$ a constant as $K \rightarrow +\infty$, $P\{B_{\mathbf{A}}(K, G)\} = o(e^{-\varphi K})$ for K large means that $P\{B_{\mathbf{A}}(K, G)\}e^{\varphi K} \rightarrow 0$ as $K \rightarrow +\infty$, and $1/P\{B_{\mathbf{A}}(K, G)\} = o(e^{\varphi K})$ for K large means that $1/[P\{B_{\mathbf{A}}(K, G)\}e^{\varphi K}] \rightarrow 0$ as $K \rightarrow +\infty$, and K_0 represents the number of restarts for the random restart local search algorithm.

Theorem 7 compares the performance of random restart local search and a GHC algorithm that visits G with probability one under four different cases. The remaining two cases (i.e., $P\{B_{\mathbf{A}}(K, G)\} = o(e^{-\varphi K})$ for K large with $\varphi < \varphi_0$ and $1/P\{B_{\mathbf{A}}(K, G)\} = o(e^{\varphi K})$ for K large with $\varphi > \varphi_0$) are inconclusive, hence the performance of each of the algorithms becomes problem specific, where general results cannot be obtained using the approach presented here. Note that an identical analysis can be used to show that Monte Carlo search yields the same conclusion obtained in both Theorems 6 and 7, with the new definitions $P\{G - \text{stop}\} \equiv |G|/|\Omega|$ and $P\{L - \text{stop}\} \equiv 1 - P\{G - \text{stop}\}$ (i.e., $P\{L - \text{stop}\}$ is no longer defined as in (20.10)). However, $P\{L - \text{stop}\}$ for Monte Carlo search will be greater than or equal to $P\{L - \text{stop}\}$ for random restart local search. This means that there exists GHC algorithms for which ii) and iv) in Theorem 7 are satisfied for random restart local search, but are not satisfied for Monte Carlo search, while there are no GHC algorithms that visit G with probability one for which the reverse is true. Therefore, random restart local search dominates Monte Carlo search.

The results in Theorem 7 suggest that the value for $P\{L - \text{stop}\}$ for random restart local search determines its relative performance with GHC algorithms.

In particular, if $P\{L - \text{stop}\} = 1$, then ii) and iv) can never be satisfied. This means that random restart local search cannot be proven to dominate GHC algorithms that visit G with probability one using Theorem 7. In general, $1 - \delta \leq P\{L - \text{stop}\} < 1$ for some $\delta > 0$ close to zero. Therefore, $\varphi_0 = -\ln(P\{L - \text{stop}\}) \geq -\ln(1 - \delta) \geq \delta$. This means that the larger the value for $P\{L - \text{stop}\}$ (i.e., the closer to one), the larger the number of restarts needed for random restart local search to dominate such a GHC algorithm. For practical purposes, this suggests that for solution spaces (and associated neighborhood functions) with many local optima, it may be more effective to use a GHC algorithm. Therefore, the design and structure of the neighborhood function (hence the number of and distribution of local optima in the solution space) is a key factor in determining whether random restart local search is preferable over a GHC algorithm.

The results in Theorems 6 and 7 do not take into account the number of (micro) iterations between each macro iteration. For random restart local search, this represents the number of iterations needed to reach a local optimum from each randomly generated initial solution, while for a GHC algorithm, this represents the number of iterations between visits to local optima. Note that as a GHC algorithm progresses, and the hill climbing random variables approach the value zero with probability one, the number of iterations between the macro iterations may be very small, as the algorithm gets stuck in a local optimum with increasing probability. If this local optimum is not a global optimum, then $P\{(B_A(K, G))^c\}$ will remain constant (or change very slowly) for all future macro iterations. Fox [19, 21] notes this point for simulated annealing, and suggests alternate ways to improve the performance of simulated annealing that overcomes this situation.

The results in Theorem 7 have important implications for practitioners using simulated annealing. Using the necessary and sufficient convergence condition for simulated annealing in Hajek [25], $r(K, G)$ can be bounded above and below by functions that are $O(e^{-(d^*/t(K))})$ for K sufficiently large. This means that there exists constants $\gamma_1 > 0$ and $\gamma_2 > 0$ and a macro iteration K_0 such that $\gamma_1 e^{-(d^*/t(K))} \leq r(K, G) \leq \gamma_2 e^{-(d^*/t(K))}$ for all $K \geq K_0$. Therefore, for all $K \geq K_0$

$$\prod_{k=1}^K (1 - \gamma_2 e^{-(d^*/t(k))}) \leq P\{B_{SA}(K, G)\} = \prod_{k=1}^K [1 - r(k, G)] \leq \prod_{k=1}^K (1 - \gamma_1 e^{-(d^*/t(k))}). \quad (20.11)$$

Hajek's condition on the cooling schedule (and indirectly, through the choice of neighborhood function which defines d^*), $\sum_{K=1}^{+\infty} e^{-(d^*/t(K))} = +\infty$, places restrictions on the rate at which the cooling schedule $t(k)$ approaches zero. Suppose that the cooling schedule is defined such that $e^{-(d^*/t(K))} = \lambda(1/k)^\delta$ for $k \geq 2$ and $\lambda \in Z^+$, for some $0 < \delta \leq 1$. Note that $t(k)$ could also be defined using iterated logarithms (e.g., $e^{-(d^*/t(K))} = \lambda(1/k(\ln(k)))$) or any form provided that Hajek's condition is satisfied.

From the expression in (20.11), one can obtain an upper and lower bound on $e^{\varphi K} P\{B_{SA}(K, G)\}$ as $K \rightarrow +\infty$. In particular, for all $K \geq K_0$,

$$e^{\varphi K} \prod_{k=1}^K (1 - \gamma_2 e^{-(d^*/t(k))}) \leq e^{\varphi K} P\{B_{SA}(K, G)\} \leq e^{\varphi K} \prod_{k=1}^K (1 - \gamma_1 e^{-(d^*/t(k))})$$

which leads to

$$e^{\varphi K} \prod_{k=1}^K (1 - \gamma_2 \lambda(1/k)^\delta) \leq e^{\varphi K} P\{B_{SA}(K, G)\} \leq e^{\varphi K} \prod_{k=1}^K (1 - \gamma_1 \lambda(1/k)^\delta). \quad (20.12)$$

Since $\prod_{k=1}^{+\infty} (1 - \gamma_1 \lambda(1/k)^\delta) = 0$ if and only if $\sum_{k=1}^{+\infty} (1/k)^\delta = +\infty$, then the rate at which $\prod_{k=1}^K (1 - \gamma_1 \lambda(1/k)^\delta)$ approach zero as $K \rightarrow +\infty$ relative to the rate at which $e^{\varphi K}$ approaches infinity determines which of the four cases described in Theorem 7 apply to this simulated annealing algorithm.

To determine this rate, note that for some $k_0 \in \mathbb{Z}^+$ where $\gamma_2 \lambda(1/k_0)^\delta < 1$,

$$\ln\left(\prod_{k=k_0}^K (1 - \gamma_1 \lambda(1/k)^\delta)\right) = \sum_{k=k_0}^K \ln(1 - \gamma_1 \lambda(1/k)^\delta) \leq \sum_{k=k_0}^K -\gamma_1 \lambda(1/k)^\delta. \quad (20.13)$$

Using the integral approximation for (20.14), this expression is $O(-\ln(K))$ as $K \rightarrow +\infty$ for $\delta = 1$, and $O(-K^{1-\delta})$ as $K \rightarrow +\infty$ for $0 < \delta < 1$. By taking the exponential function in (20.14), then $\prod_{k=1}^K (1 - \gamma_1 \lambda(1/k)^\delta)$ is $O(1/K)$ as $K \rightarrow +\infty$ for $\delta = 1$, and $O(\exp(K^{\delta-1}))$ as $K \rightarrow +\infty$ for $0 < \delta < 1$. Therefore, $e^{\varphi K} \prod_{k=1}^K (1 - \gamma_1 \lambda(1/k)^\delta) \rightarrow +\infty$ as $K \rightarrow +\infty$ for $0 < \delta \leq 1$. The same conclusions are obtained from the lower bound in (20.12). In particular,

$$\begin{aligned} \ln\left(\prod_{k=k_0}^K (1 - \gamma_2 \lambda(1/k)^\delta)\right) &= \sum_{k=k_0}^K \ln(1 - \gamma_2 \lambda(1/k)^\delta) \\ &\leq \sum_{k=k_0}^K -\gamma_2 \lambda(1/k)^\delta / (1 - \gamma_2 \lambda(1/k)^\delta). \end{aligned} \quad (20.14)$$

Once again, using the integral approximation for (20.14), this analysis yields the same results. Therefore, $e^{\varphi K} \prod_{k=1}^K (1 - \gamma_2 \lambda(1/k)^\delta) \rightarrow +\infty$ as $K \rightarrow +\infty$ for $0 < \delta \leq 1$. This means that conditions i), ii), and iii) cannot hold for this simulated annealing algorithm. Therefore, either condition iv) holds (provided $\varphi \leq \varphi_0$), hence random restart local search dominates this simulated annealing algorithm, or the results are inconclusive (if $\varphi > \varphi_0$). Note that if φ_0 is very close to zero, hence $P\{L - \text{stop}\}$ is very close to one, then if condition iv) holds, the value for K_0 may be prohibitively large, by comparing $e^{\varphi K}$ with functions that are $O(1/K)$ or $O(\exp(K^{\delta-1}))$. Therefore, for random restart local search the form of the cooling schedule for simulated annealing and the choice of neighborhood function (that defines the value for $P\{L - \text{stop}\}$) determine the

relative performance of these algorithms. To make this result more practical, it is necessary to determine the number of restarts/macro iterations K_0 that are needed such that $P\{(B_{RR}(K, G))^c\} \geq P\{(B_A(K, G))^c\}$ for all $K \geq K_0$; this is a current topic of research and investigation.

20.6 SUMMARY AND CONCLUSIONS

This paper provides a survey of recent results that use the GHC algorithm framework for studying the performance of local search algorithms. The majority of previous theoretical results on local search algorithms have focused on simulated annealing. For instance, Cohn and Fielding [9] conduct a detailed analysis of the affect of various cooling schedules on the performance of simulated annealing. They note that convergent simulated annealing algorithms are often too slow in practice, whereas non-convergent algorithms may be preferred for good finite-time performance. To test this conjecture, they analyze various cooling schedules and present cases where repeated independent runs using a non-convergent cooling schedule provide acceptable results in practice. Moreover, they provide examples of when it is both practically and theoretically justified to use boiling, fixed temperature, or even fast cooling schedules and apply these cooling schedules to traveling salesman problems of various sizes. The potential advantage of developing similar results as these within the GHC algorithm can broaden the scope and breadth of applications for other local search algorithms, including tabu search and genetic algorithms.

It should be noted that most theoretical results on simulated annealing are concerned with asymptotic convergence. The results in Theorems 6 and 7 suggest that random restart local search may outperform simulated annealing provided a sufficiently large number of restarts are executed. Therefore, the primary value of simulated annealing may be for shorter, finite-time executions where it is reasonable to assume that near-optimal solutions can be reached quickly. This suggests that finite-time performance measures and results for simulated annealing may be more valuable than the asymptotic convergence results that dominate the literature. These results also imply that the primary value of random restart local search is when it is applied over an infinite horizon. Research is in progress to use and extend these results to identify both convergent and non-convergent GHC algorithm formulations that perform well over finite horizons, as well as to determine the number of random restarts that are needed to satisfy the inequalities in Theorem 7.

Acknowledgments: This research has been supported in part by the Air Force Office of Scientific Research (F49620-98-1-0111, F49620-98-1-0432, F49620-01-1-0007) and the National Science Foundation (DMI-9907980). The author would like to thank Dr. Pierre Hansen, Dr. Diane E. Vaughan, and an anonymous referee for their encouragement of this line of research as well as their valuable and insightful comments and feedback on the original version of the manuscript.

References

- [1] E.H.L. Aarts, J.H.M. Korst, and P.J.M. van Laarhoven. Simulated Annealing. In: *Local Search in Combinatorial Optimization* E.H.L. Aarts and J.K. Lenstra, editors, pages 91–120, Wiley, 1997.
- [2] E.H.L. Aarts and J.K. Lenstra. *Local Search in Combinatorial Optimization*, Wiley, 1997.
- [3] I. Althofer and K.U. Koschnick. On the Convergence of Thresholding Accepting. *Applied Mathematics and Optimization*, 24:183–195, 1991.
- [4] E. Balas and A. Vazacopoulos. Guided Local Search with Shifting Bottleneck for Job Shop Scheduling. *Management Science*, 44:262–275, 1998.
- [5] O. Catoni. Metropolis, Simulated Annealing, and Iterated Energy Transformation Algorithms: Theory and Experiments. *Journal of Complexity*, 12:595–623, 1996.
- [6] S. Ceria, P. Nobili, and A. Sassano. A Lagrangian-Based Heuristic for Large-Scale Set Covering Problems. *Mathematical Programming*, 81:215–228, 1998.
- [7] I. Charon and O. Hudry. The Noising Method: A New Method for Combinatorial Optimization. *Operations Research Letters*, 14:133–137, 1993.
- [8] T.S. Chiang and Y.Y. Chow. A Limit-Theorem for a Class of Inhomogeneous Markov-Processes. *Annals of Probability*, 17:1483–1502, 1989.
- [9] H. Cohn and M. Fielding. Simulated Annealing: Searching for an Optimal Temperature Schedule. *SIAM Journal on Optimization*, 9:779–802, 1999.
- [10] S.A. Cook. The Complexity of Theorem-Proving Procedures. In: *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, Association for Computing Machinery, New York, pages 151–158, 1971.
- [11] C.R. Coullard, A.B. Gamble, and P.C. Jones. Matching Problems in Selective Assembly Operations. *Annals of Operations Research*, 76:95–107, 1998.
- [12] M.P. Desai. Some Results Characterizing the Finite Time Behaviour of the Simulated Annealing Algorithm. *Sadhana-Academy Proceedings in Engineering Sciences*, 24:317–337, 1999.
- [13] G. Dueck and T. Scheuer. Thresholding Accepting: A General Purpose Optimization Algorithm Appearing Superior to Simulated Annealing. *European Journal of Operational Research*, 46:271–281, 1990.
- [14] R.W. Eglese. Simulated Annealing: A Tool for Operational Research. *European Journal of Operational Research*, 46:271–281, 1990.

- [15] U. Faigle and W. Kern. Some Convergence Results for Probabilistic Tabu Search. *ORSA Journal on Computing*, 4:32–37, 1992.
- [16] A.G. Ferreira and J. Zerovnik. Bounding the Probability of Success of Stochastic Methods for Global Optimization. *Computers and Mathematics Applications*, 25:1–8, 1993.
- [17] M.A. Fleischer. Simulated Annealing: Past, Present, and Future. In: *Proceedings of the 1995 Winter Simulation Conference*, C. Alexopoulos, K. Kang, W.R. Lilegdon, and D. Goldsman, editors, pages 155–161, Institute of Electrical and Electronics Engineers Press, 1995.
- [18] M.A. Fleischer and S.H. Jacobson. Information Theory and the Finite-Time Behavior of the Simulated Annealing Algorithm: Experimental Results. *INFORMS Journal on Computing*, 11:35–43, 1999.
- [19] B.L. Fox. Integrating and Accelerating Tabu Search, Simulated Annealing, and Genetic Algorithms. *Annals of Operations Research*, 41:47–67, 1993.
- [20] B.L. Fox. Random Restarting versus Simulated Annealing. *Computers and Mathematics Applications*, 27:33–35, 1994.
- [21] B.L. Fox. Faster Simulated Annealing. *SIAM Journal of Optimization*, 5:488–505, 1995.
- [22] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.
- [23] F. Glover and M. Laguna. *Tabu Search*. Kluwer, 1997.
- [24] P. Hansen and B. Jaumard. Algorithms for the Maximum Satisfiability Problem. *Computing*, 44:279–303, 1990.
- [25] B. Hajek. Cooling Schedules for Optimal Annealing. *Mathematics of Operations Research*, 13:311–329, 1988.
- [26] Y.-C. Ho, R. Sreenivas, and P. Vakili. Ordinal Optimization of Discrete Event Dynamic Systems. *Discrete Event Dynamical Systems*, 2:61–88, 1992.
- [27] D.S. Hochbaum and A. Pathria. Generalized p -Center Problems: Complexity Results and Approximation Algorithms. *European Journal of Operational Research*, 100:594–607, 1997.
- [28] S.H. Jacobson, A.W. Johnson, K.A. Sullivan, M.A. Fleischer, and A. Kumar. Metaheuristics for a Flexible Assembly System Design Problem. *Journal of Heuristics*, 3:139–159, 1997.

- [29] S.H. Jacobson, K.A. Sullivan, and A.W. Johnson. Generalized Hill Climbing Algorithms for Discrete Manufacturing Process Design Problems using Computer Simulation Models. In: *Proceedings of the 11th European Simulation Multiconference*, pages 473–478, 1997.
- [30] S.H. Jacobson, K.A. Sullivan, and A.W. Johnson. Discrete Manufacturing Process Design Optimization Using Computer Simulation and Generalized Hill Climbing Algorithms. *Engineering Optimization*, 31:247–260, 1998.
- [31] S.H. Jacobson and E. Yücesan. On the Complexity of Verifying Structural Properties of Discrete Event Simulation Models. *Operations Research*, 47:476–481, 1999.
- [32] S.H. Jacobson and E. Yücesan. On the Effectiveness of Generalized Hill Climbing Algorithms. Technical Report, University of Illinois, 2001.
- [33] S.H. Jacobson and E. Yücesan. Assessing the Performance of Generalized Hill Climbing Algorithms. Technical Report, University of Illinois, 2001.
- [34] A.W. Johnson and S.H. Jacobson. A Convergence Result for a Class of Generalized Hill Climbing Algorithms. To appear in: *Applied Mathematics and Computation*.
- [35] A.W. Johnson and S.H. Jacobson. On the Convergence of Generalized Hill Climbing Algorithms. To appear in: *Discrete Applied Mathematics*.
- [36] E.L. Johnson and G.L. Nemhauser. Recent Developments and Future-Directions in Mathematical Programming. *IBM Systems Journal*, 31:79–93, 1992.
- [37] A. Kumar, S.H. Jacobson, and E.C. Sewell. Computational Analysis of a Flexible Assembly System Design Problem. *European Journal Operational Research*, 123:17–36, 2000.
- [38] A.M. Law and W.D. Kelton. *Simulation Modeling and Analysis*, McGraw Hill, 1991.
- [39] G.E. Liepins and M.R. Hilliard. Genetic Algorithms: Foundations and Applications. *Annals of Operations Research*, 21:31–58, 1989.
- [40] C. Mazza. Parallel Simulated Annealing. *Random Structures and Algorithms*, 3:139–148, 1992.
- [41] D. Mitra, F. Romeo, and A. Sangiovanni-Vincentelli. Convergence and Finite-Time Behavior of Simulated Annealing. *Advances in Applied Probability*, 18:747–771, 1986.
- [42] H. Mühlenbein. Genetic Algorithms. In: *Local Search in Combinatorial Optimization*, E.H.L. Aarts and J.K. Lenstra, editors, pages 137–172, Wiley 1997.

- [43] G.L. Nemhauser. The Age of Optimization - Solving Large-Scale Real-World Problems. *Operations Research*, 42:5–13, 1994.
- [44] S.M. Ross. *A First Course in Probability*, Macmillan Publishing Company, 1988.
- [45] P.C. Schuur. Classification of Acceptance Criteria for the Simulated Annealing Algorithm. *Mathematics of Operations Research*, 22:266–275, 1997.
- [46] L. Shi and S. Olafsson. Nested Partitions Method for Global Optimization. *Operations Research*, 48:390–407, 2000.
- [47] R. Srichander. Efficient Schedules for Simulated Annealing. *Engineering Optimization*, 24:161–176, 1995.
- [48] K.A. Sullivan and S.H. Jacobson. Ordinal Hill Climbing Algorithms for Discrete Manufacturing Process Design Optimization Problems. *Discrete Event Dynamical Systems*, 10:307–324, 2000.
- [49] K.A. Sullivan and S.H. Jacobson. A Convergence Analysis of Generalized Hill Climbing Algorithms. To appear in: *IEEE Transactions on Automatic Control*.
- [50] C.A. Tovey. Local Improvement on Discrete Structures. In: *Local Search in Combinatorial Optimization*, E.H.L. Aarts and J.K. Lenstra, editors, pages 57–89, Wiley, 1997.
- [51] M. Yannakakis. Computational Complexity. In: *Local Search in Combinatorial Optimization*, E.H.L. Aarts and J.K. Lenstra, editors, pages 19–55, Wiley, 1997.
- [52] E. Yücesan and S.H. Jacobson. Computational Issues for ACCESSIBILITY in Discrete Event Simulation. *ACM Transactions on Modeling and Computer Simulation*, 6:53–75, 1996.

21

ANT COLONY OPTIMIZATION: AN OVERVIEW

Vittorio Maniezzo and Antonella Carbonaro

Scienze dell'Informazione, University of Bologna

Via Sacchi 3, 47023 Cesena, Italy

{maniezzo, carbonar}@csr.unibo.it

Abstract: Ant Colony Optimization (ACO) is a class of constructive metaheuristic algorithms sharing the common approach of constructing a solution on the basis of information provided both by a standard constructive heuristic and by previously constructed solutions. This tutorial is composed of three parts. The first one frames the ACO approach in current trends of research on metaheuristic algorithms for combinatorial optimization; the second outlines current research within the ACO framework, reporting recent results obtained on different problems, while the third part focuses on a particular research line, the ANTS metaheuristic, providing some details on the algorithm and presenting results recently obtained on the quadratic and on the frequency assignment problems.

21.1 INTRODUCTION

Complex societal systems pose a lot of problems of combinatorial nature. Trucks have to be routed, depots or sale points have to be located, communication networks have to be designed, containers have to be filled, radio links must have an associated frequency, wood or leather masters have to be cut: the list could get long indeed. Complexity theory tells us that the combinatorial problems underlying these applications are often not solvable in polynomial time. Actual practice tells us that, unfortunately, the size of the instances met in real world applications rules out the possibility of solving them to optimality in an acceptable time. Nevertheless, these instances must be solved, thus the need to bow down to suboptimal solutions, provided that they are of acceptable quality and that they can be found in acceptable time.

Several approaches have been proposed for designing “acceptable quality” solutions under the “acceptable time” constraint. Besides studying the particular instances of interest, in the hope of being in a polynomial special case, one can in fact try to design an algorithm that guarantees to find a solution within a known gap from optimality, i.e. an *approximation algorithm*, or try to design an algorithm that guarantees that for instances big enough, the probability of getting a bad solution is very small, i.e. a *probabilistic algorithm*. Or one can lower even more the expectations and accept an algorithm that offers no guarantees, in front of some evidence that historically, on the average, that algorithm has a good record track on the quality/time trade off for the problem of interest. This is the area of *heuristic algorithms*.

Heuristic algorithms have a long tradition, the actual necessity to solve difficult instances has pushed research in the area. It is possible to identify different approaches to how to design such algorithms. We maintain in this paper that it is possible to identify in the literature three main classes of heuristic algorithms. The historically first one concentrates on structural properties of the problem to solve and tries to use them for defining constructive or local search algorithms. Algorithms following the second approach are those usually denoted by the term “metaheuristics”, a name first proposed by Glover in [25], and focus on the guidance of a constructive or a local search algorithm in order to allow it to overcome critical situations (which usually means to escape from local optima). Finally, a recent trend is focused on incorporating strong results from mathematical programming practice, usually from exact solution method design, into an heuristic framework.

The paper presents a brief overview on the essential features of these three types of heuristic paradigms, then focuses on one particular family of algorithms, those that follow the Ant Colony Optimization (ACO) paradigm. Finally, the third part of this overview describes in some more detail one algorithm of the ACO class, called ANTS.

The paper is structured as follows. Section 21.2 introduces the general issues that contribute to define the three classes of heuristic algorithm design approaches introduced above. Section 21.3 describes the common elements of the heuristics belonging to the ACO class. Section 21.4 concentrates on the ANTS approach, describing its essential ingredients, while Section 21.5 presents computational results obtained by ANTS algorithms on two well known combinatorial optimization (CO) problems: the quadratic assignment and the frequency assignment problems. Finally, Section 21.6 presents a brief discussion on the issues raised by this overview.

21.2 HEURISTIC FRAMEWORK

This section introduces a classification of heuristic algorithms into three classes, defining some essential traits of each of them and presenting typical examples of algorithms of each class. Notice that, although the three classes of algorithms were successively presented in the literature, this does not necessary reflect

on the comparative effectiveness of the algorithms: specific problems are best solved by either of them.

All algorithms will be described according to the following common problem framework. A combinatorial optimization problem is defined over a set $C = \{c_1, \dots, c_n\}$ of basic components. A subset S of components represents a solution of the problem; $F \subseteq 2^C$ is the subset of *feasible solutions*, thus a solution S is feasible if and only if $S \in F$. A *cost function* z is defined over the solution domain, $z : 2^C \rightarrow \mathbb{R}$, the objective of the algorithm is to find a minimum cost feasible solution S^* , i.e., to find $S^* \in F$ such that $z(S^*) \leq z(S), \forall S \in F$. Failing this, the algorithm anyway returns the best feasible solution found, $\bar{S} \in F$.

As a running example of CO problem, we can use the traveling salesman problem (TSP), defined over a weighted undirected complete graph $G=(V,E,D)$, where V is the set of vertices, E is the set of edges and D is the set of edge weights, corresponding to distances between vertices. For the TSP we have that the component set corresponds to E ($C=E$), that F corresponds to the set of Hamiltonian cycles in G and that z is defined to be, for each feasible solution S , the sum of the weights associated to the edges belonging to S .

21.2.1 Type 1 heuristics: focus on solution structure

Type 1 heuristics are methods that exploit structural properties of a feasible solution in order to quickly come to a good feasible solution. Generally they belong to one of two main classes: *constructive heuristics* or *local search heuristics*.

Constructive heuristics are often based on the following general framework:

1. Sort the components in C by increasing costs.
2. Set $\bar{S} = \emptyset$ and $i = 1$.
3. Repeat
 - If $(\bar{S} \cup \{c_i\}$ is a partial feasible solution) then $\bar{S} = \bar{S} \cup \{c_i\}$.
 - $i = i + 1$.
- Until $\bar{S} \in F$.

Notice that such a simple approach can yield optimal solutions for certain problems, eg. the minimum spanning tree of a graph, while in other cases it could be unable even to construct a feasible solution.

In the case of the TSP this approach would suggest to order all edges by increasing edge length, take the least cost one and add to it increasing cost edges, provided they do not close subtours, until a Hamiltonian circuit is completed. More involved constructive strategies give rise to well-known TSP heuristics, like the Farthest Insertion, the Nearest Neighbor or the Sweep algorithms.

Local search heuristics preliminarily need the definition of a *solution neighborhood*. The neighborhood of a solution S , $N(S)$, is a subset of 2^C defined by a specific *neighborhood definition function* $N : 2^C \rightarrow 2^{2^C}$. It is common practice, though not mandatory, to restrict the attention to feasible solutions only, thus

to neighborhood definition functions in the form $N : \mathbf{F} \rightarrow 2^{\mathbf{F}}$. The specific neighborhood function used has a deep impact on the algorithm performance and its choice is left to the algorithm designer.

Given N , a local search algorithm is an algorithm with the following structure:

1. Generate an initial feasible solution S .
2. Find S' , such that $S' \in N(S)$ and $z(S') = \min\{z(\hat{S}), \forall \hat{S} \in N(S)\}$.
3. If $z(S') < z(S)$
 $S = S'$, go to step 2.
4. $\bar{S} = S$.

The update of a solution, in step 3, which transforms solution S into solution S' is called a *move* from S to S' .

Despite its apparent triviality, the design of a good local search algorithm can be very intricate. While we refer the reader to [43] and [2] for a thorough discussion, we just want to point out that a deep impact decision is, at step 2, whether to explore the whole of $N(S)$, find the best solution S' in it and check if this improves over S , or to update S as soon as an improving S' solution is found, even if it not guaranteed to be the best in $N(S)$.

Several mathematical properties can be studied on the structure of the neighborhood sets, and it is possible to prove that there are problems for which a local search approach guarantees to find an optimal solution: suffice it to say that the simplex algorithm essentially follows the above framework.

In the case of the TSP, two very simple and widely used local search methods are 2-opt and 3-opt, which take a solution in the form of a list of n vertices and exhaustively try to swap the elements of all pairs or triplets of vertices in it, respectively. More complicated neighborhoods give rise to more effective heuristics, among which the method of Lin and Kernighan [32] deserves a special citation.

21.2.2 Type 2 heuristics: focus on heuristic guidance

Heuristics of type 1 can perform very well, at times optimally, but can also get stuck with very poor quality solutions. As mentioned, one of the major problems to face is the possibility to get trapped in local optima, with no chance to escape from them. To improve the quality of the solutions they can produce, thus the size of the problems that can be satisfactorily solved, a new wave of approaches has been presented starting from about the mid '70ies. They consist in algorithms which explicitly or implicitly manage the trade-off between search *diversification*, when evidence exists that search is going on in bad regions of the search space, and *intensification*, aimed at finding the best solutions within the region being analyzed.

These algorithms have been called *metaheuristics*. Among the best known ones, we can list:

- [1] Simulated annealing;

- [2] Tabu search;
- [3] GRASP; and
- [4] Genetic algorithms.

The list could be much longer than this, since this field of research is very active as this volume testifies. We concentrate on these just to show how the diversification/intensification issue was tackled in each of them and to set the stage for an analogous discussion in the case of ACO.

21.2.2.1 Simulated annealing. The essential idea motivating Simulated Annealing (SA) [1] is to modify local search in order to accept, in probability, worsening moves. The framework of SA is:

1. Generate an initial feasible solution S , initialize the best so far solution $\bar{S} = S$ and initialize parameter T.
2. Generate S' , where $S' \in N(S)$.
3. If $z(S') < z(S)$ then $S = S'$, if ($z(\bar{S}) > z(S)$) $\bar{S} = S'$
else accept to set $S = S'$ with probability $p = e^{-\frac{z(S') - z(S)}{T}}$.
4. If (annealing condition) decrease T.
5. If not(end condition) go to step 2.

SA, with respect to local search, only accepts partial neighborhood explorations and implements the intensification/diversification strategy by means of the annealing (decrease) of parameter T. The end condition can be on the CPU time elapsed, on the number of iterations or on other parameters specific of the run.

21.2.2.2 Tabu search. The idea behind Tabu Search (TS) [28] is again to modify local search in order to escape from local minima. In the TS case this is accomplished by moving onto the best solution of the neighborhood at each iteration, even though it is worse than the current one. A special memory structure called *tabu list*, TL, forbids the return to already explored solutions. The general structure of the TS algorithm is as follows.

1. Generate an initial feasible solution S , set $\bar{S} = S$ and initialize $TL = \emptyset$.
2. Find S' , such that $S' \in N(S)$ and $z(S') = \min\{z(\hat{S}), \forall \hat{S} \in N(S), \hat{S} \notin TL\}$.
3. $S = S'$, $TL = TL \cup S$, if ($z(\bar{S}) > z(S)$) set $\bar{S} = S$.
4. If not(end condition) go to step 2.

In the case of TS the intensification/diversification strategy must be explicitly implemented with reference to the actual memory structures used to store the tabu list. Several strategies are possible, we refer the interested reader to [28] for a discussion.

21.2.2.3 GRASP. GRASP (Greedy Randomized Adaptive Search Procedure) [19] departs from the SA and TS line of trying to escape from local minima, in that the idea here is to restart search from another promising region of the search space as soon as a local optimum is reached. GRASP essentially consists in a multistart approach, i.e., an approach which simply iterates local search starting from different initial solutions, with a suggestion on how to construct the initial solutions. GRASP combines the two algorithms of Section 21.2.1 in the following common framework.

1. Build a solution S ($=\bar{S}$ in the first iteration) by a constructive greedy randomized procedure.
2. Apply a local search procedure starting from S and producing S' .
3. If $z(S') < z(\bar{S})$ then $\bar{S} = S'$.
4. If not(end condition) go to step 2.

The greedy randomized procedure of step 2 is the constructive heuristic of Section 21.2.1 where, at step 3, the component to append is not necessarily the first feasible one but a feasible one chosen in probability among the most promising ones (i.e., at the top of the list). The diversification/intensification issue is handled by controlling the randomization of the constructive procedure.

21.2.2.4 Genetic algorithms. Genetic Algorithms (GA) [29] follow a completely different approach from those so far presented, in that they do not rely on construction or local search but on the parallel update of a set R of solutions. This is achieved by *recombining* subsets of 2 elements in R , the *parent sets*, to obtain new solutions. Actually, GA should not be properly called a metaheuristic algorithm, since they do not guide a lower level heuristic. However, to be effective on CO problems they are often hybridized with SA, TS or local search. The general framework is as follows.

1. Initialize a set R of solutions.
2. Construct a set R_c of solutions by recombining pairs of randomly chosen elements in R .
3. Construct a set R_m of solutions by randomly modifying elements in R_c .
4. Construct the new set R by extracting elements from R_m , following a Montecarlo strategy with repetitions.
5. If not(end condition) go to step 2.

Different, often problem-dependent solutions have been proposed for the recombination, the modification and the extraction operators at steps 2, 3 and 4, respectively. The diversification/intensification strategy here depends on the details of the operators above, which could lead to sets of similar solution, thus fostering intensification, or to sets of solution widely scattered over the search space, thus promoting diversification.

21.2.2.5 Ant system. Finally, a first framing of an ACO algorithm. By Ant System (AS) we mean the first ACO algorithm presented [11, 14, 15]. The idea underlying it was to modify a constructive heuristic, in the spirit of the first step of GRASP, so that the ordering of the components could be recalculated at each iteration taking into account not only the a priori expectation, η_i , of the usefulness of a particular component c_i , as in standard constructive approaches, but also an a posteriori measure, τ_i , of the goodness of solutions constructed using that particular component. The general framework is as follows.

1. Initialize a set A of partial solutions: $a_i = \emptyset, i = 1, \dots, m$.
2. For $i=1$ to m
 - choose a component c_j to append to solution a_i with probability given as a function of a_i, η_j, τ_j .
3. If the solutions in A are not complete solutions, go to step 2.
4. Evaluate $z(a_i), i = 1, \dots, m$ and update $\tau_j, j = 1, \dots, n$ accordingly.
5. If not(end condition) go to step 1.

Step 4 can be conveniently integrated with a local search starting from each a_i solution in A . Diversification and intensification are essentially controlled by the τ management policy, which will be detailed in Sections 21.3 and 21.4.

The general ACO framework can be specialized in different ways, we mention here the possibility of including mathematical programming results, which lead the resulting algorithms to be more suitably included in the third class of heuristics, see for example Subsection 21.2.3.3.

21.2.3 Type 3 heuristics: focus on mathematical programming contribution

It can be noticed that all algorithms above rely on simple general ideas. Obviously, when they are better detailed and adapted to a particular problem, mathematical considerations become an issue. However, the design of a meta-heuristic for a problem can be done neglecting much of the mathematical programming results obtained on that same problem. A straightforward consideration is that results derived from the study of exact approaches could be exploited also in the framework of heuristic design. This is an issue common to some recently presented techniques. Actually, an hybridation of lower bounding techniques like linear relaxation or subgradient optimization of Lagrangean relaxations with feasibility recovering techniques is standard practice in exact algorithms design; the main difference between these established techniques and the approaches introduced in this subsection is in the focus on primal feasible solution update that characterizes heuristic methods.

In the following we will outline three algorithms: bionomic algorithms, scatter search techniques and ANTS procedures.

21.2.3.1 Bionomic algorithms. Bionomic Algorithms (BA), proposed by Christofides [10], follow the GA framework, but replace much of the randomness of the GA operators with more normative procedures. Moreover, BA accept

multiple parent combinations and variable cardinality solution sets, which are both elements that enhance the algorithm performance on CO problems. While the actual structure of a BA cannot be described in few steps, the general framework is the following:

1. Initialize the set R of solutions.
2. Local optimization of each solution in R .
3. Define a suitable *adjacency graph* over R and use it to define parent sets.
4. Use a randomized, many to one mapping to generate solutions from parent sets.
5. Update R with the solutions found at step 4.
6. If not(end condition) go to step 2.

The essential elements, with respect to GA, are the adjacency graph for the definition of the parent sets, the structural use of local search and the room provided for tailored recombination operators. Thus, besides local search, we have variable cardinality solution sets and multiple parent recombination.

21.2.3.2 Scatter search techniques. Scatter Search (SS) is a technique originally presented by Glover in [24], but which has only recently been applied to CO problems. Two essential papers present it: a more normative one in [26] and a more general one in [27]. By making reference to [26], SS is a technique which combines polyhedral cutting planes approaches with primal solutions recombination and update. The general framework for an integer problem, IP, whose linear relaxation is LP, is the following:

1. Initialize the Reference Set R with feasible IP solutions.
2. Generate a set H of extreme points adjacent to the optimal LP solution.
3. Define parent sets as subsets containing both elements of H and of R .
4. Construct a set Q of seed solutions with points obtained from linear combination of points in the parent sets.
5. Carry the solutions in Q to their corresponding local optima.
6. Update R with the best solutions in Q obtained at step 5.
7. Find a violated valid inequality for problem IP. If no violated inequality can be identified or the maximum number of iterations is reached then STOP, otherwise add the inequality to LP and go to step 2.

Again, we find variable cardinality solution sets and multiple parent recombination. The difference with BA is that SS is much more oriented to linear recombinations of parent sets to generate new solutions and it is deeply interleaved with the cutting plane process. The algorithms thus evolves at the same time a lower and an upper bound of the problem to solve: a characteristic shared only with ANTS but of great computational impact.

21.2.3.3 ANTS procedures. Finally, ANTS [33] is an extension of AS presented in Subsection 21.2.2.5, prescribing the use of lower bounds in order to determine the *a priori* effectiveness of a move and a modified update procedure for the determination of its *a posteriori* effectiveness. Specifically, to compute $\eta_j, j = 1, \dots, n$, it is suggested to use a lower bound to the cost of completing the current partial solution and to update $\tau_j, j = 1, \dots, n$, it is suggested a moving average on the last solutions produced. These modifications do not affect the general structure of the algorithm of Subsection 21.2.2.5, but have a significant impact on its effectiveness on CO problems.

These issues will be described in detail in Section 21.4.

21.3 ANT COLONY OPTIMIZATION

Ant Colony Optimization was initially proposed by Colomni et al. [11, 15]. The main underlying idea was that of parallelizing search over several constructive computational threads, all based on a dynamic memory structure incorporating information on the effectiveness of previously obtained results and in which the behavior of each single agent is inspired by the behavior of real ants. The importance of this original *AntSystem*, AS, resides mainly in being the prototype of a number of ant algorithms which have found interesting and successful applications.

An *ant* is defined to be a simple computational agent, which iteratively constructs a solution for the problem to solve. Partial problem solutions are seen as *states*; each ant *moves* from a state i to another one ψ , corresponding to a more complete partial solution. At each step σ , each ant k computes a set $A_k^\sigma(i)$ of feasible expansions to its current state, and moves to one of these in probability, according to a probability distribution specified as follows.

For ant k , the probability $p_{i\psi}^k$ of moving from state i to state ψ depends on the combination of two values:

- [1] the attractiveness η of the move, as computed by some heuristic indicating the *a priori* desirability of that move; and
- [2] the trail level τ of the move, indicating how proficient it has been in the past to make that particular move: it represents therefore an *a posteriori* indication of the desirability of that move.

Trails are *updated* at each iteration, increasing the level of those that facilitate moves that were part of “good” solutions, while decreasing all others. The specific formula for defining the probability distribution at each move makes use of a set $tabu_k$ which indicates a problem-dependent set of infeasible moves for ant k . Different authors use different formulae, according to [33] probabilities are computed as follows: $p_{i\psi}^k$ is equal to 0 for all moves which are infeasible (i.e., they are in the tabu list), otherwise it is computed by means of the following formula (21.1), where α is a user-defined parameter ($0 \leq \alpha \leq 1$):

$$p_{i\psi}^k = \frac{\alpha \cdot \tau_{i\psi} + (1 - \alpha) \cdot \eta_{i\psi}}{\sum_{(\iota\nu) \notin tabu_k} (\alpha \cdot \tau_{\iota\nu} + (1 - \alpha) \cdot \eta_{\iota\nu})}. \quad (21.1)$$

Parameter α defines the relative importance of the trail with respect to attractiveness. After each iteration t of the algorithm, that is when all ants have completed a solution, trails are updated following formula (21.2).

$$\tau_{i\psi}(t) = \rho\tau_{i\psi}(t - 1) + \Delta\tau_{i\psi} \quad (21.2)$$

where ρ is a user-defined coefficient and $\Delta\tau_{i\psi}$ represents the sum of the contributions of all ants that used move $(i\psi)$ to construct their solution. The ants' contributions are proportional to the quality of the achieved solutions, i.e., the better an ant solution, the higher will be the trail contribution added to the moves it used. The general structure of an ACO algorithm is as follows:

Step 1. (Initialization)

 Initialize $\tau_{i\psi}, \forall(i, \psi)$.

Step 2. (Construction)

 For each ant k do

 repeat

 compute $\eta_{i\psi}, \forall(i, \psi)$.

 choose in probability the state to move into.

 append the chosen move to the k -th ants set $tabu_k$.

 until ant k has completed its solution.

 carry the solution to its local optimum.

 enddo.

Step 3. (Trail update)

 For each ant move $(i\psi)$ do

 compute $\Delta\tau_{i\psi}$.

 update the trail matrix.

 enddo.

Step 4. (Terminating condition)

 If not(end condition) go to step 2.

Figure 21.1 Pseudo code of ACO general structure.

This general framework has been specified in different ways by the authors working on the ACO approach. This variety was well represented in the ANTS series of international workshops (ANTS'98 and ANTS'2000), which are conferences entirely devoted to algorithms inspired by the observation of ants behavior. Different applications have been presented: from plan merging to routing problems, from driver scheduling to search space sharing, from set covering to nurse scheduling, from graph coloring to dynamic multiple criteria balancing problems.

These applications add to those already presented, which will be briefly overviewed in the following. Table 21.1 presents a summary of the main ACO metaheuristics so far published. The first column of the table shows the name given to the metaheuristic, when available, the second columns the authors

who followed that approach and the third the problems it has been applied to (where TSP stands for traveling salesman problem, QAP for quadratic assignment problem, JSP for job shop scheduling problem, VRP for vehicle routing problem, SOP for sequential ordering problem, FAP for frequency assignment problem, GCP for graph coloring problem, SCS for shortest common supersequence).

The variety of the contributions testifies both the flexibility of the approach and the infancy of the field, where no evidence of the superiority of a particular technique over the other has so far emerged. In Section 21.4 we will concentrate on a particular one, ANTS, to provide more insight into the computational elements of a specific implementation, while in the following of this section we will present more elements about the different ACO contributions.

ABC	Bonabeau et al. [4], van der Put and Rothkrantz [55]	network routing
ACS	Dorigo and Gambardella [17]	TSP, VRP
AntNet	Di Caro and Dorigo [7, 8]	network routing
ANTS	Maniezzo [33], Maniezzo and Carbonaro [34]	QAP, FAP
AS	Colorni et al. [11, 12, 14, 18]	TSP, QAP, JSP
ASrank	Bullnheimer et al. [6]	TSP, VRP
HAS	Gambardella et al. [20]	QAP, VRP, SOP
MMAS	Stuetzle and Hoos [47, 49]	TSP, QAP
AS-SCS	Michel and Middendorf [42]	SCS
-	Costa and Hertz [13]	GCP
-	Merkle et al. [40, 41]	Scheduling problems
-	Gambardella and Dorigo [22]	SOP
-	Kawamura et al. [30]	TSP

Table 21.1 ACO applications (adapted from [16]).

21.3.1 ACO approaches to TSP

The first application of AS used the traveling salesman problem (TSP) as a benchmark problem. This was because TSP is one of the most studied NP-hard problem, and the ant metaphor is easily adapted to it. Several authors built upon this initial contribution.

Stuetzle and Hoos [48] introduced Max-Min Ant System (*MMAS*), a modification of Ant System applied to the TSP. These authors explicitly introduced in the algorithm two parameters, a maximum and minimum trail levels, whose values are chosen in a problem-dependent way in order to restrict possible trail values to the interval $[\tau_{min}, \tau_{max}]$. Moreover, MMAS controls the trail levels (initialized to their maximum value τ_{max}), only allowing the best ant at each iteration to update trails, thus providing a feedback on its results. Trails that do not receive any or very rare reinforcements will continuously lower their strength and will be selected more and more rarely by the ants, until they reach the τ_{min} value. The τ_{min} and τ_{max} parameters are used to counteract premature stagnation of search, maintaining at the same time some kind of elitist strategy. When applied to TSP, MMAS performs better than AS.

Bullnheimer et al. [6] proposed yet another modification of AS, called AS_{rank} , introducing a rank-based version of the probability distribution to limit the danger of over-emphasized trails caused by many ants using sub-optimal solutions. The idea is the following. At each iteration, when all solutions are completed the ants are sorted by solution quality (that is, tour lengths in the case of the TSP) and the contribution of an ant to the trail level update is weighted according to the rank of the ant, considering only the ω best ants. In Table 21.2 taken from [6], we compare against each other a simulated annealing (SA), a simulated annealing with the nearest neighbour heuristic (SA_{NN}), a genetic algorithm (GA), an Ant System (AS), an Ant System with elitist strategy (AS_{elite}) and an ant system with elitist strategy and ranking (AS_{rank}). The table reports the percentage deviation from optimality of the average results obtained on five different TSP instances (considering also real-life problems from an industrial application), the percentage deviation of the best results, and the percentage deviation of the worst results.

	<i>Deviation avg. sol.</i>	<i>Deviation best sol.</i>	<i>Deviation worst sol.</i>
<i>SA</i>	2.784	0.550	7.416
<i>SA_{NN}</i>	2.036	0.148	4.902
<i>GA</i>	2.112	0.614	4.252
<i>AS</i>	1.886	0.652	2.802
<i>AS_{elite}</i>	1.112	0.200	2.866
<i>AS_{rank}</i>	1.012	0.100	2.212

Table 21.2 Comparison of different ACO approaches on the TSP [6].

In general, the Ant System can compete with the other two meta-heuristics; for large problems AS_{rank} outperforms the other methods regarding average and especially worst case behaviour.

Gambardella and Dorigo [21] merged *AS* and Q-learning, a well known reinforcement learning algorithm from artificial intelligence, into an algorithm called Ant-Q. The idea was to update trails with values which predicted the quality of solution using the edges to which the trails were associated. Even though showing a good performance, Ant-Q was abandoned for the equally good but simpler Ant Colony System (*ACS*) algorithm [17], that uses a constant value instead of the mentioned prediction term. In this algorithm the trail values are added offline, at the end of each iteration of the algorithm, only to the arcs belonging to the best tour from the beginning of the search process, while ants perform online step-by-step trail updates to favor the emergence of solutions other than the best so far. Each ant uses a *pseudo-random proportional rule* to choose the next node to move to. This is a decision rule based on a $q_0 \in [0, 1]$ parameter that permits to modulate the exploration behaviour, concentrating the system activity either on the best solutions or on the entire search space. *ACS* also uses a data structure associated to vertices called

candidate list which provides additional local heuristic information. The candidate list associated with a vertex contains only the cl vertices nearest to the one in subject, and the ants choose the next move scanning the candidate list instead of examining all the unvisited neighbouring vertices.

21.3.2 ACO approaches to QAP

The quadratic assignment problem (*QAP*) is the problem of assigning n facilities to n locations so that a quadratic assignment cost is minimized. *QAP* can be considered one of the hardest CO problems, and can be solved to optimality only for comparatively small instances. For this reason, *QAP* was chosen as a second benchmark for AS, resulting in code AS-QAP [38]. AS-QAP was of limited effectiveness, but was the first evidence of the robustness of AS.

Recent extensions make use of a well-tuned local optimizer, obtaining very good results [37]. In fact, while the process of an individual ant will almost always converge very quickly to a possibly mediocre solution, the interaction of many feedback processes can instead lead to convergence towards a region of the space containing good solutions, so that a very good solution can be found (without however being stuck on it). In other words, the ant population does not converge on a single solution, but on a set of (good) solutions; the ants continue their search to further improve the best solution found. The results obtained showed the Ant System's competitive performance on several test problems. Further developments lead to the design of ANTS, which will be detailed in Section 21.4.

Also several other systems previously introduced were adapted to the *QAP*. For example, two efficient techniques are the MMAS-QAP algorithm [49] and HAS-QAP [20]. The latter, whose name stands for Hybrid Ant System for *QAP*, is an algorithm which interleaves an ant colony algorithm with a simple local search. Comparisons with some of the best heuristics for the *QAP* have shown that HAS-QAP is among the best as far as real world, irregular, and structured problems are concerned. On the other hand, on random, regular and unstructured problems the performance resulted to be less competitive.

21.3.3 ACO approaches to VRP

Vehicle routing problems (*VRPs*) are a CO problems in which a set of vehicles stationed at a depot has to serve a set of customers before returning to the depot, minimizing the number of vehicles used and the total distance traveled by the vehicles. Capacity constraints are imposed on vehicle trips, plus possibly a number other constraints coming from real-world applications, such as time windows, backhauling, rear loading, vehicle objections, maximum tour length, etc. Also the *VRP* can be considered as a generalization of the *TSP*, in fact the *VRP* reduces to the *TSP* when only one vehicle is available. Some of the most successful applications of ACO metaheuristics to *VRP* are the following.

A direct extension of *AS* based on the AS_{rank} algorithm is *AS-VRP*, an algorithm designed by Bullnheimer et al. [6, 5]. They used various standard

heuristics to improve the quality of *VRP* solutions and modified the construction of the tabu list considering constraints on the maximum total tour length of a vehicle and its capacity. The results obtained on some problem instances were sufficiently interesting to justify a more detailed study.

	R1		C1		RC1		R2		C2		RC2	
	vehic.	dist.	vehic.	dist.	vehic.	dist.	vehic.	dist.	vehic.	dist.	vehic.	dist.
MACS	12.00	1217.73	10.00	828.38	11.63	1382.42	2.73	967.75	3.00	589.86	3.25	1129.19
RT	12.25	1208.50	10.00	828.38	11.88	1377.39	2.91	961.72	3.00	589.86	3.38	1119.59
TB	12.17	1209.35	10.00	828.38	11.50	1389.22	2.82	980.27	3.00	589.86	3.38	1117.44
CR	12.42	1289.95	10.00	885.86	12.38	1455.82	2.91	1135.14	3.00	658.88	3.38	1361.14
PB	12.58	1296.80	10.00	838.01	12.13	1446.20	3.00	1117.70	3.00	589.93	3.38	1360.57
TH	12.33	1238.00	10.00	832.00	12.00	1284.00	3.00	1005.00	3.00	650.00	3.38	1229.00

Table 21.3 HAS-VRP results [23].

Also Gambardella et al. [23] faced the VRP, adapting the *ACS* approach to define MACS-VRPTW, and considering the time window extension to *VRP* which introduces a time range within which a customer must be serviced. This approach has proved to be competitive with the best known approaches in the literature, as testified by Table 21.3, in which six different problem types (R1 and R2 with randomly distributed vertices, C1 and C2 with clustered vertices, and RC1 and RC2 with random-clustered vertices, from [46]) have been used to compare MACS-VRPTW with five other heuristics: the adaptive memory programming method of Rochat and Taillard (RT), and Taillard et al. (TB) [45, 51], the method of Chiang and Russel (CR) [9], the GA of Potvin and Bengio (PB) [44] and the method of Thangiah et al. (TH) [52]. The columns show, for each problem type, the number of vehicles (vehic.) and the global cost (dist.) of the solution obtained by each heuristic.

21.4 THE ANTS ALGORITHM

ANTS is an extension of the Ant System proposed in [11], which specifies some underdefined elements of the general algorithm, such as the attractiveness function to use or the initialization of the trail distribution. This turns out to be variations of the general ACO framework that make the resulting algorithm similar in structure to tree search algorithms. In fact, the essential trait which distinguishes ANTS from a tree search algorithm is the lack of a complete backtracking mechanism, which is substituted by a probabilistic (*Non-deterministic*) choice of the state to move to and by an incomplete (*Approximate*) exploration of the search tree: this is the rationale behind the name ANTS, which is an acronym of *Approximated Non-deterministic Tree Search*. In the following we will outline two distinctive elements of the ANTS algorithm within the ACO framework, namely the attractiveness function and the trail updating mechanism.

21.4.1 Attractiveness

The attractiveness of a move can be effectively estimated by means of lower bounds (upper bounds in case of maximization problems) to the cost of the completion of a partial solution. In fact, if a state ι corresponds to a partial problem solution it is possible to compute a lower bound to the cost of a complete solution containing ι . Therefore, for each feasible move $(\iota\psi)$, it is possible to compute the lower bound to the cost of a complete solution containing ψ : the lower the bound the better the move. Since large part of research in CO is devoted to the identification of tight lower bounds for the different problems of interest, good lower bounds are usually available. Their use has several advantages, some of which are listed in the following [33]:

- A tight bound gives strong indications on the opportunity of a move.
- When the bound value becomes greater than the current upper bound, it is obvious that the considered move leads to a partial solution which cannot possibly be completed in a solution better than the current best one. The move can therefore be discarded from further analysis.
- If the bound is derived from linear programming (LP) and dual cost information is therefore available, it is possible to compute reduced costs for the problem decision variables, which in turn - when compared to an upper bound to the optimal problem solution cost - permit to a priori eliminate some variables. This results in a reduction to the number of possible moves, therefore to a reduction of the search space.
- A further advantage of LP lower bound is that the primal values of the decision variables, as appearing in the bound solution, can be used as an estimate of the probability with which each variable will appear in good solutions. This provides an effective way for initializing the trail values, thus eliminating the need for the user-defined parameters.

The use of LP bounds is a very effective and straightforward general policy, whenever tight such bounds have been identified for the problem to solve.

21.4.2 Trail update

A good trail updating mechanism avoids stagnation, the undesirable situation in which all ants repeatedly construct the same solutions, making impossible any further exploration in the search process. This derives from an excessive trail level on the moves of one solution, and can be observed in advanced phases of the search process, if parameters are not well tuned to the problem.

The procedure evaluates each solution against the last k ones globally constructed by ANTS. As soon as k solutions are available, we compute their moving average \bar{z} ; each new solution z_{curr} is compared with \bar{z} (and then used to compute the new moving average value). If z_{curr} is lower than \bar{z} , the trail

level of the last solution's moves is increased, otherwise it is decreased. Formula (21.3) specifies how this is implemented:

$$\Delta\tau_{\iota\psi} = \tau_0 \cdot \left(1 - \frac{z_{curr} - LB}{\bar{z} - LB}\right) \quad (21.3)$$

where \bar{z} is the average of the last k solutions and LB is a lower bound to the optimal problem solution cost. The use of a dynamic scaling procedure permits to discriminate small achievement in the latest stage of search, while avoiding to focalize search only around good achievement in the earliest stages.

Based on the described elements, the ANTS metaheuristic is the following:

Step 1. (Initialization)

Compute a (linear) lower bound LB to the problem to solve.

Initialize $\tau_{\iota\psi}$, $\forall(\iota, \psi)$ with the primal variable values..

Step 2. (Construction)

For each ant k **do**

repeat

compute $\eta_{\iota\psi}$, $\forall(\iota, \psi)$, as a lower bound to the cost of a complete solution containing ψ .

choose the state to move into, with probability given by (1).

append the chosen move to the k -th ant's set $tabu_k$.

until ant k has completed its solution.

carry the solution to its local optimum.

enddo.

Step 3. (Trail update)

For each ant move $(\iota\psi)$ **do**

compute $\Delta\tau_{\iota\psi}$.

update the trail matrix by means of (2) and (3).

enddo.

Step 4. (Terminating condition)

If not(end-test) go to step 2.

Figure 21.2 Pseudo code for the ANTS algorithm.

It can be noticed that the general structure of the ANTS algorithm is closely akin to that of a standard tree-search procedure. At each stage we have in fact a partial solution which is expanded by branching on all possible offspring; a bound is then computed for each offspring, possibly fathoming dominated ones, and the current partial solution is selected among that associated to the surviving offspring on the basis of lower bound considerations. By simply adding backtracking and eliminating the Montecarlo choice of the node to move to, we revert to a standard branch and bound procedure. An ANTS code can therefore be easily turned into an exact procedure.

21.5 COMPUTATIONAL RESULTS

The above elements have been implemented and tested both on well known CO problems, such as the quadratic assignment and the frequency assignment

problems (QAP and FAP, respectively) and on CO problems arising from actual practice, such as car pooling [36] or data warehouse conceptual design [35]. In the following we present some results obtained on QAP and FAP. It is important to notice that the reason behind the choice of these problems is in the attempt to empirically evaluate the robustness of the ANTS approach. While in fact preliminary non-optimized codes testified the validity of the issues reported in Section 21.4, i.e., the effectiveness of the ANTS procedure when tight bounds are available, both QAP and FAP were chosen because of the ineffectiveness of the bounds so far presented in the literature. A good performance on these problems a fortiori suggests the efficiency of the approach in the general case.

21.5.1 QAP

The quadratic assignment problem (QAP) is one of the best known and most difficult CO problems, as it is testified by the comparatively small gap existing between the dimension of the problems that can be solved to optimality by means of complete enumeration and the dimension of the problems that can be solved by means of the most advanced exact methods proposed in the literature.

The ANTS algorithm makes use of a lower bound derived from the well-known Gilmore and Lawler bound. Details are provided in [33], where both the ANTS and the derived exact procedure are described. Computational results for the heuristic part were given on all QAPLIB problem instances of dimension up to $n = 40$ and presented for ANTS and for two state-of-the-art heuristic procedures: Li et al.'s GRASP [31] and Taillard's robust tabu search (TS) [50]. Table 21.4 shows the obtained results. It shows the percentual deviation from the optimal value, or from the best known solution, of the best solution obtained over five runs by ANTS and by two of the most effective heuristics for the QAP: Robust Tabu Search and GRASP.

It is interesting to see how, even in the presence of a bad bound at the root node, the nondeterministic strategy followed by ANTS permits to quickly identify good solutions.

21.5.2 FAP

The frequency assignment problem is the problem that arises when a region is covered, for wireless communications, by cells centered on base stations and transmitters scattered around the region want to connect with the antennas of the base stations. Each connection, or link, between a transmitter and a base station can be made on a frequency supported by the antenna. However, the frequency concurrently operated by overlapping cells must be separated in order to minimize the interference on the communications taking place in the cells.

The current state of development of the research on FAP does not provide efficient lower bounds. We developed a lower bound which is not very tight but is efficient to compute [39], and included it in the ANTS algorithm. Computational results were obtained on three well-known problem datasets from

	TS	GRASP	ANTS		TS	GRASP	ANTS
CHR20A	0.06	1.48	0.00	NUG30	0.00	0.39	0.00
CHR20B	2.06	4.65	0.00	TAI30A	0.16	1.53	0.13
ROU20	0.00	0.00	0.00	TAI30B	0.00	0.11	0.00
TAI20A	0.00	0.19	0.00	THO30	0.00	0.15	0.00
CHR22A	0.00	1.15	0.00	ESC32A	0.00	2.77	0.00
CHR22B	0.71	2.03	0.00	ESC32B	0.00	0.00	0.00
CHR25A	4.48	4.64	0.76	ESC32H	0.00	0.00	0.00
TAI25A	0.12	0.79	0.00	MC33	0.12	0.07	0.00
TAI25B	0.00	0.00	0.00	TAI35A	0.65	1.80	0.32
BUR26A	0.00	0.00	0.00	TAI35B	0.00	0.23	0.00
BUR26B	0.00	0.00	0.00	STE36A	0.00	1.76	0.00
BUR26E	0.00	0.00	0.00	STE36B	0.00	1.44	0.00
BUR26F	0.00	0.00	0.00	STE36C	0.00	0.67	0.00
KRA30A	0.00	0.34	0.00	LIPA40A	0.00	1.11	0.00
KRA30B	0.00	0.15	0.00	TAI40A	0.93	2.06	0.47
LIPA30A	0.00	0.19	0.00	TAI40B	0.00	0.03	0.00
				THO40	0.06	0.91	0.03

Table 21.4 ANTS on QAP.

the literature, obtaining the results presented in Table 21.5, whose columns show the following. CELAR, GRAPH, and PHILADELPHIA are benchmark datasets, as detailed below. For each problem, we present the cost of the best solution found by: a problem-specific heuristic (DS, an adaptation of DSATUR originally conceived for graph coloring), a tabu search algorithm (TS), ANTS, and two versions of simulated annealing (SA1 and SA2). More details can be found in [34]. In this subsection we report the computational results obtained on a number of different test problems drawn from literature: the CELAR, GRAPH, and PHILADELPHIA problems. The CELAR dataset consists of 11 problems proposed within the framework of EUCLID (European Cooperation for the Long term in Defence) CALMA (Combinatorial ALgorithms for Military Applications) project [53]. The GRAPH test problems [54] are 14 problems patterned after the CELAR problems which exhibit the same structure. The PHILADELPHIA problems, originally presented by Anderson [3], are among the most studied FAP instances. The problems are based on the area around Philadelphia and consist of cells located in a hexagonal grid. All results have been obtained implementing the algorithms in C and running the codes on a Pentium II 233 MHz machine equipped with 64 Mb of RAM for 1200 CPU seconds. Ongoing, as yet unpublished, research demonstrates however that it is possible to further improve the results of Table 21.5, demonstrating that ANTS has an effectiveness comparable with that of the state of the art heuristic approaches for the FAP.

The computational results show that the ANTS heuristic is competitive with the other approaches used for comparison. In particular, Table 21.5 shows that different problem types are best solved by different algorithms. CELAR problems are best solved by the ANTS and the SA1 algorithms. GRAPH

DS	TS	CELAR			GRAPH					PHILADELPHIA				
		ANTS	SA1	SA2	DS	TS	ANTS	SA1	SA2	DS	TS	ANTS	SA1	SA2
625	238	0	0	0	160	15	0	0	0	7	4	0	51	263
428	122	0	0	0	299	15	0	0	0	0	0	0	17	251
798	196	0	0	0	264	35	14	14	96	8	8	0	31	288
1474	012	8	0	1	519	33	42	64	213	6	6	0	36	353
1823	688	32	11	54	0	0	0	0	0	3	3	30	31	252
213866	149607	5319	6 994	30 160	0	0	0	0	0	18	17	36	30	288
$> 10^8$	$> 10^8$	8083093	11000296	4698907	0	0	0	0	0	0	0	29	30	271
2468	1364	709	306	457	566	18	0	0	0	17	14	31	22	249
79406	4988	16732	30024	23634	665	14	0	0	0	0	0	403	32	256
107310	58554	31516	31518	33557	856	37	127	91	81	1	1	51	18	254
1364	848	0	0	2	0	0	0	0	0	0	0	0	0	0
					750	19	0	0	0					

Table 21.5 ANTS on FAP [34].

problems have the same structure as the CELAR problems, and in fact the results obtained are similar among the two groups, while the PHILADELPHIA problems are more suited for the DSATUR and TS approaches. On all test problems, under the imposed computational constraints (1200 seconds of CPU time), it found a good solution and exhibited more stable results among those produced by the tested algorithms.

21.6 CONCLUSIONS

There is a natural evolution in the life of algorithms that attract the attention of the research community. At the beginning the need to face previously unsolvable problems, or the attractiveness of the motivating idea justifies interest in the approach, despite possible second-rate results. If the core idea is worthwhile, enduring research improves the results obtained up to the limit of the state-of-the-art, and possibly beyond. This is what we witnessed for local search procedures, for simulated annealing, for genetic algorithms, etc. It seems that ACO algorithms are following the same track. After the early results on the robustness of the idea, but with little effectiveness to show, now we have moved into a maturity where ant-based algorithms should not be dismissed when high quality solutions are needed for difficult CO problems.

There is still no way to tell which among the different ACO approaches currently studied is the one that will yield better results. The area is still (already?) actively investigated, new ideas are poured into the general framework and effectiveness and robustness are under improvement. We have no means to forecast how ACO will compare with the most established metaheuristic approaches so far. What we can say is that ACO is not sterile, that ants have not dried out on the first maybe intriguing promises.

References

- [1] E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*. Wiley, 1989.
- [2] E. Aarts and J.K. Lenstra (editors). *Local Search in Combinatorial Optimization*. Wiley, 1997.
- [3] L.G. Anderson. A Simulation Study of Some Dynamic Channel Assignment Algorithms in a High Capacity Mobile Telecommunications System. *IEEE Transactions on Communications*, COM-21:1294–1301, 1973.
- [4] E. Bonabeau, F. Henaux, S. Guerin, D. Snyers, P. Kuntz, and G. Theraulaz. Routing in Telecommunication Networks with “Smart” Ant-Like Agents. *Lecture Notes on Artificial Intelligence*, 1437:60–68, 1998.
- [5] B. Bullnheimer, R.F. Hartl, and C. Strauss. An Improved Ant System Algorithm for the Vehicle Routing Problem. *Annals of Operations Research*, 89:319–328, 1999.
- [6] B. Bullnheimer, R.F. Hartl, and C. Strauss. A New Rank-Based Version of the Ant System: A Computational Study. *Journal for Operations Research and Economics*, 7:25–38, 1999.
- [7] G. Di Caro and M. Dorigo. Antnet: A Mobile Agents Approach to Adaptive Routing. Technical Report IRIDIA/97-12, Université Libre de Bruxelles, 1997.
- [8] G. Di Caro and M. Dorigo. Antnet: Distributed Stigmergetic Control for Communications Networks. *Journal of Artificial Intelligence Research*, 9:317–365, 1998.
- [9] W.C. Chiang and R. Russel. Hybrid Heuristics for the Vehicle Routing Problem with Time Windows. Technical Report, Department of Quantitative Methods, University of Tulsa, 1993.
- [10] N. Christofides. The Bionic Algorithm. Presented at the Annual Conference AIRO’94, Savona, 1994.
- [11] A. Colorni, M. Dorigo, and V. Maniezzo. Distributed Optimization by Ant Colonies. *Proceedings of the 1991 European Conference on Artificial Life*, pages 134–142, Elsevier, 1991.
- [12] A. Colorni, M. Dorigo, V. Maniezzo, and M. Trubian. Ant System for Job-Shop Scheduling. *Belgian Journal of Operation Research, Statistics and Computer Scence*, 34:39–54, 1994.
- [13] D. Costa and A. Hertz. Ants Can Colour Graphs. *Journal of the Operational Research Society*, 48:295–305, 1997.

- [14] M. Dorigo. *Optimization, Learning and Natural Algorithms*. Ph.D. Thesis, Politecnico di Milano, 1992.
- [15] M. Dorigo, A. Colorni, and V. Maniezzo. Positive Feedback as a Search Strategy. Technical Report TR91-016, Politecnico di Milano, 1991.
- [16] M. Dorigo and G. Di Caro. The Ant-Colony Optimization Meta-Heuristic. In: *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, editors, pages 11–32, McGraw-Hill, 1999.
- [17] M. Dorigo and L. M. Gambardella. Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Transactions on Evolutionary Computation*, 1:53–66, 1997.
- [18] M. Dorigo, V. Maniezzo, and A. Colorni. The Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, 26:29–41, 1996.
- [19] T.A. Feo and M.G.C. Resende. Greedy Randomized Adaptive Search Procedures. *Journal of Global Optimization*, 6:109–133, 1995.
- [20] L. M. Gambardella, E. Taillard, and M. Dorigo. Ant Colonies for the Quadratic Assignment Problem. *Journal of the Operational Research Society*, 50:167–176, 1999.
- [21] L.M. Gambardella and M. Dorigo. Ant-q: A Reinforcement Learning Approach to the Travelling Salesman Problem. *Proceedings of the Twelfth International Conference on Machine Learning*, Palo Alto, pages 252–260, Morgan Kaufmann, 1995.
- [22] L.M. Gambardella and M. Dorigo. An Ant Colony System Hybridized with a New Local Search for the Sequential Ordering Problem. *INFORMS Journal on Computing*, 12:237–255, 2000.
- [23] L.M. Gambardella, E. Taillard, and G. Agazzi. Ant Colonies for Vehicle Routing Problems. In: *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, editors, pages 63–76, McGraw-Hill, 1999.
- [24] F. Glover. Heuristics for Integer Programming Using Surrogate Constraints. *Decision Sciences*, 8:156–166, 1977.
- [25] F. Glover. Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers and Operations Research*, 13:533–549, 1986.
- [26] F. Glover. Scatter Search and Star Paths: Beyond the Genetic Metaphor. *OR Spektrum*, 17:125–137, 1995.
- [27] F. Glover. A Template for Scatter Search and Path Relinking. *Lecture Notes in Computer Science*, 13363:13–54, 1997.
- [28] F. Glover and M. Laguna. *Tabu Search*. Kluwer, 1997.

- [29] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.
- [30] H. Kawamura, M. Yamamoto, K. Suzuki, and A. Ohuchi. Multiple Ant Colonies Algorithm Based on Colony Level Interactions. *IEICE Transactions Fundamentals*, E83-A:371–379, 2000.
- [31] Y. Li, P.M. Pardalos, and M.G.C. Resende. A Greedy Randomized Adaptive Search Procedure for the Quadratic Assignment Problem. In: *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 16:237–261, 1994.
- [32] S. Lin and B.W. Kernighan. An Effective Heuristic Algorithm for the Travelling Salesman Problem. *Operations Research*, 31:498–516, 1973.
- [33] V. Maniezzo. Exact and Approximate Nondeterministic Tree-Search Procedures for the Quadratic Assignment Problem. *INFORMS Journal of Computing*, 11:358–369, 1999.
- [34] V. Maniezzo and A. Carbonaro. An Ants Heuristic for the Frequency Assignment Problem. *Future Generation Computer Systems*, 16:927–935, 2000.
- [35] V. Maniezzo, A. Carbonaro, M. Golfarelli, and S. Rizzi. An ANTS Algorithm for Optimizing the Materialization of Fragmented Views in Data Warehouses: Preliminary Results. *Lecture Notes in Computer Science*, 2037:80–89, 2001.
- [36] V. Maniezzo, A. Carbonaro, D. Vigo, H. and Hildmann. An ANTS Heuristic for the Long-Term Car Pooling Problem. In: *Proceedings of the Second International Workshop on Ant Algorithms*, pages 78–81, Brussels, 2000.
- [37] V. Maniezzo and A. Colorni. The Ant System Applied to the Quadratic Assignment Problem. *IEEE Transactions on Knowledge and Data Engineering*, 11:769–778, 1999.
- [38] V. Maniezzo, A. Colorni, and M. Dorigo. The Ant System Applied to the Quadratic Assignment Problem. Technical Report IRIDIA/94-28, Université Libre de Bruxelles, 1994.
- [39] V. Maniezzo and R. Montemanni. An Exact Algorithm for the Radio Link Frequency Assignment Problem. Technical Report CSR99-02, 1999.
- [40] D. Merkle and M. Middendorf. An Ant Algorithm with a New Pheromone Evaluation Rule for Total Tardiness Problems. *Lecture Notes in Computer Science*, 1803:287–296, 2000.

- [41] D. Merkle, M. Middendorf, and H. Schmeck. Ant Colony Optimization for Resource-Constrained Project Scheduling. In: *Proceedings of the 2000 Genetic and Evolutionary Computation Conference*, pages 893–900, Las Vegas, 2000.
- [42] R. Michel and M. Middendorf. An Island Model Based Ant System with Lookahead for the Shortest Supersequence Problem. *Lecture Notes in Computer Science*, 1498:692–700, 1998.
- [43] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization, Algorithms and Complexity*. Prentice-Hall, 1982.
- [44] J.Y. Potvin and S. Bengio. The Vehicle Routing Problem with Time Windows – Part II: Genetic Search. *INFORMS Journal of Computing*, 8:165–172, 1996.
- [45] Y. Rochat and E.D. Taillard. Probabilistic Diversification and Intensification in Local Search for Vehicle Routing. *Journal of Heuristics*, 1:147–167, 1995.
- [46] M. Solomon. Algorithms for the vehicle Routing and Scheduling Problem with Time Window Constraints. *Operations Research*, 35:254–265, 1987.
- [47] T. Stuetzle and M. Dorigo. ACO Algorithms for the Quadratic Assignment Problem. In: *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, editors, pages 33–50, McGraw-Hill, 1999.
- [48] T. Stuetzle and H. Hoos. Improvements on the Ant System: Introducing $\max - \min$ Ant System. In: *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms*, G.D. Smith, N.C. Steele and R.F. Albrecht, editors, pages 245–249, Springer Verlag, 1998.
- [49] T. Stuetzle and H. Hoos. The $\max - \min$ ant System and Local Search for Combinatorial Optimization Problems: Towards Adaptive Tools for Combinatorial Global Optimization. In: *Meta-Heuristics: Advanced and Trends in Local Search Paradigms for Optimization*, S. Voss, S. Martello, I.H. Osman, and C. Roucairol, editors, pages 313–329, Kluwer, 1998.
- [50] E. Taillard. Robust Taboo Search for the Quadratic Assignment Problem. *Parallel Computing*, 17:443–455, 1991.
- [51] E.D. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J.Y. Potvin. A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows. *Transportation Science*, 31:170–186, 1997.
- [52] S.R. Thangiah, I.H. Osman, and T. Sun. Hybrid Genetic Algorithm Simulated Annealing and Tabu Search Methods for Vehicle Routing Problem with Time Windows. Technical Report 27, Computer Science Department, Slippery Rock University, 1994.

- [53] S. Tiourine, C. Hurkens, and J.K. Lenstra. An Overview of Algorithmic Approaches to Frequency Assignment Problem. Technical Report, T.U. Eindhoven, 1995.
- [54] H.P. van Benthem. *Graph: Generating Radio Link Frequency Assignment Problems Heuristically*. Master's Thesis, Faculty of Technical Mathematics and Informatics, Technical University of Delft, 1995.
- [55] R. van der Put. Routing in Packet Switched Networks Using Agents. Technical Report RD-SV-98-276, KPN Research, 1998.

22 INTENSIFICATION NEIGHBORHOODS FOR LOCAL SEARCH METHODS

Thierry Mautor

Laboratoire PRISM
Université de Versailles - Saint Quentin
45 Avenue des Etats Unis
78035 Versailles Cedex, France
Thierry.Mautor@prism.uvsq.fr

Abstract: In local search methods, even when they are used in sophisticated implementations of metaheuristics, basic neighborhoods are generally applied. The best neighbor can be quickly identified but, on the other hand, these neighborhoods give to the search a low visibility of the global search space to the detriment of the efficiency of the method. That is the first reason why we present in this paper new neighborhood structures, namely the neighborhood tree and the Mimausa neighborhood. Used in a local search method, these structures intensify the search around the current solution and may ensure a quicker convergence. These neighborhood structures could also be of interest for a Variable Neighborhood Search or Descent where a set of various neighborhood structures is required.

22.1 INTRODUCTION

The past few years, meta-heuristics based on neighborhood search, such as Tabu Search, Simulated Annealing, GRASP, etc. have been widely investigated and applied to a large spectrum of Combinatorial Optimization applications. These meta-heuristics can be seen as extensions of the classical local search method (sometimes called descent method) in which a sequence of local and basic changes, that improve each time the value of the objective function, are iteratively performed from an initial solution to a local optimum. However, the scheme of the local search method is generally kept and the local changes applied to the current solution are mainly basic, such as “2-exchanges” (the values of two variables are exchanged) or “Add/Drop”. It is true that for

specific applications, the neighborhood structure needs to be a little more sophisticated. For this type of applications, more complex neighborhoods, such as the ejection chains, have been defined and studied (see for example [15]). These cases, however, remain exceptional.

The numerous applications of the Tabu Search method have clearly shown that intensification and diversification strategies are often essential to make the method really efficient. Intensification strategies consist in guiding the search towards attractive and insufficiently explored regions, while diversification strategies try to locate new, not yet explored regions. Nevertheless, as suggested by the Variable Neighborhood Search method, recently proposed by Hansen and Mladenovic [8], these strategies could be efficiently performed by varying the neighborhood nature during the search.

From this point of view, it appears to be interesting to define new neighborhood structures which can be applied to a variety of combinatorial optimization problems (we could talk of *meta-neighborhoods*). Towards this direction, the neighborhood tree has been recently proposed and applied to the Job Shop Scheduling problem by Balas and Vazacopoulos [1]. Since this structure has certain similarities with the successive fan candidate list strategy [7], an unifying structure could be proposed. On the other hand, a method, called Mimausa, in which a coordinated series of subparts of the global problem is iteratively solved to optimality, has been introduced by Mautour and Michelon [10]. Even if Mimausa can be used as a global meta-heuristic, the ideas can also be integrated in a local search method where Mimausa defines a new neighborhood structure.

The remainder of the paper is organized as follows. In the next section, an analysis of the behaviour of a classical Tabu Search algorithm on a representative combinatorial optimization problem (the Quadratic Assignment Problem) illustrates why a new neighborhood structure could be of great interest. The two neighborhood structures are described in Sections 22.3 and 22.4. Although the focus of this paper is on basic ideas, some computational results are provided in Section 22.5.

22.2 WHY A NEW NEIGHBORHOOD STRUCTURE?

In this section, we focus on the assumption that basic neighborhoods have a low “visibility” of the search space, fact that could significantly deteriorate the efficiency of the local search methods. We chose to illustrate this effect on a representative application, by analysing the behaviour of a standard and efficient Tabu Search on a classical combinatorial optimization problem, the Quadratic Assignment Problem (QAP).

The “*Robust Tabu Search*” has been developed by Taillard in 1991 for the QAP [17]. This Tabu Search version is considered as one of the most efficient heuristic algorithms proposed for the problem. The code has been made available by the author on the “*QAPLIB*” web-site. Our purpose is not to describe this algorithm. We just mention that the neighborhood of the current solution is composed of the $n(n - 1)/2$ possible 2-exchanges.

One may define the distance between two solutions as the minimum number of basic moves required to go from the first solution to the second one. For the QAP and with a “2-exchanges” neighborhood, this distance corresponds to the minimum number of transpositions between two permutations whose computation is well known and easy. We denote this distance $d2ex$. For example: $d2ex((1, 2, 3, 4, 5, 6), (3, 2, 1, 4, 6, 5)) = 2$ (- 1 and 3 - and - 5 and 6 - need to be exchanged) and $d2ex((1, 2, 3, 4, 5, 6), (3, 1, 2, 5, 6, 4)) = 4$.

We have added to the *Robust Tabu Search* code the computation of the distance between each visited solution and the globally optimal solution (or the nearest optimal solution, if several optimal solutions exist). We can thus follow the distance to the optimal solution through the search. We count the number of times the search arrives to a short distance from the optimum and it then moves clearly far from it. This is called a “*missed approach*” and can be defined as follows.

Let us denote:

- $p(k)$, the solution obtained at iteration k ;
- $d2opt(p(k)) = \min_{o \in Opt} d2ex(p(k), o)$, the distance from the solution $p(k)$ to the nearest globally optimal solution;
- T_1 , a first threshold used to identify that the search is arrived to a short distance from an optimal solution; this threshold is thus set to a low value (for example 3); and
- T_2 , a second threshold used to identify that the search is rather far from an optimal solution; this threshold is thus set to a medium value (for example 6).

A missed approach occurs at iteration k if: $d2opt(p(k)) = T_1$ and $d2opt(p(k)) < d2opt(p(k-1))$ and $\exists k', k' > k, d2opt(p(k')) = T_2$.

The average number of these “missed approaches” for 100 runs of the *Robust Tabu Search* is given in Table 22.1 for two representative applications (Elshafei 19 is a “real-world” problem: service assignments in a Cairo hospital, with a high flow dominance [5] - Nugent 15 corresponds to the assignment on a grid of 15 units and the flow dominance is low [13]). The first threshold T_1 was set to three different values: 2, 3 and 4, while the second threshold was set to 6.

Instance	Elshafei	Nugent
size	19	15
T1 = 2	0.18	0.35
T1 = 3	2.51	0.96
T1 = 4	16.27	2.28

Table 22.1 Average number of missed approaches.

As it can be seen, the average number of missed approaches is largely significant. It appears to be relatively frequent that the search reaches a solution

whose structure is very close to an optimal solution without managing to reach this optimum. Let us mention that rarely the best move would converge but the application of an aspiration criterion used to diversify the search results to the choice of another move. In most cases, a missed approach occurs either because the best move that would bring the search to the good direction is unfortunately tabu and no aspiration criterion can be applied, or because the best move takes the search away from the optimum. Let us also mention that, in most cases (80%), a missed approach occurs while the selected move increases the current solution value.

As a consequence, it occurs frequently that the optimal solution is quickly approached but is finally reached late in the search. Figure 22.1 gives a coarse plotting of the distance $d2opt$ through the first run (among the 100 runs) of the Robust Tabu Search on the instance “Elshafei 19”. This run is representative of the algorithm behavior. An optimal solution is very quickly approached ($d2opt(p(51)) = 4$, $d2opt(p(635)) = 3$), but several times the search diverges and the optimal solution is finally reached at iteration 9530.

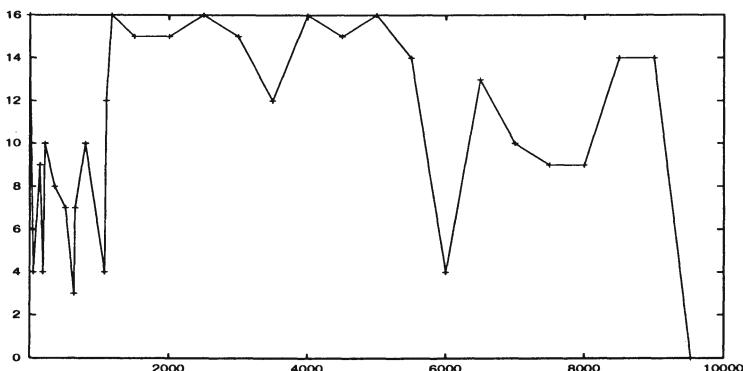


Figure 22.1 Distance to the optimal solution through the search.

One of the main objectives of the neighborhood structures presented in the next sections is to minimize these missed approaches and, thus, to make the search more efficient.

22.3 THE NEIGHBORHOOD TREE

The neighborhood tree has been recently defined and successfully applied to the Job Shop Scheduling problem by Balas and Vazacopoulos [1]. The main idea of this neighborhood structure is to study not only the direct neighbors (obtained with a basic move) of the current solution but also the neighbors of the best “direct” neighbors, and so on on few levels. This way, a neighborhood structure is built as illustrated in Figure 22.2. In this neighborhood structure, the root is the current solution, an arc corresponds to a basic move and a node is a feasible solution.

Due to the combinatorial explosion, a complete study of the whole set of neighbors on several levels cannot be considered and some reduction rules have to be defined. 3 types of reduction rules have been applied by Balas and Vazacopoulos:

- As previously defined, the neighborhood structure is not a tree but a graph, since a node can be reached by different paths (different sequences of basic moves). The first reduction consists in removing some arcs in order to be sure that the structure is a tree: the neighborhood tree. When building this tree, an arc which could create a cycle is forbidden. A short term memory, called fixed list, is defined. It stores, for each node, the forbidden moves: either a move that could lead back to an ancestor, or a transversal move that leads to a node which could be created in a leftmost branch. When a move is chosen to create a new node, the opposite move becomes forbidden and is added to the fixed list for this node and all its descendants. When, from a node, a move is chosen to create a child, this move becomes forbidden and added to the fixed list for all the next children of the node and their descendants (rightmost branches).
- The second type of reduction consists in limiting the size of the tree. Some parameters are introduced in order to bound the depth of the tree as well as its width. More precisely, the number of children generated by a node is bounded by a decreasing function depending on the level ("birth control"). Hence, $NT(K_1, K_2, \dots, K_D)$ will represent the neighborhood tree of depth D with K_1 nodes on the first level, where each of these K_1 nodes has K_2 sons on level 2 and so on (for instance $NT(5, 2, 1)$ is drawn on Figure 22.2).
- The last type of reduction rules applied by Balas and Vazacopoulos are specific to the Job Shop Scheduling problem. Some moves (2-exchanges)

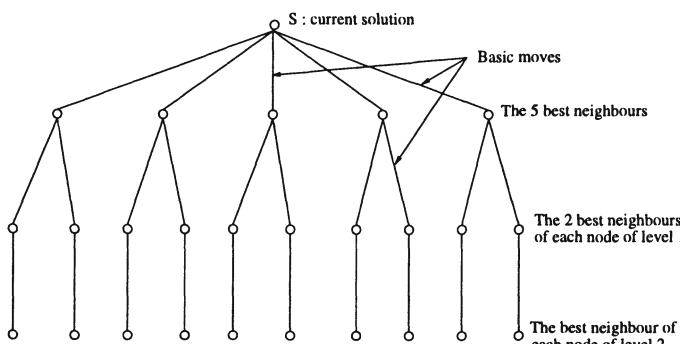


Figure 22.2 The neighborhood tree: $NT(5,2,1)$.

are proved to be unable to reduce the makespan and can be excluded from the list of candidates. The neighborhood size is thus strongly reduced.

Let us focus on the fact that Balas and Vazacopoulos use this neighborhood structure in a new search procedure, called Guided Local Search and do not integrate it in a Tabu Search (or Simulated Annealing) scheme. At each iteration, the root node of the neighborhood tree is the current solution. The corresponding neighborhood tree is built and the next solution is chosen in it. Either the neighborhood tree has produced a solution which improves the current best solution and the best solution in the tree is selected. Otherwise, to avoid cycling, a biased random scheme is applied: the selected solution is chosen among the k best solutions in the tree (as described by Balas and Vazacopoulos: “this biased random choice is the one that assigns a probability of $1/2^i$ to the i th best node”).

However, this neighborhood structure can be integrated in a Tabu Search scheme and links can be found with the “fan candidate list strategy” theoretically proposed by Glover (see [7] or [6]) where different Tabu streams are developed (perhaps in parallel) from the current solution, each of these streams owning its own tabu memory. These streams are initiated with the best moves from the current solution. The next current solution is the best one among the solutions finally developed by the different streams.

We propose a neighborhood structure that combines the main principles of the neighborhood tree and of the fan candidate list. This unifying data structure, applicable to different combinatorial optimization problems, can be drawn according to the following principles:

- The neighborhood tree general structure is kept - nevertheless, except for the first level, the parameter that bounds the number of sons of a node is set to an extremely low value in order to reinforce the similarities with the Tabu streams of the fan candidate strategy and also in order to make the computation of the structure not too expensive.
- Each node owns its own tabu memory (in addition to the fixed list) - this tabu memory is initialized with the current solution tabu memory, except in one specific branch where the tabu memory is cleared. The reason to develop such a specific branch is the observation made in Section 22.2 that a significant part of the missed approaches occurs because the best move would converge but is tabu. The development of this branch also corresponds to the generalization to the Neighborhood Tree of the well known aspiration criterion: “a tabu move can be performed if it improves the value of the best known solution”. In the other branches, the best moves which are both non tabu and not forbidden (not in the fixed list) are chosen to create the children of a node.
- The tabu memory prevents cycling and, consequently, the biased random scheme applied by Balas and Vazacopoulos to choose the next solution is not essential any more. It is true that biased random move selection

can have benefit beyond cycle avoidance. For instance, it may produce a better diversification of the search. However, we chose not to use the biased random move selection. The best solution in the Neighborhood Tree is chosen as the new current solution, but in the specific branch with a cleared tabu memory where the new current solution is chosen only if it improves the best known solution (classical aspiration criterion). Let us finally mention that one may also impose the new current solution to be deep enough in the neighborhood tree. This way, moves are more substantial and the moving in the search space is quicker.

22.4 MIMAUSA NEIGHBORHOOD

Mimausa has been proposed as a new metaheuristic by Mautor and Michelon in 1997 [10]. This method in which a coordinated series of subparts of the global problem is iteratively solved to optimality can be seen as a combination of exact solution and local search. More precisely, at each iteration of the method, the whole set of the variables is partitioned into two subsets, where the first one contains the free variables, the other variables being fixed to their current value. The exact method optimizes the subproblem induced by the free variables and the current solution is modified, according to the result of the optimization. Such a scheme that combines local and global search has been applied to different combinatorial optimization problems (see [14] for time tabling, [10] for quadratic assignment, [16] for vehicle routing, [4] for scheduling problem, [2] for network design).

The scheme of Mimausa algorithm (for a minimization problem) can be presented as follows:

```

Start with a feasible solution  $p$ .
 $BestVal := \text{Value}(p)$ ; (Best known value).
 $K := Kinit$ ;
 $Niter := 0$ ;
Repeat
  Selection of  $K$  variables (set  $S_1$ ).
  Construction of the reduced problem (size  $K$ ).
   $Solu := \text{Exact Solution}$  of the reduced problem.
  Construction of the new solution:
     $\forall i \in S_1, p(i) := Solu(i)$ ,
     $\forall i \notin S_1, p(i)$  does not change (fixed variable)
  Memorization of the best known value:
    if ( $\text{Value}(p) < BestVal$ )
      •  $BestVal := \text{Value}(p)$ ;
      •  $niter := 0$ ;
    else  $niter := niter + 1$ ;
  Adjustment of the value of  $K$ :
     $Kprev := K$ ;
    Modification (if necessary) of the value of  $K$ 
Until ( $Kprev = \text{ProbSize}$ ) or ( $niter = \text{MaxIter}$ )

```

The algorithm is rather simple. The method is iterative and, at each iteration of the method, K variables are selected. Each variable represents a part of the solution, for instance an assignment for the QAP, an arc for routing problems or the schedule of a task for scheduling problems. The $(N - K)$ other variables are fixed to the value they have in the current solution. The subproblem so induced is a reduced problem whose construction is generally easy and an exact method is applied to find its optimal solution. Then, the current solution is updated according to the solution of this optimization: the K “free” variables are assigned to the value they have in the optimal solution of the reduced problem while the other ones (the “fixed” variables) remain unchanged. The method stops either when the global problem has been solved to optimality during the previous iteration (in this case K_{prev} is equal to the problem size) or when the solution has not been improved since a given number of iterations.

The main difficulties of the method are the tuning of the parameters K and $MaxIter$ and the determination of the selection rule of the variables. K must not be set to extreme values, neither too high as the exact solution has to be obtained in a very short time, nor too low to optimize a substantial subpart of the problem. But, to determine a good value for this parameter is not such an easy exercice, since the computational time required by an exact method not only depends on the size of the subproblem but also on the form of each specific instance and on the value of the upper bound (best known solution). It is the reason why we propose in the algorithm that the value of K can vary through the search. For instance, K can be initialized to a very low value (K_{init}), incremented when the computational time required to solve exactly the reduced problem is low and decremented when this computational time becomes prohibitive. As well, K can be initialized to a supposed good value and never be modified.

Different simple selection rules can be envisaged: a complete random selection, a roulette wheel selection where the size of a sector depends on the number of times the corresponding variable has already been selected, a tabu selection where the selected variables cannot be selected again during a few number of iterations, a “2-exchange” selection where the variables belonging to the best 2-exchanges are selected.

Even if the algorithm, as presented above, defines a complete metaheuristic, let us remark that, at a given iteration, the new solution is chosen among all the solutions whose values of the unselected variables are equal to the values they have in the current solution:

$$MN = \{sol, \forall i \notin S_1 sol(i) = cs(i)\},$$

with cs = current solution.

This defines a new neighborhood structure, we call “Mimausa Neighborhood”. This neighborhood structure is generally wider than the one defined by a basic “2-exchanges”. For instance for the QAP, we have $K! - 1$ neighbors with this structure and $\frac{N(N-1)}{2}$ possible 2-exchanges. On the other hand, as illustrated on Figure 22.3, the moves in the search space can be more

substantial with the “Mimausa neighborhood” where the value of K variables can be modified.

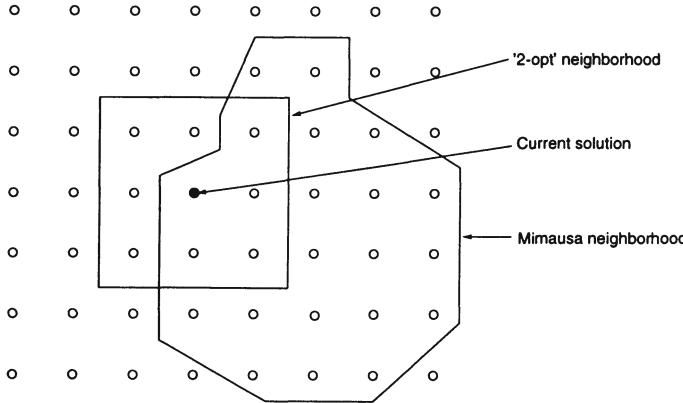


Figure 22.3 ‘2-opt’ and “Mimausa” neighborhoods.

Let us remark that such a neighborhood structure can be used either in an intensification phase of a Tabu Search algorithm or in a Variable Neighborhood Search method.

In the computational experiments presented in the next section for the QAP, mainly, the parameter K is considered as a constant that does not vary during the search. Hence, we denote as $MN(K)$ the “Mimausa Neighborhood” with a given value of the parameter K . Moreover, the K variables are randomly selected.

For the QAP, when K variables are selected and constitute the set S_1 , the $N - K$ other variables (set S_2) being fixed to their current value, the objective function has the following form:

$$\begin{aligned}
 & \text{Min} \sum_{i \in S_2} \sum_{j \in S_2} f_{ij} d_{p(i)p(j)} \\
 & \quad (\text{constant}) \\
 & + 2 \cdot \text{Min} \sum_{i \in S_1} \sum_{k \in p(S_1)} \left(\sum_{j \in S_2} f_{ij} d_{kp(j)} \right) x_{ik} \\
 & \quad (\text{linear assignment problem of size } k) \\
 & + \text{Min} \sum_{i \in S_1} \sum_{j \in S_1} \sum_{k \in p(S_1)} \sum_{l \in p(S_1)} d_{ij} f_{kl} x_{ik} x_{jl}.
 \end{aligned}$$

(quadratic assignment problem of size k)

This formulation leads to the construction of a reduced quadratic assignment problem of size K . The exact method used to solve this reduced problem is the Branch and Bound algorithm proposed in 1994 by Mautour and Roucairol [11].

	3-opt	NT(6-1)	4-opt	NT(6-2-1)	MN(7)
Nugent 15					
Comp.Time (s)	73	9	596	21	18
Avg value	1408.7	1368.2	1362.3	1308.4	1363.9
Elshafei 19					
Comp.Time (s)	241	15	2855	34	17
Avg value	34108377	31490585	30144298	28246947	30329972

Table 22.2 A comparison of three neighborhoods: K-opt, neighborhood tree, Mimausa neighborhood.

22.5 COMPUTATIONAL EXPERIMENTS

As previously mentioned, the focus of this paper is on basic ideas. Furthermore, the neighborhood structures are presented as general structures which can be applied to a variety of combinatorial optimization problems and we do not want to focus too much on a given problem, namely the Quadratic Assignment Problem. However, the second section of this paper has shown how a classical Tabu Search with a basic neighborhood often approaches the optimum without reaching it and, at least, we want to see how the neighborhood structures presented in this paper perform on that point.

But first, we want to show that the Neighborhood Tree structure as well as the Mimausa Neighborhood seem to perform much better than the classical “K-exchange” neighborhood structure (often called K-opt, this structure is composed of all the solutions where K values of the current solution are exchanged).

As an illustration, we have taken 10000 random “QAP” solutions. On each of them, one iteration (selection of the best move) has been performed with the following neighborhood structures: 3-opt, 4-opt, NT(6,1), NT(6,2,1), MN(7) (3-opt, NT(6,2,1) and MN(7) are of similar sizes). Results are summarized in Table 22.2 and clearly show the inferiority of the K-opt both in global computational time and in average quality. It is true that the quality of the solutions obtained after one iteration is not such a determining factor. A move leading to a low decrease of the objective function can be more diversifying and more interesting for the following of the search. At least, these results show that both the neighborhood tree and the Mimausa neighborhood are much more cheaper to compute than the K-opt.

One of our assertions at the beginning of this paper was that the neighborhood structures presented are of interest for the Variable Neighborhood Search (VNS) or the Variable Neighborhood Descent (VND), methods recently proposed by Hansen and Mladenovic (see for instance [8]) in which different neighborhoods are required. That is the reason why we present a second series of computational results in order to illustrate this assertion. These results are a little bit linked to the previous ones, since, in most implementations of the Variable Neighborhood methods, ‘K-opt’ neighborhoods are used.

We have taken the basic version of VNS, as presented in [8], [9] or [12] that we briefly recall:

N_k , $k = 1, \dots, k_{max}$: set of neighborhood structures.

S : starting solution.

Repeat the following until the stopping condition is met:

$k := 1$; **Repeat** the following steps until $k = k_{max}$:

a) Choose a point S' at random in $N_k(S)$.

b) Apply a local search method with S' as initial solution. The result is denoted S'' .

c) If $val(S'') < val(S)$ then $S := S''$ and $k := 1$

else $k := k + 1$.

We have applied this basic scheme with four different sets of neighborhood structures:

- **Nei1:** N_k is the “ $(k+1)$ -opt” neighborhood structure (it is composed of all the “ $(k+1)$ ”-exchanges. k_{max} is set to 9 (a 10-exchanges is performed when using $N_{k_{max}}$).
- **Nei2:** The last neighborhood structure (N_9) is replaced by a neighborhood tree whose parameters are randomly chosen in short intervals. Each time the neighborhood tree is applied ($k = 9$), the depth is randomly chosen between 2 and 3, the number of sons on the first level in the interval [4, 8], on the second level between 1 and 2: we could say that the structure is “between $NT(4, 1)$ and $NT(8, 2, 1)$ ”.
- **Nei3:** The last neighborhood structure (N_9) is replaced by a Mimausa neighborhood. The size of the optimized subproblem (parameter K) is randomly chosen in the interval [6, 9].
- **Nei4:** The two last neighborhood structures (N_8 and N_9) are replaced by a “neighborhood tree” and a “Mimausa neighborhood” whose parameters are randomly chosen as in Nei2 and Nei3.

The local search method performed in step b. is a basic descent method using the “2-opt” neighborhood. The global method stops when 100000 iterations have been performed (stopping criterion). As a consequence of such a basic implementation, the results are good but not excellent. Anyway, our goal was not to produce the best possible results but just to measure the impact of introducing our neighborhood structures in the set of neighborhoods. In Table 22.3, the results over 10 runs are presented (average value, best value, and average computational time are given).

These results show that the use of a neighborhood tree or of a Mimausa neighborhood in a VNS often causes a slight increase in the computational time. But, on the other hand, the quality of the solutions is better. On the small instances, the optimal solution is more frequently obtained. Moreover, when an optimal solution is reached, it is generally earlier in the search and

	Nugent 15	Elsh. 19	Skor. 56
<i>Avg. value</i>			
Nei1	1161	18518364	35134
Nei2	1154	18012652	34942
Nei3	1153	17845666	35023
Nei4	1153	17823112	34915
<i>Best value</i>			
Nei1	1150	17212548	34868
Nei2	1150	17212548	34810
Nei3	1150	17212548	34842
Nei4	1150	17212548	34790
<i>Avg. time (s)</i>			
Nei1	5.8	6.2	395.8
Nei2	6.4	6.6	427.3
Nei3	7.6	5.3	478.0
Nei4	8.2	6	523.7

Table 22.3 Comparison of four neighborhood sets for a VNS.

this quicker result sometimes produces a decrease in the global computational time (Elsh. 19).

The last computational experiments consist of the integration of our neighborhood structures in the “Robust Tabu Search” and of the comparison with the results presented in the second section of this paper, specially the ones concerning the number of missed approaches. Even if the computation of our neighborhood structures has appeared to be cheaper than a K-opt (with $K > 2$), it remains quite expensive in comparison with a basic neighborhood, such as the 2-opt used in the Robust Tabu Search. The exclusive use of the neighborhood tree or of the Mimausa neighborhood being too much expensive, these neighborhood structures are not applied at each iteration of the method. Mostly, the Robust Tabu Search iterative process is not modified and the 2-opt basic neighborhood is kept. Our neighborhood structures are only deployed when the selected move increases the value of the current solution (local optimum) and when the structures have not been used since a given number of iterations. The scheme can be drawn as follows:

```

 $N_1 :=$  “2-exchange” neighborhood.
 $N_2 :=$  Neighborhood tree.
 $N_3 :=$  Mimausa neighborhood.
 $S :=$  starting solution;
 $it_1 := 0$ ;  $it_2 := 0$ ;
While not stopping criterion do
   $S' :=$  best “not tabu” solution in  $N_1(S)$ ;
  if ((val( $S'$ ) < val( $S$ )) or ( $it_1 < MaxIt_1$ )))
    then

```

```

 $S := S'; it_1 := it_1 + 1;$ 
else
if ( $it_2 < MaxIt_2$ )
then
     $S :=$  best solution in  $N_2(S)$ ;  $it_1 := 0$ ;  $it_2 := it_2 + 1$ ;
else
     $S :=$  best solution in  $N_3(S)$ ;  $it_1 := 0$ ;  $it_2 := 0$ ;

```

Some first results are presented in Table 22.4. For the two Quadratic Assignment applications used in Section 22.2 (Nugent 15 and Elshafei 19), we give the average number (on 100 runs) of basic moves (2-exchanges) performed before reaching the optimum (which is larger than the number of iterations since the move selected in a neighborhood tree or in Mimausa often corresponds to several 2-exchanges) (*Avg # 2 Exch.*). We give also the average number of times our neighborhood structures have been used (*Avg # structure*). We give the average computational times (*Avg Comp Time*) and we finally mention the average number of missed approaches for three different values of the threshold T_1 (see Section 22.2) ($T_1 = 2, 3$ and 4). In these computational experiments, the neighborhood tree and the Mimausa neighborhood have been tested in an exclusive way. In one case, $MaxIt_2$ has been set to infinite (the Mimausa neighborhood is never used) while, in the other case, $MaxIt_2$ has been set to 0 (the neighborhood tree is never used).

Neighborhood structure	Basic neighborhood	Neighborhood tree	Mimausa neighborhood
<hr/>			
Elsh. 19			
Avg # 2 Exch.	12265	3242	561
Avg # structure		344	62
Avg Comp Time	3.83	3.49	0.93
$T_1 = 2$	0.18	0.24	0.03
$T_1 = 3$	2.51	1.04	0.08
$T_1 = 4$	16.27	5.75	0.56
<hr/>			
Nug. 15			
Avg # 2 Exch.	665	488	344
Avg # structure		58	34
Avg Comp Time	0.18	0.28	0.21
$T_1 = 2$	0.35	0.03	0.1
$T_1 = 3$	0.96	0.12	0.26
$T_1 = 4$	2.28	0.48	0.6

Table 22.4 Use of different neighborhood structures.

As, on many iterations, the basic “2-opt” neighborhood keeps being applied, the missed approaches are not completely eliminated by such an occasional use of more sophisticated neighborhood structures. However, as can be seen on Table 22.4, they are noticeably reduced. Consequently, the average number of basic moves performed to reach the optimum is also appreciably reduced. The over-cost due to the occasional use of the special neighborhood structures

is counterbalanced by this quicker convergence and, finally, the computational times are similar for the three implementations. Let us remark that on these results, the “Mimausa neighborhood” appears to be a little bit superior to the neighborhood tree but it is true that on these small instances, the exact method optimizes a substantial subpart of the global problem.

In order to show how these neighborhood structures perform on larger instances, we present in Table 22.5 the results obtained on a small subset of the QAPLIB instances [3] (*Skorin-Kapov 56*, *Taillard 64c* and *Skorin-Kapov 100a*). The optimal solution is unknown for these large instances and it is impossible to know all the solutions whose value is equal to the best known value. For these reasons, the numbers of missed approaches could not be computed on these instances and do not appear on the table. 10 runs have been performed with a maximal number of basic moves (stopping criterion) fixed at 100000. The results presented in the last column correspond to the case where the neighborhood tree and the Mimausa neighborhood are alternatively applied ($MaxIt_2 = 1$). Since, on these large instances, the best known solution is rarely reached, we give the average (on the 10 runs) best value found by the different implementations (*Avg value*).

	Taillard algorithm	Neigh. tree	Mimausa neigh.	Neigh. tree + Mimausa neigh.
<i>Skor. 56 (34458)</i>				
Avg value	34494	34471.6	34474	34473.2
Avg # structure		5220	8552	6756
Avg Comp Time	135	283	207	230
<i>Tail64c</i>				
Avg # 2 Exch.	47226	14939	21781	16435
Avg # structure		933	1750	1123
Avg Comp. Time	197	93.4	181.5	96.7
<i>Skor 100a (152002)</i>				
Avg value	152236	152194	152202	152160
Avg # structure		5450	7672	7036
Avg Comp. Time.	419	685	658	674

Table 22.5 Computational results on large instances.

Despite the size of the instance, the optimum (or supposed optimum) is relatively easy to reach on Taillard 64c. The results obtained on that instance are similar to those presented in Table 22.4: the optimal solution is reached earlier when using the neighborhood tree or the Mimausa neighborhood (or both of them). The Skorin-Kapov instances are more complex to solve and the best known (supposed optimal) solution is rarely obtained. However, as it can be seen on the table, the average value of the solutions finally obtained is noticeably improved by the use of “sophisticated” neighborhood structures. If the average values are not very different, they have to be compared to the optimal value to see that the gap is strongly reduced. On the other hand, it is true that the computational time is a little bit increased. Let us finally remark

that the neighborhood tree seems to perform better than the Mimausa neighborhood on instances with a low flow dominance (Nugent and Skorin-Kapov) while the Mimausa neighborhood seems superior when the flow dominance is high (Elshafei and Taillard 64).

22.6 CONCLUSION

The goal of this paper was to present new neighborhood structures widely applicable to combinatorial optimization problems. Used in a metaheuristic based on local search - a Tabu Search for instance - they may constitute an interesting alternative or complement to the basic neighborhoods that are generally used. Indeed, their occasional use can be seen as an intensifying process that allows a sensible decrease in the number of missed approaches to an optimal (or sub-optimal) solution and that may speed-up the convergence of the method. But, they may also be added to the set of neighborhood structures used in a Variable Neighborhood Search or Descent. In a VND, they appear to be more efficient than a classical K-opt. In a VNS, these neighborhood structures may ensure a quicker convergence to the optimal solution when this one is reached by the method. Otherwise, their use generally improve the quality of the solutions finally obtained. Even though the focus of this paper was general and not on a given application, the computational results obtained on the Quadratic Assignment Problem, are really encouraging since illustrating the previous points.

References

- [1] E. Balas and A. Vazacopoulos. Guided Local Search with Shifting Bottleneck for Job Shop Scheduling. *Management Science*, 44:262–275, 1998.
- [2] K. Budenbender, T. Grunert, and H.-J. Sebastian. A Tabu Search Algorithm for the Direct Flight Network Design Problem. Technical Report 98/04, University of Technology, Aachen, 1998.
- [3] R. Burkard, S. Karisch, and F. Rendl. Qaplib — A Quadratic Assignment Problem Library. *Journal of Global Optimization*, 10:391–409, 1997.
- [4] Y. Caseau and F. Laburthe. Effective Forget-and-Extend Heuristics for Scheduling Problems. In: *Extended Abstracts of the III Metaheuristics International Conference*, pages 129–133, Angra dos Reis, 1999.
- [5] A. Elshafei. Hospital Layout as a Quadratic Assignment Problem. *Operational Research Quarterly*, 28:167–179, 1977.
- [6] F. Glover and M. Laguna. *Tabu Search*. Kluwer, 1997.
- [7] F. Glover, E. Taillard, and D. de Werra. A Users Guide to Tabu Search. *Annals of Operations Research*, 41:12–37, 1993.

- [8] P. Hansen and N. Mladenović. An Introduction to Variable Neighborhood Search. In: *Meta-Heuristics : Advances and Trends in Local Search Paradigms for Optimization*, S. Voss, S. Martello, I.H. Osman, and C. Roucairol, editors, pages 433–458, Kluwer, 1999.
- [9] P. Hansen and N. Mladenovic. Variable Neighborhood Search for the p -Median. *Location Science*, 5:207–226, 1999.
- [10] T. Mautor and P. Michelon. Mimausa: A New Hybrid Method Combining Exact Solution and Local Search. In: *Extended Abstracts of the II Metaheuristics International Conference*, pages 15–16, Sophia-Antipolis, 1997.
- [11] T. Mautor and C. Roucairol. A New Exact Algorithm for the Solution of Quadratic Assignment Problems. *Discrete Applied Mathematics*, 55:281–293, 1994.
- [12] N. Mladenovic and P. Hansen. Variable Neighborhood Search. *Computers and Operations Research*, 24:1097–1100, 1997.
- [13] C. Nugent, T. Vollmann, and J. Rumel. An Experimental Comparison of Techniques for the Assignment of Facilities to Locations. *Operations Research*, 16:150–173, 1968.
- [14] G. Pesant and M. Gendreau. A View of Local Search in Constraint Programming. In: *Proceedings of the Second International Conference on Principles and Practice of Constraint Programming*, pages 353–366, Cambridge, 1996.
- [15] C. Rego and C. Roucairol. Using Tabu Search for Solving a Dynamic Multi-Terminal Truck Dispatching Problem. *European Journal of Operational Research*, 83:411–429, 1995.
- [16] P. Shaw. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In: *Proceedings of the Fourth International Conference on Principles and Practice of Constraint Programming*, page 417, Pisa, 1998.
- [17] E. Taillard. Robust Taboo Search for the Quadratic Assignment Problem. *Parallel Computing*, 17:443–455, 1991.

23 NEW HEURISTICS FOR THE EUCLIDEAN STEINER PROBLEM IN R^n

Flávio Montenegro¹, Nelson Maculan¹,
Gérard Plateau², and Patrick Boucher²

¹Universidade Federal do Rio de Janeiro
COPPE, Programa de Engenharia de Sistemas e Computação
P.O. Box 68511
Rio de Janeiro, RJ 21945-970, Brasil
{ fmonte, maculan }@cos.ufrj.br

²Laboratoire d'Informatique de Paris Nord, UPRES-A CNRS 7030
Institut Galilée
Université Paris XIII
Avenue Jean-Baptiste Clément
93430 Villetteuse, France
{ gerard.plateau, patrick.boucher }@lipn.univ-paris13.fr

Abstract: We introduce two new heuristic approaches for the Euclidean Steiner Tree Problem in R^n (ESTP). The first approach is an extension to $n \geq 3$ of a relaxation scheme developed in the plane and based on a physical model for the ESTP. Such a model arises from the dynamics of a fluid film subject to surface tension forces. The second one is based on a local search of full Steiner tree topologies within a neighborhood for which we apply a tabu search heuristic. Some ingredients of the first approach are used in the second one. Implementations have been carried out and preliminary results are provided for certain parameters of both these heuristics.

23.1 INTRODUCTION

The Euclidean Steiner Tree Problem in R^n (ESTP) can be defined as follows: given p points in R^n , find a minimum tree which spans these points using or

not extra points (Steiner points). The distances considered are Euclidean ones. For a historical background see [8].

An enumerative scheme to solve the ESTP was proposed by Smith [14]. Maculan et al. [9] formulated the ESTP as a nonconvex mixed-integer programming problem and proposed a Lagrangean dual program in order to develop a branch-and-bound method.

In this paper we present two new heuristic approaches to this problem. The first one is an extension for $n \geq 3$ of a relaxation scheme developed by Chapeau-Blondeau et al. [2] in the plane. It is based on a physical analogy related to the dynamics of a fluid film under surface tension forces. The second approach is a local search method for the ESTP for which we apply a tabu search metaheuristic technique. Some ingredients of the first approach are used in the second one. We are applying both these approaches for $n \geq 3$, see [1, 16] for references for the case $n = 2$. MacGregor-Smith et al. [13] have presented an interesting heuristic for $n = 3$.

Some examples of Steiner minimum trees are illustrated below (Figure 23.1).

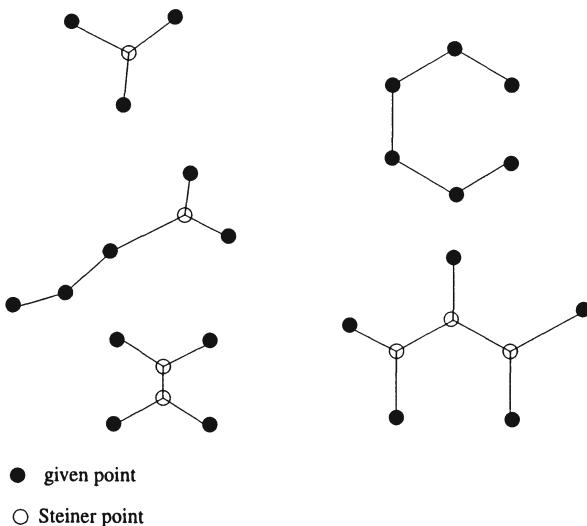


Figure 23.1 Examples of Steiner minimum trees.

Gilbert and Pollack [7] stated the following properties:

Property 1 Given p points $x^i \in R^n$, $i = 1, 2, \dots, p$, the maximum number of Steiner points is $p - 2$.

Property 2 A Steiner point (nondegenerated) has degree (valence) equal to 3.

Property 3 The edges emanating from a Steiner point (nondegenerated) lie in a plane and have mutual angle equal to 120° .

The Steiner trees with $p - 2$ Steiner points are known as full topology Steiner trees. To illustrate full topologies we will consider four given points in R^3 and we present three full topologies associated with these four points in Figure 23.2.

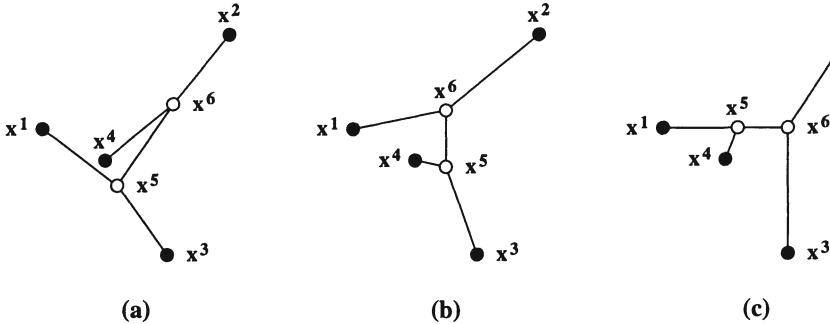


Figure 23.2 Three different full topologies associated with the same configuration of four given points in R^3 .

We denote by $|x^i - x^j| = \sqrt{\sum_{l=1}^n (x_l^i - x_l^j)^2}$ the Euclidean distance between x^i and x^j . Thus for computing the positions of x^5 and x^6 in Figure 23.2 we have to solve three different optimization problems.

For the example (a): minimize $|x^1 - x^5| + |x^3 - x^5| + |x^5 - x^6| + |x^4 - x^6| + |x^2 - x^6|$, subject to x^5 and $x^6 \in R^3$.

For the example (b): minimize $|x^3 - x^5| + |x^4 - x^5| + |x^5 - x^6| + |x^1 - x^6| + |x^2 - x^6|$, subject to x^5 and $x^6 \in R^3$.

For the example (c): minimize $|x^1 - x^5| + |x^4 - x^5| + |x^5 - x^6| + |x^2 - x^6| + |x^3 - x^6|$, subject to x^5 and $x^6 \in R^3$.

These minimization problems can be solved by a nonlinear programming algorithm presented in [14].

If we consider again p points in R^n and $k \in \{0, 1, 2, \dots, p - 2\}$ Steiner points, Gilbert and Pollack [7] stated that the total number of different topologies with k Steiner points is:

$$C_{p,k+2} \frac{(p+k-2)!}{k!2^k}.$$

When $k = p - 2$ we have full Steiner topologies. In this case the above relation will be written as $1.3.5.7\dots(2p-5) = (2p-5)!!$, where the two exclamation marks stand for the double factorial notation. For example, if $p = 10$ then the total number of full Steiner topologies will be $15!! = 1.3.5.7.9.11.13.15 = 2,027,025$.

Now we consider four given points illustrated at Figure 23.3: x^1, x^2, x^3 and x^4 ; and we consider a full Steiner topology which is presented by dashed lines. Thus we minimize $|x^1 - x^6| + |x^2 - x^6| + |x^5 - x^6| + |x^3 - x^5| + |x^4 - x^5|$, subject to x^5 and $x^6 \in R^n$, and we have the optimal solution $(x^{5'}, x^{6'})$, in which $x^{6'} = x^2$ and $x^{5'}$ is an actual Steiner point. One can state that $x^{6'}$ is a degenerated Steiner point. The best solution associated with this configuration has only one Steiner point.

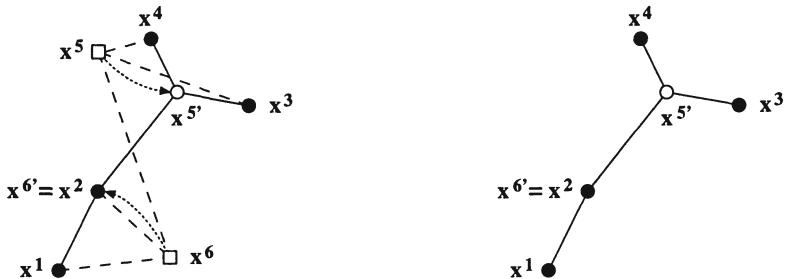


Figure 23.3 Optimization of a tree with full topology. At the end of the minimization process, we obtained only one actual Steiner point ($x^{5'}$), plus a degenerated Steiner point ($x^{6'} = x^2$).

Another example using four points is presented in Figure 23.4. In this case the two Steiner points are the same (degenerated) one: $x^{5'} = x^{6'}$.

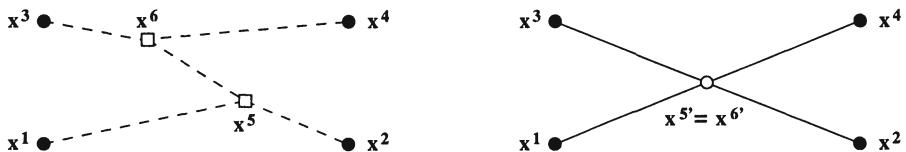


Figure 23.4 An example of two Steiner points that end up at the same position (a degenerated Steiner point) in the minimization process.

Sometimes the best solution obtained from a full Steiner topology has no Steiner points, as illustrated in Figure 23.5.

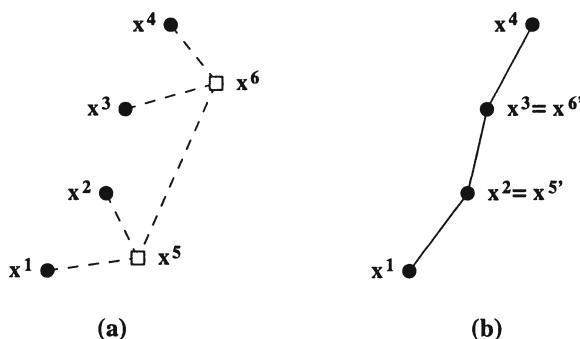


Figure 23.5 Example of a solution with no actual Steiner points after the minimization process.

The Steiner tree associated with Figure 23.5(b) can be defined as a full Steiner topology in which $|x^2 - x^5| = |x^3 - x^6| = 0$.

Actually, we can consider all (connected) nonfull tree topologies as full Steiner topologies with one or more degenerated Steiner points [14]. So, making use of adequate minimization methods, it suffices to concentrate our attention on full topologies when we are looking for solutions to the ESTP.

In the next sections we present two heuristic approaches to find good solutions to the ESTP. Both heuristics perform the minimization of full Steiner topologies. To judge how good is the solution found by a heuristic algorithm we use the so called *Steiner ratio* ρ , defined by

$$\rho = \frac{L_H}{L_{MST}}, \quad (23.1)$$

where L_H is the length of the solution obtained by the heuristic and L_{MST} is the length of the minimum spanning tree (MST) for the given distribution of points. So, we want to find solutions with low ρ -values. In the cases in which the optimal solution is already known, the solution of the heuristic can be compared directly with the optimum value of the Steiner ratio, given by

$$\rho_{OPT} = \frac{L_{SMT}}{L_{MST}}, \quad (23.2)$$

where L_{SMT} is the length of the Steiner minimum tree (SMT), the solution of the Steiner tree problem.

23.2 A DYNAMIC RELAXATION SCHEME

The Chapeau-Blondeau et al. [2] heuristic is based on a physical analogy to model the Steiner problem in the Euclidean plane [3, 10].

Consider a soap film hooked between pins (the given points of a Steiner problem) that connect two parallel plates. Subject to unit forces due to surface tension, the film relaxes to an equilibrium configuration that, neglecting gravity, minimizes the potential energy associated with the surface tension. This configuration of minimal energy corresponds to a minimum of the length of the film among the pins, describing a pattern of a (local minimum) Steiner tree.

The heuristic of Chapeau-Blondeau et al. starts from an initial tree with full Steiner topology. This tree is constructed by the addition of Steiner points to the MST, in such a manner that each given point ends up connected to the rest of the tree by only one edge (see [2] for details). The relaxation scheme for this initial tree consists of two basic iterative processes: an evolution process and an interaction process.

In the evolution process each Steiner point S is allowed to move under the resultant \vec{F} of the unit forces acting on it by its three incoming edges. The displacement is proportional to \vec{F} , with a proportionality coefficient λ .

The interaction process occurs when a Steiner point S comes, by the evolution process, within a distance T from a neighbor Steiner point S' . This process makes possible to change the topology of the tree, as illustrated in Figure 23.6.

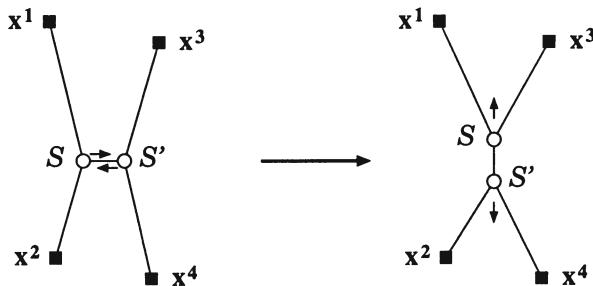


Figure 23.6 Interaction process: changing the connections of two Steiner points S and S' within a distance T . Each point x^1, x^2, x^3 , and x^4 , represented by solid squares, can be either a given point or a Steiner point.

For each one of the three triplets $\{x^1, x^2, S'\}$, $\{x^1, x^3, S'\}$ and $\{x^1, x^4, S'\}$ the resultant force on S is computed (these triplets correspond to the three possibilities to connect the points x^1, x^2, x^3 and x^4 through full topologies; the triplet $\{x^3, x^4, S'\}$ is not interesting, because it represents just a permutation of the positions of S and S' with no change to the topology of connections of the tree). These three forces are compared. The triplet for which the resultant force has the greater magnitude is retained for S . The complementary change of connections is applied to S' , and this completes the interaction process.

The algorithm of Chapeau-Blondeau et al., based on this relaxation scheme to obtain heuristic solutions for a Steiner problem with p given points, is presented in Figure 23.7.

Given an MST and a fixed number of iterations, this algorithm has complexity $O(p)$. In dimension $n = 2$ the MST can be obtained in an $O(p \log p)$ procedure [8, 12], so a complete heuristic for the Steiner problem in the Euclidean plane can be constructed, through the application of the relaxation scheme, with complexity $O(p \log p)$.

This algorithm was implemented by the authors for configurations of p points randomly distributed in an unit square. The variable σ_p in the algorithm can be understood as a natural unit of length, an average separation between a given point and its nearest neighbor in that unit square. The schedule to reduce the values of the parameters T and λ was the following: start to reduce the value of T , with constant λ , at the iteration $k = 100$; T is reduced by one-fifth of its initial value after every twenty iterations, vanishing at iteration $k = 180$. Start reducing λ at iteration $k = 200$, with its value being divided by two every twenty iterations, until $k = 400$, when the algorithm stops. Despite of the low complexity $O(p)$, some interesting results, corresponding to final configurations of relatively low Steiner ratios ρ , were obtained for some initial values of the parameters T and λ .

We have developed an almost direct extension of the heuristic of Chapeau-Blondeau et al. to treat the Steiner problem with p given points in dimension

```

Set  $\sigma_p = p^{-1/2}$  as the unit of length.
Initialization: construct an initial Steiner tree.
Initialize  $T$  and  $\lambda$  to small fractions of  $\sigma_p$ .
Iteration step  $k = 1$ 
Repeat
  For each Steiner point  $S$  of the Steiner tree
    Evolution:
       $S$  is connected to 3 neighboring nodes  $A, B$  and  $C$ 
      Compute the resultant force on  $S$  as  $\vec{F} = \frac{\vec{SA}}{\|\vec{SA}\|} + \frac{\vec{SB}}{\|\vec{SB}\|} + \frac{\vec{SC}}{\|\vec{SC}\|}$ 
      Displace  $S$  by  $\vec{OS}' \leftarrow \vec{OS} + \lambda \vec{F}$ 
    If among the 3 neighbors of  $S$  is a Steiner point distant of less than  $T$ 
      Interaction:
        Call  $S'$  the Steiner point neighbor of  $S$  such that  $\|SS'\| < T$ 
        ( $S'$  can be  $A, B$  or  $C$ . If there are more than one possible  $S'$ 
        then pick one of them at random).
        Exchange neighbors between  $S$  and  $S'$  as explained in the text
        and in Figure 23.6.
    EndIf
  EndFor
  Decrease  $T$  and  $\lambda$  according to a predefined schedule.
   $k \leftarrow k + 1$ 
Until a criterion for convergence is satisfied.

```

Figure 23.7 Algorithm of Chapeau-Blondeau et al. for the Steiner problem for p given points, which results from the application of the relaxation scheme for an initial Steiner tree.

$n \geq 3$. To do that, we do not worry, at least at this time, about the interpretation of the dynamics of the soap film in these higher dimensions. We just use the composition of unit forces (like that related to surface tension) acting in the ($n \geq 3$)-dimensional space, as we suppose that, in the relaxation scheme above, it suffices to drive the topology of the solution tree to an adequate equilibrium configuration, where the tree may present a good local minimum length.

The algorithm is basically the same of Figure 23.7. However, some considerations are made, specially with regard to obtaining the initial Steiner tree, whose construction, following that of Chapeau-Blondeau et al., is made by visiting once each given point and its neighbors in the MST. The number of neighbors of a given point in the MST cannot be greater than 6 for $n = 2$, but actually this number increases with n . To solve this problem, we have explored the fact that the maximum number of neighbors to a given point in the MST cannot be greater than $p - 1$. This number is also exactly the number of edges in the MST, no matter the value of n , which has enabled us to avoid an excessive increment of the required computer time and memory.

Other consideration is related to the variable σ_p , which is now treated as dependent of both the number of given points p and the dimension n :

$$\sigma_p = p^{-1/n}. \quad (23.3)$$

Also, in our implementation Steiner points are treated in random order at each iteration, in such a way to prevent a bias that a prefixed order may introduce.

Similarly to the authors of the original 2-dimensional relaxation scheme, and following the same schedule with 400 iterations, we implemented and applied the extended relaxation scheme for various configurations of given points randomly distributed in an unit hypercube. Some results are listed in Table 23.1, which presents the mean ρ -value obtained by this extended heuristic for 10^{5-a} problems with $p = 10^a$ given points, where a is an integer exponent varying from 1 to 4. The results are presented for dimensions 3, 4, 5 and 10. To construct the MST we are using the $O(p^2)$ -complexity Prim's algorithm [11]. The CPU time (in seconds) considers the total time to construct all the MSTs and to do all the relaxations for each set of problems. All the time results presented in this work are referent to a Sun Ultra 1 workstation. The standard deviations are also presented for each set. These results were obtained with initial parameters $T = 0.18\sigma_p$ and $\lambda = 0.12\sigma_p$.

p		n			
		3	4	5	10
10 (10,000 pbs.)	Mean ρ	0.951522	0.932607	0.916304	0.864513
	st. dev.	0.017663	0.018602	0.019062	0.018263
	CPU time	640	750	870	1,500
100 (1,000 pbs.)	Mean ρ	0.953682	0.939578	0.928165	0.886429
	st. dev.	0.005476	0.005804	0.005487	0.005531
	CPU time	860	1,000	1,200	2,000
1,000 (100 pbs.)	Mean ρ	0.953672	0.940196	0.930097	0.900173
	st. dev.	0.001482	0.001633	0.001650	0.001777
	CPU time	1,000	1,200	1,400	2,300
10,000 (10 pbs.)	Mean ρ	0.954098	0.941104	0.931523	0.904967
	st. dev.	0.000472	0.000305	0.000535	0.000516
	CPU time	2,900	3,400	3,900	6,400

Table 23.1 Mean ρ -value, standard deviation and CPU total time (in seconds) obtained by the extended heuristic for various sets of randomly distributed given points in the unit hypercube. For each dimension, it was tested a set of 10^{5-a} problems with 10^a points, where a is an integer exponent varying from 1 to 4.

From Table 23.1 we can observe some interesting features. For example, the mean ρ -value seems to decrease with the increasing of the dimension n . In fact, we have verified that this trend continues even for very high dimensions. Experimenting 10 configurations of 10 given points in dimensions $n = 100$ and $n = 1,000$, our extended heuristic obtained the mean values $\rho = 0.771929$ and $\rho = 0.741995$ respectively, both these cases presenting low dispersions

(st. dev. < 0.005). This feature is according to some conjectures about the minimum optimal ρ -value possible for a dimension n , that is, its *infimum* ρ_n . In dimension 3 the conjectured infimum is $\rho_3 \approx 0.784190373377122$ [15]. For higher dimensions the value of ρ_n is supposed to be reduced as the value of n increases, with a lower bound $\rho_n \geq 0.615827\dots$ [4, 5]. Observing the values reported above, we see that if the conjecture for $n = 3$ is true, then some reduction is really necessary for the ρ_n -value in higher dimensions, at least to permit that results for $n = 100$ and $n = 1,000$. Furthermore, it seems that the reduction of the (conjectured) ρ_n -value with the raising of the dimension may be accompanied by a reduction also in the expected ρ -value for generic distributions.

We also have tested a set of 1,000 configurations of 10 given points in $n = 3$ dimensions for which we know the optimal solutions. These optimal solutions were obtained through the exact (exhaustive) algorithm of Warren Smith [14]. His algorithm provided the mean value $\rho = 0.946758$ (st. dev. = 0.017531) spending a total CPU time of about 19 hours. Our extended heuristic has obtained the mean value $\rho = 0.951185$ (st. dev. = 0.018012), spending a total CPU time of 73 seconds. There was not any coincidence of ρ -values, but the heuristic has obtained 461 results up to 0.01%, 831 up to 1.00% and all the 1000 results up to 4.87% greater than the respective results obtained by the Smith's algorithm.

These are some preliminary results. There is more work to be done, for example, in adjusting parameters T and λ and testing other different types of configuration of given points. Besides, a problem with this extended heuristic is that it is not sufficiently robust, providing solutions distant from the optimum in a considerable number of cases. We hope, in a future work, to improve these results by combining some characteristics of this relaxation scheme with that of another heuristic approach to be introduced in the following.

Our purpose in the next section is to develop a local search from a given full Steiner topology tree. We try to find in the neighborhood of this topology other full Steiner topologies to be minimized. A tabu search is then applied in order to escape from local minima.

23.3 A LOCAL SEARCH FOR THE ESTP

In this section we introduce another approach to the ESTP. It consists of the development of a local search to explore a defined neighborhood of a given full Steiner topology and its implementation in the context of the tabu search metaheuristic. This approach uses some ingredients of the extended heuristic presented in the last section. Some preliminary results we have found are presented at the end of this section, and are related to some of the parameters of the tabu search heuristic.

23.3.1 Local search: a neighborhood of a given Steiner topology

Given a full Steiner topology we would like to find another full Steiner topology from this given topology.

We will introduce some definitions.

[1] Free edges:

A free edge is an edge of a full Steiner tree topology whose both terminals are Steiner points.

[2] Steiner free tree:

The Steiner free tree is the sub-tree of the full Steiner tree topology formed by all free edges (Figure 23.8).

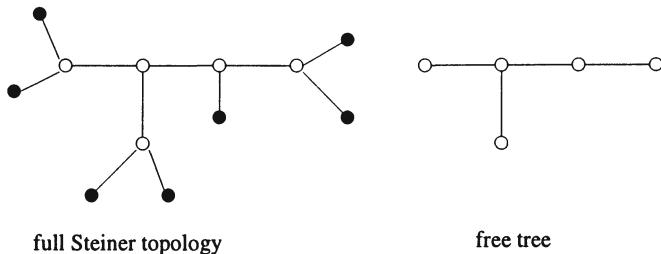


Figure 23.8 A full Steiner topology and its corresponding Steiner free tree.

We will just consider the full Steiner topologies derived from a given full Steiner tree topology in which the connections among given points and Steiner points do not change. We can replace only free edges. For an example see Figure 23.9.

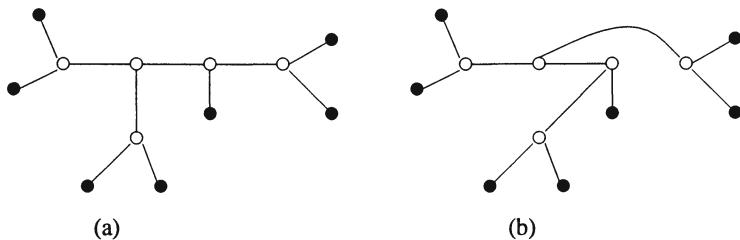


Figure 23.9 Replacement of two free edges of a full Steiner topology.

From a given full Steiner topology we will only consider the neighbors in which we just replace two free edges. It is easy to see that to obtain a new full Steiner topology the free edges to be replaced cannot be adjacent. In Figure 23.10 we illustrate an example.

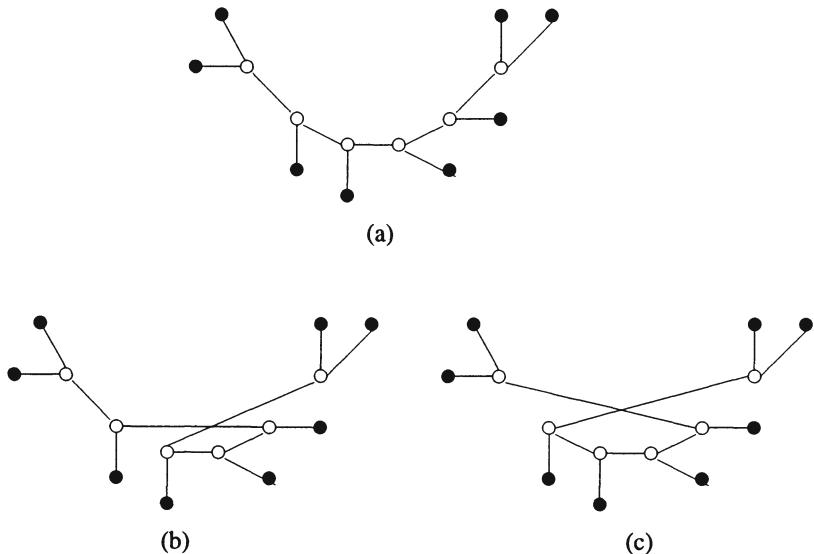


Figure 23.10 Different choices of nonadjacent free edges to be replaced in (a) will lead to different full topologies, for example that showed in (b) and (c). Replacement of adjacent free edges in (a) will result in the same topology.

When we remove two nonadjacent free edges to be replaced by two others we have to create a new full Steiner tree topology. Once we choose two nonadjacent free edges to be replaced, we have just one possibility to form a new full Steiner tree topology. To justify this statement, we observe that to join four nodes (two by two) we have only three ways. For one possibility of connection we reconstruct the initial full Steiner tree topology, another possibility gives us a nonconnected graph. Thus we have only one possibility to obtain a new full Steiner tree topology from a given full Steiner tree topology, see Figure 23.11.

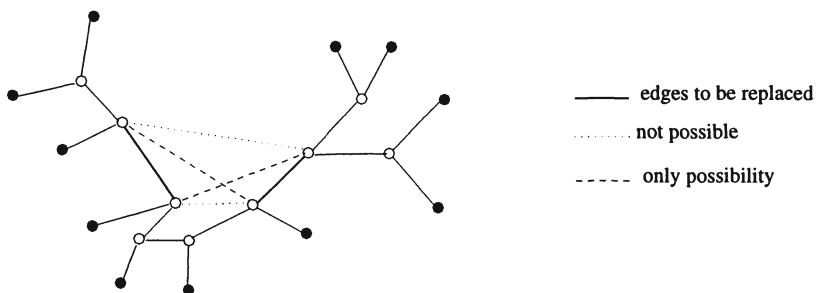


Figure 23.11 Dashed lines represent the only possibility of reconnection for the chosen edges to be replaced, in such a way to obtain a new full topology.

Neighborhood search. The total number of neighbors associated with a given full Steiner tree topology is $O(p^2)$. Depending on the size of the full Steiner tree topology, we will not be able to consider all its $O(p^2)$ neighbors. Thus we introduce a root and a radius for the Steiner free tree. Once we choose a node of the Steiner free tree to be a root, we orientate the edges of this tree in order to allow a directed path from this root to all others nodes, i. e., we define a partial order in this directed tree.

The root divides the directed tree in at most three regions, according to the number of free edges outgoing from it, limited by the radius. Replacements are made between pairs of free edges within the same region of the tree. We say that two free edges are in the same region of the directed tree if at least one of the two extreme nodes of each edge has a non-root common ancestor.

The local search works as follows:

Step 1) given an initial full Steiner topology, let us call it the current topology; apply a minimization process to this current topology to find the current solution tree and the current best ρ -value;

Step 2) (iteration) choose a root and direct the current topology;

Step 3) replace two free edges in the current topology and apply the minimization process to the resulting full Steiner topology;

Step 4) compare the Steiner ratio ρ obtained with the current best ρ -value.

If this new ρ -value is better, take it as the new current best ρ -value and store the corresponding new current solution tree;

Step 5) restore the current topology;

Step 6) repeat steps 3 to 5 to investigate all possible pairs of free edges in all regions;

Step 7) repeat steps 3 to 6 with other roots, in the same (redirected) current full Steiner topology, separated by regular distance intervals from the first root; this step avoids an excessive concentration into a restricted region of the neighborhood;

Step 8) if the current best ρ -value is better than that of the current solution tree, take the corresponding topology as the new current topology and go to step 2 (new iteration);

Step 9) else, output the current solution as the best full Steiner solution and stop the local search.

Implementation of the local search. To implement the local search we used the Smith's exact exhaustive algorithm [14]. This program generates and optimizes a great number of full topologies to find SMTs in R^n , so we have made an adaptation on it to optimize just given full topologies. Roots were randomly choosed at step 2, and the interval among roots in step 7 was choosed

as $\lfloor p/3 \rfloor$ for the problems we are presenting here. The radius of search was fixed at three.

The results obtained were not good to random initial full Steiner topologies. As an example of a typical result, for a random distribution of 30 given points in a cube (dimension 3), the ρ -value goes down from $\rho = 1.908440$ to $\rho = 1.797613$ (greater than the MST) after 10 iterations (CPU time: 15 seconds).

Besides the known behaviour of local searches, that they commonly fall rapidly in a local minimum, we have two more explanations for these bad results. First, the neighborhood we are using, constructed based on replacing of just free edges, is somewhat restricted. Usually the replacement of pairs of free edges cannot enable us to reach the optimal topology. For example, if two given points x^1 and x^2 are adjacent to the same Steiner point S , it will remain adjacent to S all the time. If x^1 and x^2 are not adjacent in the optimal topology, replacements of free edges cannot conduct the current solution to the optimal one.

Second, when we do the optimization of a prespecified Steiner tree topology, it is common to obtain some Steiner points at the same position in the solution tree, describing a pattern of crossing edges (see Figure 23.4). Usually this is indicative of a constraint of the given topology that forbids the Steiner point to go to its correct position, giving rise to bad solutions.

To try to solve these problems, we propose a tabu search metaheuristic, whose main features, with some preliminary results, we present on the following.

23.3.2 A tabu search algorithm

We have implemented a tabu search based on the local search described above. The tabu search metaheuristic enables the search to continue beyond the first local minimum found, using a tabu list of forbidden solutions to prevent the search going back to an already visited local minimum, see [6] for details. Tabu search also enable us to introduce a mechanism of diversification to try to compensate some of the restrictions of the defined neighborhood. This mechanism is based on the extended relaxation scheme we presented in the last section.

In our tabu search implementation, a move is the replacement of two free edges in the current solution to reach the best neighbor solution. If the best neighbor solution has a ρ -value greater than the current one, the reverse of the current move is stored in a tabu list of size $Tmaxsize$.

Diversification proceeds as follows: After a certain number $Tdiver$ of iterations without reduction of the best ρ -value, look for pairs of adjacent Steiner points localized at the same position or near each other, within a maximum distance $Tmaxdist$ in the current best solution tree. Then replace edges, not necessarily free edges, choosed according to the mechanism of comparison of forces used in the relaxation scheme described in the last section (see Figure 23.6). That mechanism leads to new topologies that cannot be reached through replacements of free edges. After a number $Tmaxiter > Tdiver$ of iterations without reduction of the best ρ -value the search ends.

We have used the best ρ -value as the aspiration criterion. An intensification is made just when the diversification fails, that is, when the optimal topology is not changed at the diversification.

At the current stage of the implementation, the quality of the best solution found depends on the choice of the initial topology. Good initial topologies can be obtained from the MST using the same construction of initial full Steiner trees that we have presented in the last section.

Some preliminary results were obtained for sets of given points randomly distributed in an unit hypercube in R^n . Table 23.2 contains some of these results, related to four distributions of 30 given points, one for each dimension $n = 3, 4, 5$ and 10, and parameters $Tmaxsize = 10$, $Tmaxiter = 60$, $Tdiver = 20$ and $Tmaxdist = 0.1$. The presented distribution in three dimensions is that for which we have applied the local search in the example above. We include the solution obtained by the extended relaxation scheme for comparison.

Heuristic		n			
		3	4	5	10
Tabu: random initial tree	Initial ρ	1.908440	1.609124	1.380460	1.049683
	Final ρ	1.143431	1.166904	1.053670	0.970418
	nb. iter.	405	475	403	453
	CPU time	710	990	790	1,300
Tabu: MST derived initial tree	Initial ρ	0.951580	0.954699	0.940332	0.888046
	Final ρ	0.943675	0.950510	0.928105	0.869665
	nb. iter.	84	185	68	82
	CPU time	140	380	200	650
Extended relaxation	Final ρ	0.946895	0.942472	0.916997	0.871621
	CPU time	0.23	0.27	0.32	0.52

Table 23.2 Values of ρ obtained for the tabu search for random initial tree and MST derived initial tree, and for the extended relaxation scheme. The parameters for the tabu search are $Tmaxsize = 10$, $Tmaxiter = 60$, $Tdiver = 20$ and $Tmaxdist = 0.1$. The number of iterations is presented for each given set. The CPU times are presented in seconds. For each dimension, 3, 4, 5 and 10, it was tested one distribution of 30 given points.

As we see in the Table 23.2, the computational times and the ρ -values obtained starting from the MST seems to be much better than those obtained when we start from a random initial tree in the tabu search. But if we compare the computational times of the best tabu search results with that of the extended relaxation, we can see clearly the supremacy of the latter in relation to the former. Despite of this, the ρ -values are sometimes better when obtained by the tabu search/MST heuristic. Besides, from the tabu search results we can also perceive the same trend to reduction of the ρ -value in higher dimensions that we observe for the extended relaxation scheme.

Also, we have applied this tabu search to the same set of one thousand problems with $p = 10$ given points in dimension three, for which we know the optimal solutions. The tabu search has obtained the mean value $\rho =$

0.955982 (st. dev. = 0.017954) for the entire set, against the mean value $\rho = 0.946758$ found by the Smith's algorithm and $\rho = 0.951185$ found by the extended relaxation. The total CPU time spent was about 7 hours. Despite the relatively high mean ρ -value obtained, tabu search found 285 optimal solutions and the worst among the one thousand results obtained was 6.53% greater than the respective optimum.

23.4 CONCLUSIONS

In this work we have introduced two new heuristics to the ESTP. The first one is an extension for $n \geq 3$ of a heuristic based on a physical model to the ESTP and developed in the plane. This heuristic seems to work well and quickly for problems with up to 10,000 given points, despite some problems with respect to the robustness of the search. The second heuristic, a tabu search applied to a local search of neighboring topologies, demands much more time to be finished, but the solutions obtained seem to be better, at least in some cases, obtaining the optimal solution in a considerable number of cases. We intend to continue this study in a future work, not only with the adjustment of parameters (specially in the tabu search) and testing with other types of configurations, but also combining the two heuristics and exploring some of the better characteristics of one and other heuristic.

Acknowledgements: We would like to thank the referees for their useful comments.

References

- [1] J.E. Beasley. A Heuristic for Euclidean and Rectilinear Steiner Problems. *European Journal of Operational Research*, 58:284–292, 1992.
- [2] F. Chapeau-Blondeau, F. Janez, and J.-L. Ferrier. A Dynamic Adaptive Relaxation Scheme Applied to the Euclidean Steiner Minimal Tree Problem. *SIAM Journal on Optimization*, 7:1037–1053, 1997.
- [3] R. Courant and H. Robbins. *What is Mathematics*. Oxford University Press, 1941.
- [4] D.Z. Du. On Steiner Ratio Conjecture. *Annals of Operations Research*, 33:437–449, 1991.
- [5] D.Z. Du and W.D. Smith. Disproofs of Generalized Gilbert-Pollak Conjecture on the Steiner Ratio in Three or More Dimensions. *Journal of Combinatorial Theory*, series A, 74:115–130, 1996.
- [6] F. Glover and M. Laguna. *Tabu Search*. Kluwer, 1997.
- [7] E.N. Gilbert and H.O. Pollak. Steiner Minimal Trees. *SIAM Journal on Applied Mathematics*, 16:1–29, 1968.

- [8] F.K. Hwang, D.S. Richards, and P. Winter (editors). *The Steiner Tree Problem*. North-Holland, 1992.
- [9] N. Maculan, P. Michelon, and A.E. Xavier. The Euclidean Steiner Tree Problem in R^n : A Mathematical Programming Formulation. *Annals of Operations Research*, 96:209–220, 2000.
- [10] W. Miehle. Link-Length Minimization in Networks. *Operations Research*, 6:232–243, 1958.
- [11] R.C. Prim. Shortest Connection Networks and Some Generalizations. *The Bell System Technical Journal*, 36:1389–1401, 1957.
- [12] M.I. Shamos and D. Hoey. Closest-Point Problems. *Proceedings of the 16th Annual Symposium on Foundations of Computer Science*, 151–162, 1975.
- [13] J. MacGregor Smith, R. Weiss, and M. Patel. An $O(N^2)$ Heuristic for Steiner Minimal Trees in E^3 . *Networks*, 25:273–289, 1995.
- [14] W.D. Smith. How to Find Steiner Minimal Trees in Euclidean d-Space. *Algorithmica*, 7:137–177, 1992.
- [15] W.D. Smith and J. MacGregor Smith. On the Steiner Ratio in 3-Space. *Journal of Combinatorial Theory*, series A, 69:301–332, 1995.
- [16] P. Winter and M. Zachariasen. Large Euclidean Steiner Minimum Trees in an Hour. Technical Report, Department of Computer Science, University of Copenhagen, 1996.

24 MATHEMATICAL ANALYSIS OF EVOLUTIONARY ALGORITHMS

H. Mühlenbein and Th. Mahnig

Real World Computing Partnership Theoretical Foundation GMD Laboratory
GMD-Forschungszentrum Informationstechnik D-53754 Sankt Augustin
Germany
`{muehlenbein, mahnig}@gmd.de`

Abstract: We analyze evolutionary algorithms which use a population of search points at each step. A popular example are genetic algorithms. We approximate genetic algorithms by an algorithm using *Univariate Marginal Distributions* (UMDA). UMDA generates the search points from a probability distribution. We derive difference equations for the marginal distributions which completely describe the dynamic behavior of UMDA. The key result is that UMDA transforms the discrete optimization problem to a continuous one. The continuous optimization problem is solved by gradient ascent. We show that UMDA solves many difficult multi-modal optimization problems. UMDA is extended to the *Factorized Distribution Algorithm* (FDA) which uses a general factorization of the search distribution into conditional and marginal distributions. We prove that FDA with Boltzmann selection converges to the set of global optima. A new adaptive selection schedule for Boltzmann selection is derived. FDA is extended to an algorithm LFDA, which computes the factorization from promising search points. LFDA is based on learning of Bayesian networks.

24.1 INTRODUCTION

In this paper we analyze a large class of heuristic optimization methods using a population of search points. The mathematical analysis is based on search distributions and their evolution in time. The analysis reveals similarities between heuristics considered to be different - for example genetic algorithms, simulated annealing, ant colony optimization and the system dynamics approach to optimization.

The outline of the paper is as follows. In Section 24.2 the *Univariate Marginal Distribution Algorithm* (UMDA) is introduced and analyzed. We show that UMDA is mathematically an approximation of any genetic algorithm using primarily recombination. The main theoretical result is a difference equation describing the change of the univariate marginal distributions. The difference equation is derived for proportionate selection. Tournament selection leads to more complex difference equations (Section 24.3.1). Truncation selection is analyzed by a theory invented in quantitative genetics (Section 24.3.2).

Mathematically the difference equation transforms the discrete optimization problem into a continuous one. It defines a dynamic equation for solving the continuous problem. Thus it is possible to solve the continuous optimization problem instead of the discrete one. This approach is discussed in Section 24.4.1. It leads to the *system dynamics approach* for optimization. We evaluate UMDA for discrete functions by numerical experiments and theoretical analysis in Section 24.5.

UMDA is extended to the *Factorized Distribution Algorithm* (FDA), which uses a factorization of the search distribution into conditional distributions. FDA with Boltzmann selection is investigated in Section 24.6. We prove convergence to the set of global optima. A new adaptive schedule for Boltzmann selection is derived in Section 24.7.

The mathematical analysis is exact for an infinite population. The problem of finite samples and mutation is discussed in the next two sections. In Section 24.10 LFDA (Learning Factorized Distribution Algorithm) is introduced. LFDA computes a factorization of the distribution from the selected data points. LFDA is based on the theory of Bayesian networks. We conclude the paper with a classification of the different search methods analyzed.

24.2 UNIVARIATE MARGINAL DISTRIBUTION ALGORITHM

Let $\mathbf{x} = (x_1, \dots, x_n)$ denote a vector. For notational simplicity we restrict the discussion to binary variables $x_i \in \{0, 1\}$. We use the following conventions. Capital letters X_i denote variables, small letters x_i assignments. Let a function $f : \mathcal{X} \rightarrow \mathbb{R}_+$ be given. We consider the optimization problem $\mathbf{x}_{opt} = \operatorname{argmax} f(\mathbf{x})$.

Definition: Let $p(\mathbf{x}, t)$ denote the probability of \mathbf{x} in the population at generation t . Then $p_i(x_i, t) = \sum_{\mathbf{x}, X_i=x_i} p(\mathbf{x}, t)$ defines the univariate marginal distributions.

We write $p_i(x_i)$ if just one generation is discussed. We recall two popular recombination/crossover operators used in genetic algorithms.

Definition: Let two strings \mathbf{x} and \mathbf{y} be given. In *one-point crossover* the string \mathbf{z} is created by randomly choosing a crossover point $0 < l < n$ and setting $z_i = x_i$ for $i \leq l$ and $z_i = y_i$ for $i > l$. In *uniform crossover* z_i is randomly chosen with equal probability from $\{x_i, y_i\}$.

For our result concerning genetic algorithms, we need the following definitions.

Definition: *Robbins' proportions* are defined by the distribution

$$\pi_p(\mathbf{x}, t) := \prod_{i=1}^n p_i(x_i, t). \quad (24.1)$$

A population in Robbins' proportions is called to be in *linkage equilibrium* in population genetics.

In [10, 16] we have shown: All complete recombination schemes lead to the same univariate marginal distributions after one step of selection and recombination. If recombination is used for a number of times without selection, then the genotype frequencies converge to linkage equilibrium. This means that *all genetic algorithms are identical if after one selection step recombination is done without selection a sufficient number of times*. This fundamental algorithm keeps the population in linkage equilibrium.

Instead of performing recombination a number of times in order to converge to linkage equilibrium, one can achieve this in one step by *gene pool recombination* [17]. In gene pool recombination a new string is computed by randomly taking for each loci a gene from the distribution of the selected parents. This means that gene x_i occurs with probability $p_i^s(x_i)$ in the next population. $p_i^s(x_i)$ is the distribution of x_i in the selected parents. Thus new strings \mathbf{x} are generated according to the distribution

$$p(\mathbf{x}, t+1) = \prod_{i=1}^n p_i^s(x_i, t). \quad (24.2)$$

One can simplify the algorithm still more by directly computing the univariate marginal frequencies from the data. Then Equation 24.2 can be used to generate new strings. This method is used by the *Univariate Marginal Distribution Algorithm* (UMDA).

UMDA

STEP 0: Set $t \leftarrow 1$. Generate $N \gg 0$ points randomly.

STEP 1: Select $M \leq N$ points according to a selection method. Compute the marginal frequencies $p^s(x_i, t)$ of the selected set.

STEP 2: Generate N new points according to the distribution

$$p(\mathbf{x}, t+1) = \prod_{i=1}^n p_i^s(x_i, t). \text{ Set } t \leftarrow t + 1.$$

STEP 3: If termination criteria are not met, go to STEP 1.

UMDA is defined by $2n$ parameters, the marginal distributions $p_i(x_i)$. For proportionate selection the average fitness $\bar{f}(t) = \sum_x p(\mathbf{x}, t)f(\mathbf{x})$ is needed. We consider $\bar{f}(t)$ as a function which depends on $p_i(x_i)$. To emphasize this dependency we write

$$W(p_1(X_1=0), p_2(X_1=1), \dots, p_n(X_n=1)) := \bar{f}(t). \quad (24.3)$$

W formally depends on $2n$ parameters. $p_i(X_i=1)$ and $p_i(X_i=0)$ are considered as two independent parameters despite the constraint $p_i(X_i=0) = 1 - p_i(X_i=1)$. We abbreviate $p_i := p_i(X_i=1)$. If we insert $1 - p_i$ for $p_i(X_i=0)$ into W , we obtain \tilde{W} . \tilde{W} depends on n parameters. Now we can formulate the main theorem.

Theorem 1 *For infinite populations and proportionate selection UMDA changes the gene frequencies as follows:*

$$p_i(x_i, t+1) = p_i(x_i, t) \frac{\bar{f}_i(x_i, t)}{W(t)}, \quad (24.4)$$

where $\bar{f}_i(x_i, t) = \sum_{\mathbf{x}, X_i=x_i} f(\mathbf{x}) \prod_{j \neq i}^n p_j(x_j, t) = \frac{\partial W}{\partial p_i(x_i)}$. The equations can also be written as

$$p_i(x_i, t+1) = p_i(x_i, t) + p_i(x_i, t) \frac{\frac{\partial W}{\partial p_i(x_i)} - W(t)}{W(t)} \quad (24.5)$$

$$p_i(t+1) = p_i(t) + p_i(t)(1 - p_i(t)) \frac{\frac{\partial \tilde{W}}{\partial p_i}}{\tilde{W}(t)}. \quad (24.6)$$

Furthermore the average fitness W never decreases:

$$W(t+1) \geq W(t). \quad (24.7)$$

Proof: Equation 24.4 and the monotonicity of W has been proven in [10]. Equation 24.5 directly follows. We only have to prove Equation 24.6. Note that

$$p_i(t+1) - p_i(t) = p_i(t) \frac{\bar{f}_i(1, t) - \tilde{W}(t)}{\tilde{W}(t)}.$$

Obviously we have

$$\frac{\partial \tilde{W}}{\partial p_i} = \bar{f}(1, t) - \bar{f}(0, t).$$

From $p_i(t)\bar{f}_i(1, t) + (1 - p_i(t))\bar{f}_i(0, t) = \tilde{W}(t)$, we obtain

$$\bar{f}_i(1, t) - \tilde{W}(t) - (1 - p_i(t))\bar{f}_i(1, t) + (1 - p_i(t))\bar{f}_i(0, t) = 0.$$

This gives

$$\bar{f}_i(1, t) - \tilde{W}(t) = (1 - p_i(t)) \frac{\partial \tilde{W}}{\partial p_i}.$$

Inserting this equation into the difference equation gives Equation 24.6. \square

The above equations completely describe the dynamics of UMDA with proportionate selection. Equation 24.6 is especially suited for theoretical analysis. It has first been proposed by Wright [23]. Therefore it is called *Wright's equation* [15]. Wright's remarks are still valid today: "The appearance of this

formula is deceptively simple. Its use in conjunction with other components is not such a gross oversimplification in principle as has sometimes been alleged ... Obviously calculations can be made only from rather simple models, involving only a few loci or simple patterns of interaction among many similarly behaving loci... Apart from application to simple systems, the greatest significance of the general formula is that its form brings out properties of systems that would not be apparent otherwise." The extension of Wright's equation to multiple alleles can be found in [15].

The restricted application lies in the following fact. In general the difference equations need the evaluation of 2^n terms. The computational complexity can be drastically reduced if the fitness function has a special form. This is discussed next.

24.2.1 The average fitness and the landscape metaphor

Wright's equation transforms the discrete optimization problem with variables \mathbf{x} into a continuous optimization problem with variables \mathbf{p} . In the usual definition the computation of $W(\mathbf{p})$ is exponentially in n . This makes a numerical computation prohibitively. But there exists functions where $W(\mathbf{p})$ can be computed in polynomial number of operations. Let $\alpha = (\alpha_1, \dots, \alpha_n)$ with $\alpha_i \in \{0, 1\}$ be a multi-index. We define with $0^0 := 1$:

$$\mathbf{x}^\alpha := \prod_i x_i^{\alpha_i}.$$

Definition: The representation of a binary discrete function using the ordering according to function values is given by

$$f(\mathbf{x}) = f(0, \dots, 0)(1 - x_1) \cdots (1 - x_n) + \cdots + f(1, \dots, 1)x_1 \cdots x_n. \quad (24.8)$$

The representation using the ordering according to variables is

$$f(\mathbf{x}) = \sum_{\alpha} a_{\alpha} x^{\alpha}. \quad (24.9)$$

$\max\{|\alpha|_1 = \sum_i \alpha_i : a_{\alpha} \neq 0\}$ is called the order of the function. In both representations the function is linear in each variable x_i . The following two lemmas are obvious.

Lemma: *The two representations are unique. There exist a unique matrix A of dimension $2^n * 2^n$ such that*

$$a_{\alpha} = (Af)_{\alpha}.$$

Lemma: *$\tilde{W}(p) := \bar{f}(t)$ is an extension of $f(x)$ to the unit cube $[0, 1]^n$. There exist two representations for $\tilde{W}(p)$. These are given by*

$$\tilde{W}(p) = f(0, \dots, 0)(1 - p_1) \cdots (1 - p_n) + \cdots + f(1, \dots, 1)p_1 \cdots p_n \quad (24.10)$$

and

$$\tilde{W}(p) = \sum_{\alpha} a_{\alpha} p^{\alpha}. \quad (24.11)$$

If the function is given in analytical form (Equation 24.9) and the order of the function is bounded by a constant independent of n , then $\tilde{W}(p)$ can be computed in polynomial time. Equation 24.11 can also be used to compute the derivative. It is given by

$$\frac{\partial \tilde{W}(p)}{\partial p_i} = \sum_{\alpha | \alpha_i=1} a_{\alpha} p^{\alpha'}, \quad (24.12)$$

with $\alpha'_i = 0, \alpha'_j = \alpha_j$. If the function is of a low order the partial derivatives can be easily evaluated. This allows to compute the difference equations for p_i . In special cases the difference equation can even be solved analytically.

We will now characterize the attractors of UMDA. Let $x_i \in \{0, 1\}$, $S_i = \{q_i | \sum_{k \in \{0, 1\}} q_i(x_k) \leq 1; 0 \leq q_i(x_k) \leq 1\}$ and $S = \prod_i S_i$ the Cartesian product. Then $S = [0, 1]^n$.

Theorem 2 *The stable attractors of Wright's equation are at the corners of S , i.e. $p_i \in \{0, 1\} \quad i = 1, \dots, n$. In the interior there are only saddle points where $\text{grad } \tilde{W}(p) = 0$. The attractors are local maxima of $f(x)$ according to one bit changes. Wright's equation solves the continuous optimization problem $\text{argmax}\{\tilde{W}(p)\}$ in S by gradient ascent.*

Proof: W is linear in p_i , therefore it cannot have any local maxima in the interior. Points with $\text{grad } W(p) = 0$ are unstable fixpoints of UMDA.

We next show that boundary points which are not local maxima of $f(x)$ cannot be attractors. Without loss of generality, let the boundary point be $\hat{p} = (1, \dots, 1)$. We now consider an arbitrary neighbor, i.e. $p^* = (0, 1, \dots, 1)$. The two points are connected at the boundary by

$$p(z) = (1 - z, 1, \dots, 1) \quad z \in [0, 1].$$

We know that \tilde{W} is *linear* in the parameters p_i . Because $\tilde{W}(p^*) = f(0, 1, \dots, 1)$ and $\tilde{W}(\hat{p}) = f(1, \dots, 1)$ we have

$$\tilde{W}(p(z)) = f(1, \dots, 1) + z \cdot [f(0, 1, \dots, 1) - f(1, \dots, 1)]. \quad (24.13)$$

If $f(1, 1, \dots, 1) < f(0, 1, \dots, 1)$ then \hat{p} cannot be an attractor of UMDA. The mean fitness decreases with z . But this is impossible because of Theorem 1. \square

The difference equation has been derived for proportionate selection. Next we will investigate tournament and truncation selection.

24.3 THE SELECTION PROBLEM

Fitness proportionate selection is the undisputed selection method in population genetics. It is considered to be a model for *natural selection*. But the

selection strongly depends on the fitness values. When the population approaches an optimum, selection gets weaker and weaker, because the fitness values become similar. This behavior can be derived from Equation 24.5. The step-size decreases if p_i approaches 0 or 1. This slows down the convergence near the corners.

Therefore breeders of livestock use other selection methods. For large populations they mainly apply *truncation selection*. It works as follows. A truncation threshold $0 < \tau < 1$ is fixed. Then the τN best individuals are selected as parents for the next generation. These parents are then randomly mated.

We mainly use truncation selection in our algorithms. Another popular scheme is *tournament selection of size k*. Here k individuals are randomly chosen. The best individual is taken as parent. Unfortunately the mathematical analysis for both selection methods is more difficult than for proportionate selection. Analytical results for tournament selection have been first obtained by Mühlenbein [10].

24.3.1 Binary tournament selection

We model binary tournament selection as a game. Two individuals with genotype \mathbf{x} and \mathbf{y} “play” against each other. The one with the larger fitness gets a payoff of 2. If the fitness values are equal, both will win half of the games. This gives a payoff of 1. A loser gets no payoff. The game is defined by a *payoff matrix* with coefficients

$$a_{xy} = \begin{cases} 2 & f(\mathbf{x}) > f(\mathbf{y}) \\ 1 & f(\mathbf{x}) = f(\mathbf{y}) \\ 0 & f(\mathbf{x}) < f(\mathbf{y}). \end{cases}$$

One can easily show that

$$\sum_{\mathbf{x}} \sum_{\mathbf{y}} p(\mathbf{x}, t) a_{xy} p(\mathbf{y}, t) = 1. \quad (24.14)$$

After a round of tournaments the genotype frequencies are given by

$$p^s(\mathbf{x}, t+1) = p(\mathbf{x}, t) \sum_{\mathbf{y}} a_{xy} p(\mathbf{y}, t). \quad (24.15)$$

If we set

$$b(\mathbf{x}, t) = \sum_{\mathbf{y}} a_{xy} p(\mathbf{y}, t),$$

then the above equation is similar to proportionate selection using the function $b(\mathbf{x}, t)$ instead of $f(\mathbf{x})$. b can be seen as a frequency dependent fitness function. Furthermore the average of the population $\bar{b}(t) = \sum p(\mathbf{x}, t) b(\mathbf{x}, t)$ remains constant, $\bar{b}(t) \equiv 1$.

The difference equations for the univariate marginal frequencies can be derived in the same manner as for proportionate selection. They are given by

$$p_i(x_i, t+1) = p_i(x_i, t) \cdot \bar{B}_i(t) \quad (24.16)$$

$$\bar{B}_i(t) = \sum_{\mathbf{x}, X_i=x_i} b(\mathbf{x}, t) \prod_{j=1, j \neq i}^n p_j(x_j, t). \quad (24.17)$$

The difference equations are more complex than the equations for proportionate selection. \bar{B}_i is quadratic in $p(x_j)$. Even the simple linear function $\text{OneMax}(n) = \sum_i x_i$ leads to fairly complicated equations. For $p_i = p_j := p$ they have been computed in [10]. The difference equation is of order $2n$ in p , indicating that the exact computation becomes prohibitively. If the size of the tournament is increased, the equations get still more difficult.

For truncation selection it is impossible to derive difference equations for the univariate marginal frequencies. Therefore breeders have invented a macroscopic theory using mainly the average fitness and variance of the population.

24.3.2 The science of breeding

For a single trait the theory can be easily summarized. Starting with the fitness distribution, the *selection differential* $S(t)$ is introduced. It is the difference between the average of the selected parents and the average of the population:

$$S(t) = W(\mathbf{p}^s(t+1)) - W(\mathbf{p}(t)). \quad (24.18)$$

Similarly, the response $R(t)$ is defined as

$$R(t) = W(\mathbf{p}(t+1)) - W(\mathbf{p}(t)). \quad (24.19)$$

Next, a linear regression is done:

$$R(t) = b(t)S(t), \quad (24.20)$$

where $b(t)$ is called *realized heritability*. The selection differential can often be approximated by

$$S(t) \approx I_\tau V^{\frac{1}{2}}(t) \quad (24.21)$$

where I_τ is called the *selection intensity*. V is the variance of the fitness distribution. I_τ can be computed from order statistics [10]. Combining the two equations we obtain the famous equation for the response to selection:

$$R(t) \approx b(t)I_\tau V^{\frac{1}{2}}(t). \quad (24.22)$$

The most difficult problem is to estimate $b(t)$. Breeders use the estimate

$$b(t) \approx \frac{V_A(t)}{V(t)}, \quad (24.23)$$

where $V_A(t)$ denotes the additive genetic variance. For UMDA it is defined as

$$V_A(t) = \sum_{i=1}^n \sum_{x_i} p_i(x_i, t) \left(\frac{\partial W}{\partial p_i(x_i)} - W(t) \right)^2. \quad (24.24)$$

These equations are in depth discussed in [10]. The reader interested in quantitative genetics is referred to [5]. For the special case that all univariate marginal distribution are equal, i.e. $p_i := p$, the response to selection equation gives a difference equation for p . Thus it might be possible to obtain an analytical solution for $p(t)$.

We apply the theory to the linear function $\text{OneMax}(n) = \sum_i x_i$. The computation can be found in [10]. The equations and their solutions are given by:

Theorem 3 *If in the initial population all univariate marginal frequencies are identical to $p_0 > 0$, then we obtain for UMDA and OneMax proportionate selection:*

$$R(t) = 1 - p(t) \quad (24.25)$$

$$p(t) = 1 - (1 - p_0)(1 - \frac{1}{n})^t \quad (24.26)$$

truncation selection:

$$R(t) \approx I_\tau \sqrt{np(t)(1 - p(t))} \quad (24.27)$$

$$p(t) \approx 0.5 \left(1 + \sin \left(\frac{I_\tau}{\sqrt{n}} t + \arcsin(2p_0 - 1) \right) \right) \quad (24.28)$$

tournament selection:

$$\begin{aligned} R(t) = & np(1 - p) \left(2 \sum_{k=1}^n \sum_{j=0}^{k-1} \binom{n-1}{k-1} \binom{n}{j} p^{k+j-1} (1-p)^{2n-k-j-1} \right. \\ & \left. + \sum_{k=1}^{n-1} \binom{n-1}{k-1} \binom{n}{k} p^{2k-1} (1-p)^{2n-2k-1} - \sum_{j=0}^{2n-2} p^j \right) \end{aligned} \quad (24.29)$$

$$R(t) \approx 0.564 \sqrt{np(t)(1 - p(t))}. \quad (24.30)$$

The difference equation for proportionate selection is easy to solve. The solution for the approximate equation for truncation selection is also straightforward. But the analytical solution for the exact difference equation for binary tournament selection seems to be very difficult. Therefore we use an approximation. It has been shown in [10] that binary tournament selection can be approximated by truncation selection with selection intensity $I_2 = 0.564$. The analytical solutions almost perfectly match the results obtained from actual UMDA runs (see Figure 24.1).

With proportionate selection the population needs a long time to approach the optimum. In contrast, truncation selection and tournament selection lead to much faster convergence. p increases almost linearly until near the optimum. The exact difference equation for binary tournament selection has p^{2n} as the largest exponent.

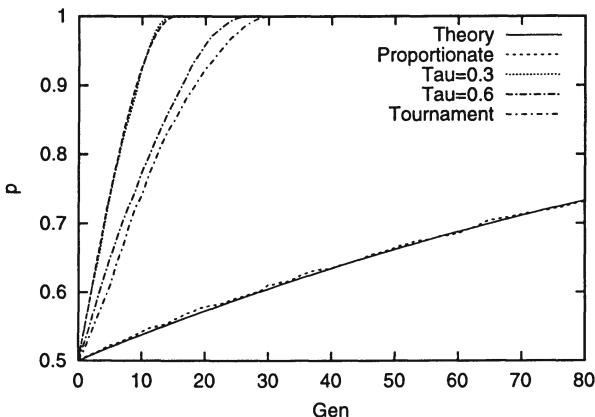


Figure 24.1 Comparison of selection methods for OneMax(128).

The theory of breeding uses macroscopic variables, the average and the variance of the population. But we have derived only one equation, the response to selection equation. We need a second equation connecting the average fitness and the variance in order to be able to compute the time evolution of the average fitness and the variance. There have been many attempts in population genetics to find a second equation. But all equations assume that the variance of the population continuously decreases. This is not the case for arbitrary fitness functions. Recently Prügel-Bennet and Shapiro [19] have independently proposed to use moments for describing genetic algorithms. They apply methods of statistical physics to derive equations for higher moments for special fitness functions.

24.4 OPTIMIZATION METHODS USING UNIVARIATE DISTRIBUTIONS

The importance of using univariate marginal distributions in evolutionary algorithms has been independently discovered by several researchers. We shortly discuss *PBIL* of Baluja and Caruana [2] and *ant colony optimization* (ACO) by Dorigo & Di Caro [4]. A third approach, using Wright's equation, is investigated in the next subsection.

PBIL does not use strict Darwinian selection in populations, but the probabilities are updated according to

$$p_i(x_i, t + 1) = p_i(x_i, t) + \lambda(p^s(x_i, t) - p_i(x_i, t)). \quad (24.31)$$

A new string \mathbf{x} is generated in UMDA manner

$$p(\mathbf{x}, t + 1) = \prod_{i=1}^n p_i(x_i, t + 1). \quad (24.32)$$

The convergence speed of this algorithm critically depends on λ . For $\lambda = 0$ we have a random search, for $\lambda = 1$ we obtain UMDA. The smaller λ , the slower the convergence speed. This problem is discussed in [10]. Our numerical experiments indicate that the UMDA selection method is to be preferred, because it is very difficult to choose λ for a given problem.

Univariate marginal distributions are also used in ant colony optimization [4]. Each state i has a neighborhood $N(i)$. The "ant" k uses a probability \tilde{p}_{ij}^k to move from state i to state j . The probability is given by

$$\tilde{p}_{ij}^k := \begin{cases} \frac{\tau_{ij}(t)}{\sum_{j \in N(i)} \tau_{ij}(t)} & \text{for } j \in N(i) \\ 0 & \text{for } j \notin N(i). \end{cases} \quad (24.33)$$

The variable τ_{ij} is updated according to

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \Delta\tau_{ij}. \quad (24.34)$$

τ_{ij} estimates the desirability of going from state i to state j . It is updated after the ant has created a solution. The update depends on the fitness of the solution.

ACO is similar to UMDA, but it is a more local algorithm. $p_{ij} = p_i(x_j)$ plays the role of our univariate marginal distributions. If all states j are contained in the neighborhood $N(i)$ then we would have an UMDA algorithm with integer variables (multiple alleles). But ACO uses problem dependent neighborhoods. These allow the application to combinatorial permutation optimization problems like the traveling salesman problem TSP or the quadratic assignment problem. Feasible solutions for these problems are only the permutations of $\{1, \dots, n\}$. We take TSP as an example. If city l is chosen to be at a certain place of the tour, it is not allowed to be chosen again. ACO solves this problem by setting all \tilde{p}_{il}^k with $i > l$ to 0. The other probabilities \tilde{p}_{ij}^k are renormalized, so that the sum of the probabilities of all feasible moves is 1. Thus ACO constructs feasible solutions. The same technique could be applied to UMDA. But now the search points depend on an ordering of the variables. Thus the theory presented in the previous sections has to be extended.

24.4.1 The system dynamics approach to optimization

Theorem 1 can be used to define purely mathematical optimization algorithms. Instead of generating a population of points and selecting promising points, we can use Wright's equation to update the univariate marginal distributions. We have shown by Theorem 2 that this method optimizes in a certain sense the continuous optimization problem $\operatorname{argmax} \tilde{W}(\mathbf{p})$. But Wright's equation is not the only possibility. Voigt [22] proposed the following difference equation.

Definition: The *Discrete Diversified Replicator Equation* DDRP is given by

$$p_i(x_k)(t+1) - p_i(x_k)(t) = p_i(x_k)(t) \frac{f_{ik}(\mathbf{p}) - \sum_{x_k} p_i(x_k) f_{ik}(\mathbf{p})}{\sum_{x_k} p_i(x_k) f_{ik}(\mathbf{p})}. \quad (24.35)$$

V_i is called a *potential* of DDRP if $\partial V_i / \partial p_i(x_k) = f_{ik}(\mathbf{p})$. Wright's equation and DDRP are connected by the following theorem.

Theorem 4 *If the average fitness $W(\mathbf{p})$ is used as potential, then Wright's equation and the Discrete Diversified Replicator Equation are identical.*

Proof: The average fitness is defined as

$$W(\mathbf{p}) = V(\mathbf{p}) = \sum_x a_x \prod_{i=1}^n p_i(x_i).$$

We compute the derivatives

$$\begin{aligned} \frac{\partial V(\mathbf{p})}{\partial p_i(1)} &= \sum_{x|x_i=1} a_x \prod_{j \neq i}^n p_j(x_j) \\ \frac{\partial V(\mathbf{p})}{\partial p_i(0)} &= \sum_{x|x_i=0} a_x \prod_{j \neq i}^n p_j(x_j). \end{aligned}$$

Obviously

$$p_i(1) \frac{\partial V}{\partial p_i(1)} + p_i(0) \frac{\partial V}{\partial p_i(0)} = V(\mathbf{p}).$$

The conjecture now follows from the proof of Wright's equation. \square

We recently discovered that Baum and Eagon [3] have proven a discrete maximum principle for certain instances of the DDRP and for Wright's equation.

Theorem 5 (Baum-Eagon) *Let $V(\mathbf{p})$ be a polynomial with nonnegative coefficients homogeneous of degree d in its variables $p_i(x_j)$ with $p_i(x_j) \geq 0$ and $\sum_{x_j} p_i(x_j) = 1$. Let $\mathbf{p}(t+1)$ be the point given by*

$$p_i(x_j, t+1) = \frac{p_i(x_j, t) \frac{\partial V}{\partial p_i(x_j)}}{\sum_{x_k} p_i(x_k) \frac{\partial V}{\partial p_i(x_k)}}. \quad (24.36)$$

The derivatives are taken at $\mathbf{p}(t)$. Then $V(\mathbf{p}(t+1)) > V(\mathbf{p}(t))$ unless $\mathbf{p}(t+1) = \mathbf{p}(t)$.

Equation 24.36 is exactly DDRP with a potential V . Thus the DDRP could be called the Baum-Eagon equation. From the above theorem the discrete maximum principle for Wright's equation follows by setting $V = W$ and $d = n$. Thus the average fitness is used as a potential, which is homogeneous of degree n . So both equations can be used for optimization.

The maximum principle shows that the potential never decreases. Early applications of the DDRP to combinatorial optimization can be found in [22, 11]. But there is a problem: the difference equations are deterministic. For

difficult optimization problems, there exists a huge number of attractors, each with a corresponding attractor region. If the iteration starts at a point within the attractor region, it will converge to the corresponding attractor at the boundary. But if the iteration starts at points which lie at the boundary of two or more attractors, i.e. on the separatrix, the iteration will be confined to the separatrix. The deterministic system cannot decide for one of the attractors.

UMDA with a finite population does not have a sharp boundary between attractor regions. We model this behavior by introducing randomness into the difference equation. The new value $p_i(x_j, t + 1)$ is randomly chosen from the interval

$$[(1 - c)p_i'(x_j, t + 1), (1 + c)p_i'(x_j, t + 1)]$$

where $p_i'(x_j, t + 1)$ is determined by the deterministic equation. c is a small number. For $c = 0$ we obtain the deterministic equation. In order to prohibit too early convergence, we do not allow the boundary values $p_i = 0$ or $p_i = 1$. We use $p_i = p_{min} > 0$ and $p_i = 1 - p_{min}$ instead.

A third extension is the computation of the solution. All dynamic equations presented use variables, which can be interpreted as probabilities. Thus instead of waiting that the dynamic system converges to some boundary point, we terminate the iteration at a suitable time and generate a set of solutions. Thus, given the values for $p_i(x_i)$ we generate points \mathbf{x} according to the UMDA distribution $p(\mathbf{x}) = \prod_{i=1}^n p_i(x_i)$.

We can now formulate a family of optimization algorithms, based on difference equations (DIFFOPT).

DIFFOPT

STEP 0: Set $t \leftarrow 0$ and $p_i(x_j, 0) = 0.5$, input c , p_{min} .

STEP 1: Compute $p_i'(x_j, t + 1)$ according to a dynamic difference equation.

STEP 2: Compute randomly $p_i(x_j, t + 1)$ in the interval $(1 - c)p_i'(x_j, t + 1), (1 + c)p_i'(x_j, t + 1)$. If $p_i(x_j, t + 1) < p_{min}$ then $p_i(x_j, t + 1) = p_{min}$. If $p_i(x_j, t + 1) > 1 - p_{min}$ then $p_i(x_j, t + 1) = 1 - p_{min}$. Set $t \leftarrow t + 1$.

STEP 3: If termination criteria are not met, go to STEP 1.

STEP4: Generate N solutions according to $p(\mathbf{x}, t) = \prod_{i=1}^n p_i(x_i, t)$ and compute $\max f(\mathbf{x})$ and $\arg\max f(\mathbf{x})$.

DIFFOPT is not restricted to Wright's equation or DDRP. We propose a third equation. Its rationale is as follows. From the analysis of UMDA we know that Wright's equation models proportionate selection. But this method converges very slowly when approaching the boundary. We have not been able to derive dynamic equations for truncation selection. Therefore we experimented with a number of faster versions of Wright's equation. The following difference equation was ultimately chosen.

Definition: $F - Wright(\alpha)$ (Fast Wright) is defined by the following difference equation

$$p_i(x_i, t + 1) = p_i(x_i, t) + sign(\Delta) * \exp(\alpha \ln abs(\Delta)) \quad (24.37)$$

$$\Delta = p_i(x_i, t)(1 - p_i(x_i, t)) \frac{\frac{\partial \tilde{W}}{\partial p_i(x_i)}}{\tilde{W}(\mathbf{p})}. \quad (24.38)$$

If a value outside the interval $(p_{min}, 1 - p_{min})$ is generated, we just set the value to the corresponding boundary value of the interval. For $\alpha = 1$ we obtain Wright's equation. We usually set $\alpha = 0.5$. The reason for this choice is that we wanted a difference equation which resembles as much as possible *truncation selection*. If we take the fitness function *OneMax*, we obtain for $F - Wright(0.5)$ the difference equation

$$p(t + 1) - p(t) = \sqrt{p(t)(1 - p(t))/(np(t))} = \sqrt{(1 - p(t)/n)}. \quad (24.39)$$

This equation is similar to the approximate equation we have computed for UMDA with truncation selection. Only the multiplication by p is missing. This means that $F - Wright$ will normally converge faster than UMDA with truncation selection.

A comparison of the three methods can be found in [15]. In this paper we will only discuss some numerical results for UMDA.

24.5 NUMERICAL RESULTS FOR UMDA

This section solves the problem put forward in [9]: to understand the class of problems for which genetic algorithms are most suited, and in particular, for which they will outperform other search algorithm. A detailed comparison between theory and numerical results can be found in [14].

24.5.1 Multi-modal functions suited for optimization by UMDA

Equation 24.5 shows that UMDA performs a gradient ascent in the landscape given by W . This helps our search for functions best suited for UMDA. We take the function Saw as example. The definition of the function can be extrapolated from Figure 24.2. In $Saw(n, m, k)$, n denotes the number of bits and $2m$ the distance from one peak to the next. The highest peak is multiplied by k (with $k \leq 1$), the second highest by k^2 , then k^3 and so on. The landscape is very rugged. In order to get from one local optimum to another one, one has to cross a deep valley.

We used UMDA with truncation selection for the solution. In order to graphically show the results, we assume that $p_i = p_j := p$. Then the average fitness depends on a single variable p . In Figure 24.3 the results are displayed. The $\tilde{W}(p)$ landscape is fairly smooth. In the simulation two truncation thresholds, $\tau = 0.05$ and $\tau = 0.01$, have been used. For $\tau = 0.05$ the probability p stops at the local maximum for $\tilde{W}(p)$. It is approximately $p = 0.78$. For $\tau = 0.01$ UMDA is able to converge to the optimum $p = 1$. It does so by even going one step downhill in the $W(p)$ landscape.

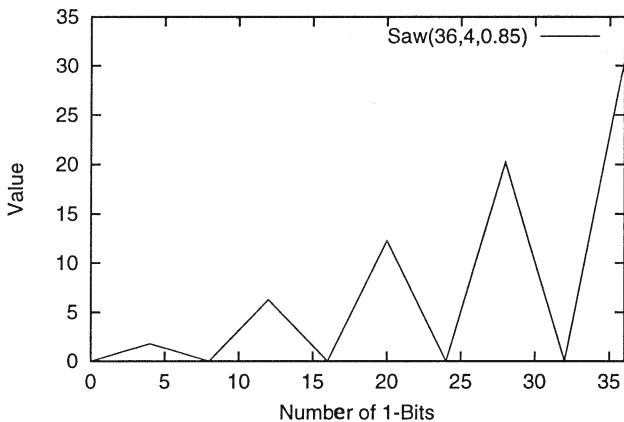


Figure 24.2 Definition of $\text{Saw}(36,4,0.85)$.

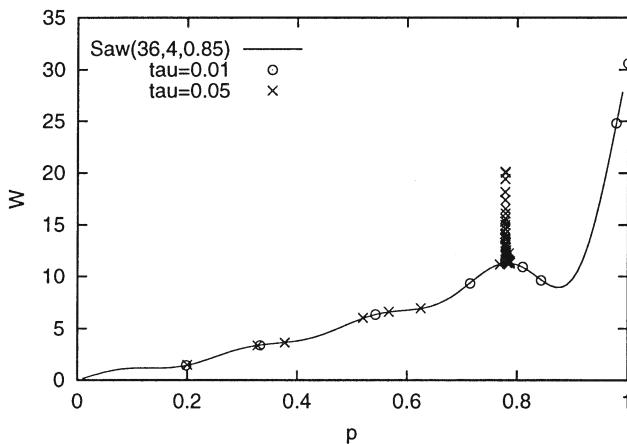


Figure 24.3 Results for two different truncation thresholds.

24.5.2 Deceptive functions misleading UMDA

There are many optimization problems where UMDA is misled. UMDA will converge to local optima, because it does not use correlations between the variables. We demonstrate this problem by a deceptive function. We use the definition

$$\text{Decep}(\mathbf{x}, k, a) := \begin{cases} k - 1 - |\mathbf{x}|_1 & 0 \leq |\mathbf{x}|_1 < k \\ a & |\mathbf{x}|_1 = k \quad a \geq k, \end{cases} \quad (24.40)$$

where $|\mathbf{x}|_1 = \sum x_i$. The global maximum is isolated at $\mathbf{x} = (1, \dots, 1)$. A deceptive function of order n is a needle in a haystack problem. This is far too difficult to optimize for any optimization method. We simplify the optimization

problem by adding l distinct $Decep(k)$ -functions to give a fitness function of size $n = l * k$. This function, called $Decep(n, k, a)$, is also deceptive. The local optimum $x = (0, \dots, 0)$ is surrounded by good fitness values, whereas the global optimum is isolated:

$$Decep(n, k, a) = \sum_{i=1, k+1, \dots}^n Decep((x_i, x_{i+1}, \dots, x_{i+k-1}), k, a). \quad (24.41)$$

If the fitness of the optimum increases, we expect that UMDA will converge to the optimum. We will confirm this behavior for $Decep(n, 4, a)$. For our theoretical analysis we have to compute

$$\tilde{W}(\mathbf{p}) = \sum_{i=1, 5, \dots}^n (3 - p_i - p_{i+1} - p_{i+2} - p_{i+3} + (a + 1) \cdot p_i p_{i+1} p_{i+2} p_{i+3}). \quad (24.42)$$

From this equation the partial derivatives can easily be computed. We assume $p_i = p_j := p$ and just compute one derivative:

$$\frac{\partial \tilde{W}}{\partial p_1} = -1 + (a + 1) \cdot p_2 p_3 p_4. \quad (24.43)$$

From Equation 24.6 we obtain the difference equation

$$p(t + 1) = p(t) + p(t) \cdot (1 - p(t)) \frac{(a + 1) \cdot p(t)^3 - 1}{l((a + 1) \cdot p(t)^4 - 4p(t) + 3)}, \quad (24.44)$$

where $l = n/4$. $p(t)$ decreases for $p(0) = 0.5$ and $a < 7$. Thus, we obtain:

Corollary: *If the initial population is generated randomly, i.e. $p(0) = 0.5$, then UMDA converges to $p(t) \rightarrow 0$ for $a < 7$. For $a > 7$ UMDA converges to $p(t) \rightarrow 1$ if started randomly.*

In Figure 24.4 we show the average fitness $W(p)$ and an actual UMDA run for $a = 4$. Starting at $p(0) = 0.5$ UMDA converges to the local optimum $\mathbf{x} = (0, \dots, 0)$ because $5p(t)^3 - 1$ is negative. The local minimum of $W(p)$ is at about $p = 0.585$. UMDA would converge to the global optimum if it starts nearer to the optimum, e.g. $p(0) \geq 0.6$. Also shown is a curve which arises from FDA, an algorithm using fourth order marginal distributions. This algorithm always converges to the global optimum, independent of the initial population. Further numerical results can be found in [14]. FDA is discussed next.

24.6 FDA – THE FACTORIZED DISTRIBUTION ALGORITHM

UMDA converges to the global optima in quite a number of interesting cases. Convergence has been proven for generalized linear functions $f(\mathbf{x}) = \sum_i g(x_i)$. For other functions like *Saw*, convergence depends on the landscape defined by $W(\mathbf{p})$. UMDA might easily be deceived as we have shown with the deceptive function.

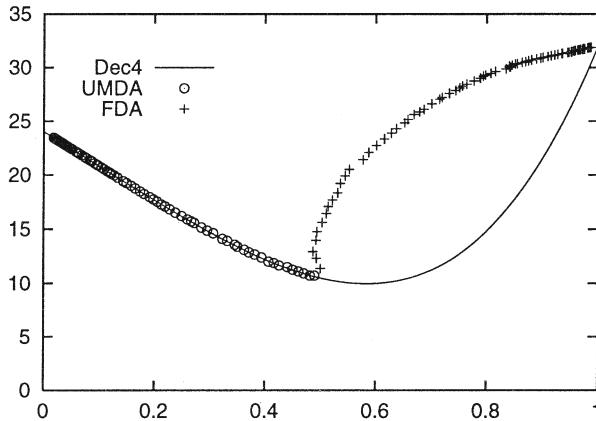


Figure 24.4 Average fitness $W(p)$ for UMDA and FDA for Decep(36,4,4).

Convergence can be proven, if UMDA uses more complex search distributions, which capture the structure of the function. Before we define such a search distribution, we introduce a special distribution needed for the theoretical analysis.

Definition: Let $0 \leq \beta < \infty$ be the inverse temperature. Then the *weighted Boltzmann distribution* is defined by

$$p(\mathbf{x}, \beta) = \frac{p_0(\mathbf{x})e^{\beta f(\mathbf{x})}}{Z_\beta}, \quad (24.45)$$

where $Z_\beta = \sum_{\mathbf{x}} p_0(\mathbf{x})e^{\beta f(\mathbf{x})}$ is the partition function. $p_0(\mathbf{x})$ is the distribution at $\beta = 0$. For $p_0(\mathbf{x}) = 1/2^n$ we obtain the Boltzmann distribution. *Boltzmann selection* with parameter β_s changes a distribution defined by $p(\mathbf{x}, t)$ according to

$$p^s(\mathbf{x}, \beta) = p(\mathbf{x}, \beta) \frac{e^{\beta_s f(\mathbf{x})}}{\sum_{\mathbf{x}} p(\mathbf{x}, \beta) e^{\beta_s f(\mathbf{x})}}. \quad (24.46)$$

Boltzmann selection has the following important property.

Lemma: Let $p(\mathbf{x}, \beta)$ be a weighted Boltzmann distribution. After Boltzmann selection with parameter β_s we again have a weighted Boltzmann distribution with $\tilde{\beta} = \beta + \beta_s$, i.e.

$$p^s(\mathbf{x}, \beta) = p(\mathbf{x}, \beta + \beta_s). \quad (24.47)$$

We can now define a conceptual algorithm BEDA, the Boltzmann Estimation Distribution Algorithm.

BEDA

STEP 0: Set $t \leftarrow 0$, $\beta = 0$. Generate $N \gg 0$ points randomly.

STEP 1: Set $t \leftarrow t + 1$. Perform Boltzmann selection with parameter β_t , giving $p^s(\mathbf{x}, \beta) = p(\mathbf{x}, \beta + \beta_t)$. Set $\beta \leftarrow \beta + \beta_t$.

STEP 2: Generate N new points according to the distribution $p(\mathbf{x}, \beta)$.

STEP 3: If termination criteria are not met, go to STEP 1.

The reader should have noticed that BEDA and simulated annealing are related. BEDA can be seen as an idealized simulated annealing algorithm insofar as BEDA generates the search points directly from a Boltzmann distribution. The following theorem is identical to the convergence theorem for time homogeneous simulated annealing [1].

Theorem 6 *Let G^* be the set of all global optima. Let $\beta_0 = 0, \beta_i > 0$, $i = 1, 2, \dots$ be a selection schedule. Let*

$$\beta^t = \sum_{i=1}^{t-1} \beta_i.$$

If $\lim_{t \rightarrow \infty} \beta^t = \infty$, the population converges to the global optima, i.e.

$$\lim_{t \rightarrow \infty} p(x, \beta^t) = \begin{cases} \frac{1}{|G^*|} & x \in G^* \\ 0 & \text{else.} \end{cases} \quad (24.48)$$

We call BEDA conceptual, because it needs the computation of $2^n - 1$ values of $p(\mathbf{x})$. We now transform BEDA into a practical algorithm. This requires that the distribution $p(\mathbf{x})$ can be defined by a polynomial number of parameters. Thus we are lead to the problem of *factorization of the distribution*. In order to deal with this problem, we need the concepts of conditional probability and conditional independence.

Definition. The *conditional probability* $p(x|y)$ is defined for $p(y) > 0$ as

$$p(x|y) = \frac{p(x, y)}{p(y)}. \quad (24.49)$$

Two variables X and Y are conditional independent given Z , if for all instances

$$p(x, y|z) = p(x|z)p(y|z). \quad (24.50)$$

An equivalent definition of conditional independence is

$$p(x|y, z) = p(x|z).$$

Theorem 7 (Bayesian factorization) *Each distribution can be factored into*

$$p(\mathbf{x}) = \prod_{i=1}^n p(x_i|pa_i) \quad (24.51)$$

where $PA_i \subset \{X_1, \dots, X_{i-1}\}$ are called the parents of variable X_i .

Proof: From the definition of conditional probabilities we obtain

$$p(\mathbf{x}) = p(x_1) \prod_{i=2}^n p(x_i|x_1, \dots, x_{i-1}).$$

If X_i and $\{X_1, \dots, X_{i-1}\} \setminus PA_i$ are conditional independent given PA_i , we can simplify

$$p(x_i|x_1, \dots, x_{i-1}) = p(x_i|pa_i).$$

□

The Bayesian factorization can be represented as a directed graph. In the context of graphical models the graph and the conditional probabilities are called a Bayesian network [8, 6]. The factorization is used by the Factorized Distribution Algorithm FDA.

FDA

STEP 0: Set $t \leftarrow 0$. Generate $N \gg 0$ points randomly.

STEP 1: Selection.

STEP 2: Compute the conditional probabilities $p^s(x_i|pa_i, t)$ using the selected points.

STEP 3: Generate a new population according to $p(\mathbf{x}, t+1) = \prod_{i=1}^n p^s(x_i|pa_i, t)$.

STEP 4: If termination criteria is met, FINISH.

STEP 5: Set $t \leftarrow t + 1$. Go to STEP 2.

If the factorization does not contain conditional marginal distributions, but only marginal distributions, FDA can be theoretically analyzed. The difference equations of the marginal distributions are of the form given by Equation 24.5 [12]. If the function $f(\mathbf{x})$ is decomposable, then an exact factorization can be computed [16].

The amount of computation for FDA depends on the size of the population (N) and the number of variables used for the factors. The computational complexity can be bounded for one generation by

$$\text{cost} \leq \sum_{i=1}^n 2^{|PA_i|} N \leq nN2^k \quad (24.52)$$

where $k = \max_i |PA_i|$ is the maximum number of parents. There exist many problems where k is bounded independent from n . The interested reader is referred to [16]. In this case the computational complexity of FDA is polynomial. Furthermore from the BEDA convergence theorem we can infer that FDA with Boltzmann selection converges to the global optima if the size of the population N is large enough. The scaling of the numerical complexity of FDA is discussed in [13].

We next derive an adaptive Boltzmann selection schedule.

24.7 RESPONSE TO SELECTION FOR FDA WITH BOLTZMANN SELECTION

The theoretical analysis of FDA turns out to be very difficult. The dynamical equations for the conditional marginal distributions are much more difficult than for UMDA. But for Boltzmann selection some interesting results can be obtained.

Let $p(\mathbf{x}, \beta)$ be a weighted Boltzmann distribution, $W_f(\beta) = \sum_x p(\mathbf{x}, \beta) f(x)$ the average fitness of the population. Then we have

$$W_f(\beta) = \frac{\frac{dZ_\beta}{d\beta}}{Z_\beta}, \quad (24.53)$$

where Z_β is the partition function of the Boltzmann distribution. With this equation the next theorem can be proven.

Theorem 8 *The response to selection for Boltzmann selection is given for $\tilde{\beta} = \beta + \beta_s$*

$$R = W_f(\tilde{\beta}) - W_f(\beta) = \sum_{i \geq 1} \frac{(\tilde{\beta} - \beta)^i}{i!} M_{i+1}^c(\beta) \quad (24.54)$$

where $M_i^c = \sum_x (f(\mathbf{x}) - W_f(\beta))^i p(\mathbf{x}, \beta)$ are the centered moments.

Proof: The proof uses the Taylor expansion for $W_f(\beta)$. Here we prove just the first term.

$$\begin{aligned} \frac{dW_f(\beta)}{d\beta} &= \frac{Z''(\beta)Z(\beta) - Z'(\beta)^2}{Z(\beta)^2} \\ &= \frac{\sum_x f(\mathbf{x})^2 e^{\beta f(\mathbf{x})}}{Z(\beta)} - W_f(\beta)^2 \\ &= \sum_x (f(\mathbf{x}) - W_f(\beta))^2 p(\mathbf{x}, \beta) = M_2^c(\beta). \end{aligned}$$

The theorem itself can be proven by induction. \square

Corollary: *For Boltzmann selection the response is given approximately by*

$$R = W_f(\tilde{\beta}) - W_f(\beta) \approx (\tilde{\beta} - \beta) \sigma_f^2(\beta). \quad (24.55)$$

$\sigma_f(\beta)$ is the standard deviation.

The above equation gave us an idea how to compute an adaptive selection schedule. We try to make the response of Boltzmann selection similarly to the response of truncation selection.

Definition: The standard deviation Boltzmann selection schedule SDS is defined by

$$\beta_s = \frac{c}{\sigma_f(\beta)}. \quad (24.56)$$

For SDS the response to selection is approximately given by

$$R \approx c \cdot \sigma_f(\beta). \quad (24.57)$$

We can now compare the above equation with the response to selection Equation (24.22) obtained for truncation selection. The comparison shows that the constant c plays the role of $b(t)I_\tau$ used for truncation selection. We normally set $c = 1$.

We now investigate SDS for linear functions

$$\text{Linear}(\mathbf{x}) = \sum_{i=1}^n \alpha_i x_i. \quad (24.58)$$

The Boltzmann distribution can easily be calculated and is given by

$$Z_f(\beta) = \prod_{i=1}^n (1 + e^{\beta \alpha_i}) \quad (24.59)$$

and

$$p_i(\beta) := p_\beta(X_i=1) = \frac{e^{\beta \alpha_i}}{1 + e^{\beta \alpha_i}}. \quad (24.60)$$

The variance is just the sum of the variance of the factors and we have

$$Var_f(\beta) = \sum_{i=1}^n \frac{\alpha_i^2 e^{\beta \alpha_i}}{(1 + e^{\beta \alpha_i})^2} = \sum_{i=1}^n \alpha_i^2 p_i(\beta)(1 - p_i(\beta)). \quad (24.61)$$

Thus SDS is given by

$$\beta(t+1) = \beta(t) + \frac{c}{\sqrt{\sum_i \alpha_i^2 p_i(\beta)(1 - p_i(\beta))}}. \quad (24.62)$$

By differentiating Equation 24.60 we get

$$\begin{aligned} \frac{dp_i(\beta)}{dt} &= \frac{\alpha_i e^{\beta \alpha_i} (1 + e^{\beta \alpha_i}) \beta' - e^{\beta \alpha_i} \alpha_i e^{\beta \alpha_i} \beta'}{(1 + e^{\beta \alpha_i})^2} \\ &= p_i(\beta)(1 - p_i(\beta)) \alpha_i \frac{d\beta}{dt}. \end{aligned} \quad (24.63)$$

Therefore we obtain the differential equation

$$\frac{dp_i(\beta)}{dt} = c \cdot \frac{p_i(\beta)(1 - p_i(\beta)) \alpha_i}{\sqrt{\sum_i \alpha_i^2 p_i(\beta)(1 - p_i(\beta))}}. \quad (24.64)$$

For *OneMax* we have $\alpha_i = 1$. In this case all marginal frequencies are equal to p_β . We obtain the differential equation

$$\frac{dp_\beta}{dt} = c \sqrt{p_\beta(1 - p_\beta)/n}. \quad (24.65)$$

This equation has been used as an approximation for truncation selection before (see Equation 24.28). Note that the equation is exact for SDS! Boltzmann selection has another appealing numerical behavior - the average fitness always increases.

Theorem 9 *For $f(x) \neq \text{const}$ we have*

$$\tilde{\beta} > \beta \implies W_f(\tilde{\beta}) > W_f(\beta). \quad (24.66)$$

Proof: Obviously $p(\mathbf{x}, \beta) > 0$. Furthermore from the above theorem

$$\frac{dW_f(\beta)}{d\beta} = \sigma_f^2(\beta) > 0$$

for $f(x) \neq \text{const}$. Because the derivative is positive, $W_f(\beta)$ increases. \square

The similarity of SDS and truncation selection can be demonstrated with the following property.

Lemma: *The search distributions created by an SDS schedule are invariant with respect to positive linear transformations. This means the following. Let $\tilde{f}(\mathbf{x}) = c \cdot f(\mathbf{x}) + d$ with $c > 0$. Let $0, \beta_1, \beta_2, \dots$ be the SDS schedule for $f(\mathbf{x})$. Then $0, \beta_1/c, \beta_2/c, \dots$ is the corresponding SDS schedule for $\tilde{f}(\mathbf{x})$. Furthermore*

$$p_f(\mathbf{x}, \beta_i) = p_{\tilde{f}}(\mathbf{x}, \beta_i/c).$$

This result easily follows from the identity $\sigma_{\tilde{f}}(\beta) = c \cdot \sigma_f(\beta)$. It shows that the actual search distributions are independent from the scaling factor c and a translation d . This is a great advantage in our opinion. Optimization of $f(\mathbf{x})$ and $c \cdot f(\mathbf{x})$ is equally difficult. Optimization algorithms should behave identical for both problems. Invariance under positive linear transformations is also an important feature of truncation selection. It is not fulfilled for proportionate selection! The SDS schedule makes Boltzmann selection now very attractive, both theoretically and numerically.

24.8 FINITE POPULATIONS

Our theoretical analysis assumes an infinite population. We have shown that the theory exactly describes UMDA using large populations. But with an infinite population one should select as severely as possible. The optimum is already generated in the initial population!

The analysis is different if we investigate the problem to obtain the optimum with as few as possible function evaluations. We have obviously the following qualitatively behavior. We can find the optimum in one step of selection, if the population is large enough so that the optimum is contained in the initial population. This means that the population size is exponential ($O(2^n)$). If we select very weakly, then genetic drift is a problem. Genes are randomly fixed to the wrong value. Thus it again requires a huge population size not

to converge prematurely. In between the two extremes there might exist a moderate selection scheme which needs a population size which is polynomially bounded. In order to minimize the number of function evaluations we have to minimize the product of number of steps and the size of the population needed for convergence to the optimum. We will very shortly discuss this problem here, the interested reader is referred to [13].

In finite populations convergence of UMDA or FDA can only be probabilistic. Since UMDA is a specialized FDA algorithm, it is sufficient to discuss FDA. This section is extracted from [13].

Definition: Let ϵ be given. Let $P_{conv}(N)$ denote the probability that FDA with a population size of N converges to the optima. Then the critical population size is defined as

$$N^*(\epsilon) = \min_N P_{conv}(N) \geq 1 - \epsilon. \quad (24.67)$$

If FDA with a finite population does not converge to an optimum, then at least one gene is fixed to a wrong value. The probability of fixation is reduced if the population size is increased. We obviously have for FDA

$$N_1 \leq N_2 : \quad P_{conv}(N_1) \leq P_{conv}(N_2).$$

We discuss the fixation problem with a special function called *Int*. $Int(\mathbf{x})$ gives the integer value of the binary representation.

$$Int(\mathbf{x}, n) = \sum_{i=1}^n 2^{i-1} x_i. \quad (24.68)$$

The function has 2^n different fitness values. We show the cumulative fixation probability in Table 24.1 for $Int(16)$. Fixation means that at least one gene is fixed to 0. The fixation probability is larger for stronger selection. For truncation selection with $\tau = 0.25$ the fixation probability is 0.074 for $N = 80$.

Boltzmann selection with a fixed selection schedule with $\beta_s = 0.01$ is still very strong for the fitness distribution given by $Int(16)$. For $N = 700$ the largest fixation probability is still at the first generation. In comparison, the adaptive Boltzmann schedule SDS has a total fixation probability of 0.084 for a population size of $N = 100$.

This example shows that Boltzmann selection in finite populations critically depends on a good annealing schedule. Normally we run FDA with truncation selection. This selection method is a good compromise. But Boltzmann selection with SDS schedule is of comparable performance.

Estimates for the necessary size of the population can also be found in [7]. But they use a weaker performance definition. The goal is to have a certain percentage of the bits of the optimum in the final population. Furthermore their result is only valid for fitness function which are approximately normally distributed.

The danger of fixation can further be reduced by a technique very popular in Bayesian statistics. This is discussed in the next section.

t	$\tau = 0.25$	$\tau = 0.5$	$\tau = 0.25$	$\tau = 0.5$	Boltz.	SDS
	$N = 30$	$N = 30$	$N = 80$	$N = 60$	$N = 700$	$N = 100$
1	0.0955	0.0035	0.0	0.0	0.0885	0.0
2	0.4065	0.0255	0.0025	0.0095	0.1110	0.0
3	0.5955	0.1040	0.0165	0.0205	0.1275	0.0
4	0.6880	0.2220	0.0355	0.0325	0.1375	0.002
5	0.7210	0.3270	0.0575	0.0490	0.1455	0.002
6	0.7310	0.4030	0.0695	0.0630	0.1510	0.008
7	0.7310	0.4470	0.0740	0.0715	0.1555	0.018
8	0.7310	0.4705	0.0740	0.0780	0.1565	0.030
9	0.7310	0.4840	0.0740	0.0806	0.1575	0.036
14						0.084

Table 24.1 Cumulative fixation probability for $Int(16)$. Truncation selection vs. Boltzmann selection with $\beta_t = 0.01$; N denotes size of population.

24.9 POPULATION SIZE, BAYESIAN PRIOR, AND MUTATION

Usually the empirical probabilities are computed by the maximum likelihood estimator. For N samples with $m \leq N$ instances of x the estimate is defined by

$$\hat{p}(x) = \frac{m}{N}.$$

For $m = N$ we obtain $p(x) = 1$ and for $m = 0$ we obtain $p(x) = 0$. This leads to our gene fixation problem, because both values are attractors. The fixation problem is reduced if $\hat{p}(x)$ is restricted to an interval $0 < p_{min} \leq \hat{p}(x) \leq 1 - p_{min} < 1$. This is exactly what results from the *Bayesian estimation*. The estimate $\hat{p}(x)$ is the expected value of the posterior distribution after applying Bayes formula to a prior distribution and the given data. For binary variables x the estimate

$$\hat{p}(x) = \frac{m + r}{N + 2r} \quad (24.69)$$

is used with $r > 0$. r is derived from a Bayesian prior. $r = 1$ is the result of the uniform Bayesian prior. The larger r , the more the estimates tend towards $1/2$. The reader interested in a derivation of this estimate in the context of Bayesian networks is referred to [8].

How can we determine an appropriate value for r for our FDA application? The uniform prior gives for $m = 0$ the value $\hat{p}_{min} = 1/(N + 2)$. If N is small, then p_{min} might be so large that we generate the optima with a very small probability only. This means we perform more a random search instead of converging to the optima. This consideration leads to a constraint. $1 - p_{min}$ should be so large that the optima are still generated with high probability. We now heuristically derive p_{min} under the assumption that the optimum is unique. To simplify the formulas we require that

$$\max \hat{p}(x_{opt}) \geq e^{-1}. \quad (24.70)$$

This means that the optimum string x_{opt} should be generated more than 30% at equilibrium. This is large enough to observe equilibrium and convergence. Let us first investigate the UMDA factorization $p(\mathbf{x}) = \prod p(\mathbf{x}_i)$. For $r = 1$ the largest probability is $\hat{p}_{max} = (N + 1)/(N + 2)$. Obviously

$$\hat{p}_{max} = 1 - \frac{1}{N + 2} = 1 - p_{min}.$$

The largest probability to generate the optimum is given by

$$\hat{p}(x_{opt}) = \prod_{i=1}^n \left(1 - \frac{1}{N + 2}\right) \approx e^{-\frac{n}{N+2}}.$$

If $N = O(n^{1-\alpha})$ with $\alpha > 0$, then $p(x_{opt})$ will get arbitrarily small for large n . For $N = n$ we obtain $\hat{p}(x_{opt}) \approx e^{-1}$. This gives the following guideline, which actually is a lower bound of the population size.

Rule of Thumb: *For UMDA the size of the population should be at least equal to the size of the problem.*

Bayesian priors are also defined for conditional distributions. The above heuristic derivation can be also used for general Bayesian factorizations. The Bayesian estimator is for binary variables

$$\hat{p}(x_i | pa_i) = \frac{m + r}{P + 2r}.$$

P is the number of occurrences of pa_i . We make the assumption that in the best case the optimum constitutes 25% of the population. This gives $P \geq N/4$. For $r = 1$ we compute as before

$$\hat{p}(x_{opt}) = \prod_{i=1}^n \hat{p}(x_{opt_i} | pa_{opt_i}) = \prod_{i=1}^n \left(1 - \frac{1}{N/4 + 2}\right) \approx e^{-\frac{n}{N/4+2}}.$$

If we set $N = 4n$ we obtain $\hat{p}(x_{opt}) \approx e^{-1}$. Thus we obtain a lower bound for the population size:

Rule of Thumb: *For FDA working with a factorization with many conditional distributions, the size of the population should be about four times the size of the problem.*

These rule of thumbs have been heuristically derived. They have to be confirmed by numerical studies. Our FDA estimate is a crude lower bound. There exist more general estimates. We just cite Vapnik [21]. In order to approximate a distribution with a reasonable accuracy, he proposes to use a sample size which is about 20 times larger than the number of free parameters of the distribution. This means for a Bayesian factorization

$$N \approx 20 \sum_{i=1}^n 2^{|PA_i|}. \quad (24.71)$$

For UMDA this gives $20n$, i.e. 20 times our estimate. For more complex distributions used by FDA the bound might still be larger. If $|PA_i|$ is bounded by a constant independent of n , the proposed sampling size is linear in n , but exponential in $|PA_i|$. Thus the factor of Vapnik's estimate might be much larger than the factor of our proposal, which is 4. But note that we do not need to completely approximate a distribution. We only need a good approximation in the area where the optimum is located. This problem is discussed in the next section.

A Bayesian prior is identical to a mutation rate in genetic algorithms. The population does not converge to boundary points, but to attractors in the interior. We expect that a Bayesian prior reduces the critical popsize. But it increases the number of generations until convergence.

We demonstrate the importance of using a Bayesian prior by an example. It is a deceptive function of order 4 and problem size of $n = 32$. Our convergence theorem gives convergence of FDA with Boltzmann selection and an exact factorization. The exact factorization consists of marginal distributions of size 4. We compare in Figure 24.5 FDA with SDS Boltzmann selection and truncation selection without Bayesian prior. We also show a run with SDS Boltzmann selection and Bayesian prior.

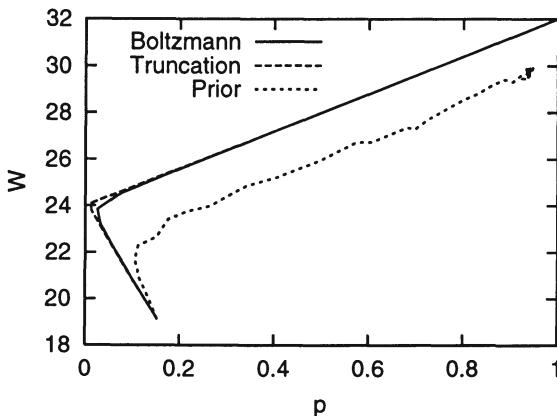


Figure 24.5 Average fitness $W(p)$ for FDA for Decep(32,4); population size $N = 20000$ without prior and $N = 200$ with prior $r = 1$.

The simulation was started at $p = 0.15$, i.e. near the local optimum $p = 0$. Nevertheless, FDA converges to the global optimum at $p = 1$. It is interesting to note that FDA at first moves into the direction of the local optimum. At the very last moment the direction of the curve is dramatically changed. SDS Boltzmann selection behaves almost identical to truncation selection with threshold $\tau = 0.35$. But both methods need a huge population size in order to converge to the optimum. In this example it is $N = 20000$. If a prior of $r = 1$ is used the population size can be reduced to $N = 200$. With this prior the

curve changes direction earlier. Because of the prior the univariate marginal probabilities never reach $p = 0$ or $p = 1$. In this example p stops at about $p = 0.975$.

Let us now summarize our results. *FDA with a finite population of size $N = 4n$, SDS Boltzmann selection, Bayesian prior, and a Bayesian factorization where the number of parents is restricted by k independent of n , will converge to the optimum in polynomial time with high probability.*

It is obvious that a Bayesian prior slows down the convergence. It reduces the probability of fixation, but it increases the number of generations to convergence.

But how do we compute a factorization? A factorization can be derived from the structure of the function, if the function is additively decomposed. This is described in [16]. Another, more general method is to determine the factorization from the data sampled. This is described next.

24.10 LFDA - LEARNING A BAYESIAN FACTORIZATION

Computing the structure of a Bayesian network from data is called learning. Learning gives an answer to the question: *Given a population of selected points $M(t)$, what is a good Bayesian factorization fitting the data?* The most difficult part of the problem is to define a quality measure also called scoring measure.

A Bayesian network with more arcs fits the data better than one with less arcs. Therefore a scoring metric should give the best score to the minimal Bayesian network which fits the data. It is outside the scope of this paper to discuss this problem in more detail. The interested reader is referred to the two papers by Heckerman and Friedman et al. in [8].

For Bayesian networks two quality measures are most frequently used - the *Bayes Dirichlet* (BDe) score and the *Minimal Description Length* (MDL) score. We concentrate on the MDL principle. It is motivated by universal coding. Suppose we are given a set D of instances, which we would like to store. Naturally, we would like to conserve space and save a compressed version of D . One way of compressing the data is to find a suitable model for D that the encoder can use to produce a compact version of D . In order to recover D we must also store the model used by the encoder to compress D . Thus the total description length is defined as the sum of the length of the compressed version of D and the length of the description of the model. The MDL principle postulates that the optimal model is the one that minimizes the total description length.

In the context of learning Bayesian networks, the model is a network B describing a probability distribution p over the instances appearing in the data. Several authors have approximately computed the MDL score. Let $M = |D|$ denote the size of the data set. Then MDL is approximately given by

$$MDL(B, D) = -\text{ld}(P(B)) + M \cdot H(B, D) + \frac{1}{2}PA \cdot \text{ld}(M), \quad (24.72)$$

with $\text{ld}(x) := \log_2(x)$. $P(B)$ denotes the prior probability of network B , $PA = \sum_i 2^{|pa_i|}$ gives the total number of probabilities to compute. $H(B, D)$ is defined by

$$H(B, D) = - \sum_{i=1}^n \sum_{pa_i} \sum_{x_i} \frac{m(x_i, pa_i)}{M} \text{ld} \frac{m(x_i, pa_i)}{m(pa_i)} \quad (24.73)$$

where $m(x_i, pa_i)$ denotes the number of occurrences of x_i given configuration pa_i . $m(pa_i) = \sum_{x_i} m(x_i, pa_i)$. If $pa_i = \emptyset$, then $m(x_i, \emptyset)$ is set to the number of occurrences of x_i in D .

The formula has an interpretation which can be easily understood. If no prior information is available, $P(B)$ is identical for all possible networks. For minimizing, this term can be left out. $0.5PA \cdot \text{ld}(M)$ is the length required to code the parameter of the model with precision $1/M$. Normally one would need $PA \cdot \text{ld}(M)$ bits to encode the parameters. However, the central limit theorem says that these frequencies are roughly normally distributed with a variance of $M^{-1/2}$. Hence, the higher $0.5\text{ld}(M)$ bits are not very useful and can be left out. $-M \cdot H(B, D)$ has two interpretations. First, it is identical to the logarithm of the maximum likelihood ($\text{ld}(L(B|D))$). Thus we arrive at the following principle:

Choose the model which maximizes $\text{ld}(L(B|D)) - \frac{1}{2}PA \cdot \text{ld}(M)$.

The second interpretation arises from the observation that $H(B, D)$ is the conditional entropy of the network structure B , defined by PA_i , and the data D . The above principle is appealing, because it has no parameter to be tuned. But the formula has been derived under many simplifications. In practice, one needs more control about the quality vs. complexity tradeoff. Therefore we use a weight factor α . Our measure to be maximized is called BIC .

$$BIC(B, D, \alpha) = -M \cdot H(B, D) - \alpha PA \cdot \text{ld}(M). \quad (24.74)$$

This measure with $\alpha = 0.5$ has been first derived by Schwarz [20] as *Bayesian Information Criterion*.

To compute a network B^* which maximizes BIC requires a search through the space of all Bayesian networks. Such a search is more expensive than to search for the optima of the function. Therefore the following greedy algorithm has been used. k_{max} is the maximum number of incoming edges allowed.

$$\text{BN}(\alpha, k_{max})$$

STEP 0: Start with an arc-less network.

STEP 1: Add the arc (x_i, x_j) which gives the maximum increase of $BIC(\alpha)$
if $|PA_j| \leq k_{max}$ and adding the arc does not introduce a cycle.

STEP 2: Stop if no arc is found.

Checking whether an arc would introduce a cycle can be easily done by maintaining for each node a list of parents and ancestors, i.e. parents of parents etc. Then $(x_i \rightarrow x_j)$ introduces a cycle if x_j is ancestor of x_i .

A similar algorithm has been proposed in [18]. It uses a different score. Numerical results comparing UMDA, FDA, and LFDA can be found in [13, 14]. They can be summarized as follows.

UMDA most efficiently optimizes the functions *OneMax* and *Saw*. FDA is efficient if the exact factorization needs a small number of parents in the Bayesian graph ($k \leq 5$). LFDA most of the time also finds the optimum. From the functions considered it has the largest difficulty with the function *Saw*. The performance of LFDA can be substantially improved, if for each fitness function a suitable value of α is chosen. We recall that a small value of α leads to more complex Bayesian factorizations. The *BIC* score uses $\alpha = 0.5$. This value is a good compromise. But $\alpha = 0.75$ gives much better performance for the functions *OneMax* and *Saw*, whereas $\alpha = 0.25$ gives the best results for the function *Decep(36, 4)*. These results are explained next.

24.10.1 Optimization, dependencies, and search distributions

We have proven in Section 24.6 convergence of FDA with Boltzmann selection to the set of global maxima. If the Boltzmann distribution can be factorized, the computational complexity for one generation is bounded by $O(n \cdot N \cdot 2^k)$. k denotes the maximum number of parents. A factorization can be determined if the fitness function is decomposed. It can also be obtained from the data sampled. Unfortunately for many interesting applications k is very large. If the fitness function is additively decomposed on a 2D grid of size n , then k scales like $O(\sqrt{n})$. It is easy to show that k scales even like $O(n)$ for the function *Saw*.

But we have demonstrated that the simple search distribution used by UMDA guides the search to the optimum of *Saw*. The reason is that for *Saw* we have a tendency: the more bits on, the higher the fitness value. Therefore an exact Boltzmann factorization is not needed for optimization! The problem of finding a good approximation of the Boltzmann distribution which generates the optima with high probability cannot be solved theoretically. Therefore we propose the following heuristic.

Multi-Factorization LFDA *Use different values of α in order to obtain factorizations of different complexity. In a standard setting, use $\alpha = 0.25, 0.5$, and 0.75 . Generate new search points using the different factorizations for a certain percentage of the population.*

24.11 THREE ROYAL ROADS TO OPTIMIZATION

In this section we will try to classify the different approaches presented. Population search methods are based on two components at least – selection and reproduction with variation. In our research we have transformed genetic algorithms to a family of algorithms using search distributions instead of recombination/mutation of strings. The simplest algorithm of this family is the univariate marginal distribution algorithm UMDA.

Wright's equation describes the behavior of UMDA using an infinite population and proportionate selection. The equation shows that UMDA does *not* primarily optimize the *fitness function* $f(x)$, but the *average fitness* of the population $W(\mathbf{p})$ which depends on the continuous marginal frequencies $p_i(x_i)$. Thus the important landscape for population search is *not* the landscape defined by the fitness function $f(x)$, but the landscape defined by $W(\mathbf{p})$. In mathematical terms Wright's equation transforms the discrete optimization problem into a continuous one. Thus mathematically we can try to optimize $W(\mathbf{p})$ instead of $f(x)$. This later approach we (and other researcher) call the system dynamics approach to optimization.

The two components of population based search methods — selection and reproduction with variation — can work on a microscopic (individual) or a macroscopic (population) level. The level can be different for selection and reproduction. It is possible to classify the different approaches according to the level the components work. The following table shows three classes of evolutionary algorithms, each with a representative member:

Algorithm	Selection	Reproduction
Genetic Algorithm	microscopic	microscopic
UMDA	microscopic	macroscopic
System Dynamics	macroscopic	macroscopic

A genetic algorithm uses a population of individuals. Selection and recombination is done by manipulating individual strings. UMDA uses marginal distributions to create individuals. These are macroscopic variables. Selection is done on a population of individuals, as genetic algorithms do. In the system dynamics approach selection is modeled by a specific dynamic difference equation for macroscopic variables. We believe that a fourth class — macroscopic selection and microscopic reproduction — makes no sense.

Each of the approaches have their specific pros and cons. Genetic algorithms are very flexible, but the standard recombination operator has limited capabilities. UMDA can use any kind of selection method which is also used by genetic algorithm. UMDA be extended to an algorithm which uses a more complex factorization of the distribution. This is done by the factorized distribution algorithm FDA. Selection is very difficult to model on a macroscopic level. Wright's equation are valid for proportionate selection only. Other selection schemes lead to very complicated system dynamics equations.

Thus for proportionate selection and gene pool recombination all methods will behave similarly. But each of the methods allows extensions which cannot be modeled with an approach using a different level.

Mathematically especially interesting is the extension of UMDA to FDA with an adaptive Boltzmann annealing schedule. For this algorithm convergence for a large class of discrete optimization problems has been shown.

24.12 SUMMARY AND OUTLOOK

We have presented three approaches to optimization. We believe that the optimization methods based on search distributions (UMDA,FDA,LFDA) have the greatest optimization power. FDA with Boltzmann selection SDS is an extension of simulated annealing to a population of points. It shares with simulated annealing the convergence property, but convergence is much faster.

Ultimately our theory leads to a synthesis problem: finding a good factorization for a search distribution defined by a finite sample. This is a central problem in probability theory. One approach to this problem uses Bayesian networks. For Bayesian networks numerically efficient algorithms have been developed. Our LFDA algorithm computes a Bayesian network by minimizing the Bayesian Information Criterion.

The computational effort of both FDA and LFDA is substantially higher than that of UMDA. Thus UMDA should be the first algorithm to be tried in a practical problem. Next the Multi-Factorization LFDA should be applied.

The theory and the algorithms can be extended to optimization problems with constraints. A first step has already been made in [16]. Genetic algorithms need a different recombination operator for each constraint problem.

References

- [1] E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines*. Wiley, 1989.
- [2] S. Baluja and R. Caruana. Removing the Genetics from the Standard Genetic Algorithm. In: *Proceedings of the 12th International Conference on Machine Learning*, Morgan Kaufmann, pages 38–46, San Francisco, 1995.
- [3] L.E. Baum and J.A. Eagon. An Inequality with Applications to Statistical Estimation for Probabilistic Functions of Markov Processes and to a Model for Ecology. *Bulletin of the American Mathematical Society*, 73:360–363, 1967.
- [4] M. Dorigo and G. Di Caro. The Ant Colony Optimization Meta-Heuristic. In: *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, editors, MacGraw-Hill, pages 11–32, 1999.
- [5] D.S. Falconer. *Introduction to Quantitative Genetics*. Longman, 1981.
- [6] B.J. Frey. *Graphical Models for Machine Learning and Digital Communication*. MIT Press, 1998.
- [7] G. Harik, E. Cantu-Paz, D.E. Goldberg, and B.L. Miller. The Gambler’s Ruin Problem, Genetic Algorithms, and the Sizing of Populations. *Evolutionary Computation*, 7:231–255, 1999.
- [8] M.I. Jordan. *Learning in Graphical Models*. MIT Press, 1999.

- [9] M. Mitchell, J.H. Holland, and St. Forrest. When Will a Genetic Algorithm Outperform Hill Climbing? *Advances in Neural Information Processing Systems*, 6:51–58, 1994.
- [10] H. Mühlenbein. The Equation for the Response to Selection and its Use for Prediction. *Evolutionary Computation*, 5:303–346, 1997.
- [11] H. Mühlenbein, M. Gorges-Schleuter, and O. Krämer. Evolution Algorithms in Combinatorial Optimization. *Parallel Computing*, 7:65–88, 1988.
- [12] H. Mühlenbein and Th. Mahnig. Convergence Theory and Applications of the Factorized Distribution Algorithm. *Journal of Computing and Information Technology*, 7:19–32, 1999.
- [13] H. Mühlenbein and Th. Mahnig. FDA – A Scalable Evolutionary Algorithm for the Optimization of Additively Decomposed Functions. *Evolutionary Computation*, 7:353–376, 1999.
- [14] H. Mühlenbein and Th. Mahnig. Evolutionary Algorithms: From Recombination to Search Distributions. In: *Theoretical Aspects of Evolutionary Computing*, L. Kallel, B. Naudts, and A. Rogers, editors, Natural Computing, Springer Verlag, pages 137–176, 2000.
- [15] H. Mühlenbein and Th. Mahnig. Evolutionary Computation And Wright’s Equation. Submitted to *Journal of Computer Science*, 2001.
- [16] H. Mühlenbein, Th. Mahnig, and A. Rodriguez Ochoa. Schemata, Distributions and Graphical Models in Evolutionary Optimization. *Journal of Heuristics*, 5:215–247, 1999.
- [17] H. Mühlenbein and H.-M. Voigt. Gene Pool Recombination in Genetic Algorithms. In: *Metaheuristics: Theory and Applications*, J.P. Kelly and I.H Osman, editors, Kluwer, pages 53–62, 1996.
- [18] M. Pelikan, D.E. Goldberg, and E. Cantu-Paz. Linkage Problem, Distribution Estimation, and Bayesian Networks. *Evolutionary Computation*, 8:311–341, 2000.
- [19] A. Prügel-Bennet and J.L. Shapiro. An Analysis of a Genetic Algorithm for Simple Random Ising Systems. *Physica D*, 104:75–114, 1997.
- [20] G. Schwarz. Estimating the Dimension of a Model. *Annals of Statistics*, 7:461–464, 1978.
- [21] V. Vapnik. *Statistical Learning Theory*. Wiley, 1998.
- [22] H.-M. Voigt. *Evolution and Optimization*. Akademie-Verlag, 1989.
- [23] S. Wright. Random Drift and the Shifting Balance Theory of Evolution. In: *Mathematical Topics in Population Genetics*, K. Kojima, editor, Springer Verlag, pages 1–31, 1970.

25 FORMULATION AND TABU SEARCH ALGORITHM FOR THE RESOURCE CONSTRAINED PROJECT SCHEDULING PROBLEM

Koji Nonobe and Toshihide Ibaraki

Department of Applied Mathematics and Physics
Graduate School of Informatics, Kyoto University, Kyoto 606-8501, Japan
`{nonobe,ibaraki}@i.kyoto-u.ac.jp`

Abstract: The resource constrained project scheduling problem (RCPSP) can formulate many scheduling problems including jobshop and flowshop scheduling problems. In this paper, we extend the definition of RCPSP further so that various complicated constraints and objective functions arising in practice can be handled; for example, each activity can be processed in one of the selectable modes, the available amounts of renewable resources may vary depending on the periods, setup activities can be dealt with, and complex objective functions can be handled. Then, we develop a tabu search based heuristic algorithm, which contains elaborations in representing solutions and in constructing neighborhood. Our code was tested for many benchmarks of RCPSP, and also for some problems from real applications. For a number of RCPSP instances, we found better solutions than the best ones found so far. These computational results indicate the effectiveness and usefulness of our approach.

25.1 INTRODUCTION

The resource constrained project scheduling problem (RCPSP) asks to determine start times of the participating activities under the constraints of various resources that are consumed by the activities, as well as precedence constraints between activities. For the RCPSP, many exact and approximate methods have been proposed so far. Recently, to deal with problems of large scale, meta-heuristic algorithms have been studied and brought successful computational results. In these developments [1, 2, 5, 7, 8, 12, 13, 14, 15, 19, 21, 22],

most literature employs a rather simple model of RCPSP, whose objective is to minimize the makespan, and whose constraints are only those of renewable resources and precedence constraints. In real situations, however, more complicated constraints often arise, which cannot be formulated in such a simple form of RCPSP.

Our objective in this paper is to develop a general purpose RCPSP algorithm that can handle much wider range of scheduling problems. Some generalization of RCPSP have already been discussed in references such as [4, 9]. Our generalization consists in that each activity can be processed in one of the selectable modes (RCPSP with this generalization is called the multi-mode RCPSP [4, 9, 15]), nonrenewable resources can be included in addition to renewable resources, the available amounts of renewable resources may vary depending on the periods, and setup activities can also be dealt with. A wider class of objective functions, other than the makespan, can be included as soft constraints of our model. For this generalized formulation of RCPSP, we develop a tabu search algorithm. This code was applied to many benchmarks of RCPSP, and improved the best known solutions for a number of problem instances. Some problems from real applications were also solved to demonstrate that our generalization of RCPSP is essential to handle practical problems. We report that rather good solutions could be obtained to these instances.

Of course, general purpose algorithms like ours may not be efficient enough if compared with special purpose algorithms tailored to special cases of RCPSP (e.g., jobshop scheduling). But we believe that the generality is very important in order to be accepted by people in practical applications, since it can save a great deal of manpower, which is otherwise necessary to develop special purpose codes.

25.2 FORMULATION OF RCPSP

We explain ingredients of RCPSP separately in the following subsections.

25.2.1 Resources and activities

Let R denote a set of resources, and J denote a set of activities. The set R is divided into the set of renewable resources R^{re} and the set of nonrenewable resources R^{non} . Renewable resources are those whose available amounts are restricted only on a period basis, and are independent of other periods, while nonrenewable resources are limited in total throughout the schedule horizon without any restriction in each period. For example, manpower and machines are regarded as renewable resources, while raw materials and budgets are typical examples of nonrenewable resources. Each activity $j \in J$ must be processed in one mode m_j chosen from a set M_j . For each mode $m_j \in M_j$, its processing time p_{m_j} , the set of required resources $R_{m_j} = R_{m_j}^{re} \cup R_{m_j}^{non}$ and their required amounts are prespecified. Once the processing of an activity j is initiated in mode m_j , it is processed in the consecutive p_{m_j} periods to finish. A schedule is represented by $(m, s) = ((m_j \mid j \in J), (s_j \mid j \in J))$, where m_j is the mode

in which activity j is processed, and s_j is the *start time* of j (the *completion time* of j is then given by $c_j = s_j + p_{m_j}$). The values of p_{m_j}, s_j and c_j are all assumed to be non-negative integers, in this paper.

25.2.2 Resource constraints

Resource constraints are classified into the following two types.

- [1] Renewable resource constraints: For each renewable resource $r \in R^{re}$, in each period $(t - 1, t]$ ($t = 1, 2, \dots$), the total requirement of r by all activities must not exceed the available amount $K_{r,t}$. Here, unlike the standard assumption frequently used, the available amounts $K_{r,t}$ can vary depending on the periods $(t - 1, t]$, and the amounts of r required during the processing periods of an activity may not be uniform.

Machines that can process only one activity at a time are special renewable resources that are often encountered in real scheduling problems. In the RCPSP models, a machine is regarded as a renewable resource r with unit of available amount in every period. When an activity is processed on the machine, it consumes unit amount in each period of processing.

- [2] Nonrenewable resource constraints: For each nonrenewable resource $r \in R^{non}$, its total requirement in the entire schedule horizon must be within the prespecified available amount. The required amount of r by an activity depends on only its mode, and is independent of its start time. Thus, these constraints prohibit some combinations of modes; if all activities have only one mode, the nonrenewable resource constraint is redundant.

25.2.3 Precedence constraints

Two types of precedence constraints are considered in our formulation of RCPSP.

- [1] A *precedence constraint* $j_1 \prec j_2$ between activities, which says that activity j_2 cannot be started unless activity j_1 is finished, i.e.,

$$c_{j_1} \leq s_{j_2}. \quad (25.1)$$

- [2] An *immediate precedence constraint* $j_1 \prec\prec_r j_2$ on a renewable resource $r \in R^{re}$ (i.e., j_1 in mode m_{j_1} immediately precedes j_2 in mode m_{j_2} if both activities use $r \in R_{m_{j_1}}^{re} \cap R_{m_{j_2}}^{re}$), which is described by

$$c_{j_1} \leq s_{j_2} \quad \text{and} \quad s_{j_2} \leq s_{j'} \quad \text{for all } j' \text{ such that } r \in R_{m_{j'}}^{re} \text{ and } c_{j_1} \leq s_{j'}. \quad (25.2)$$

These immediate precedence constraints are restricted versions of the precedence constraint (25.1), but are useful to describe some complicated constraints which appear in the following situations.

- (i) The setup activity $\sigma(j)$ for activity j must be processed just before j . This constraint can be described by an immediate precedence constraint. Setup activities will be discussed in more details in Section 25.2.4.
- (ii) Two activities j_1 and j_2 must be initiated at the same time. This constraint can be described by introducing a fictitious activity j' and a fictitious renewable resource r which is consumed by j_1, j_2 and j' , and adding two immediate precedence constraints $j' \prec\!\prec_r j_1$ and $j' \prec\!\prec_r j_2$.
- (iii) The processings of two activities j_1 and j_2 must overlap. This constraint can be described by two immediate precedence constraints $j_1 \prec\!\prec_r j'$ and $j_2 \prec\!\prec_r j'$ for the fictitious activity j' as in 2.
- (iv) The processing of j_2 must be started within q periods after j_1 is completed (i.e., $s_{j_2} \leq c_{j_1} + q$ holds), where $j_1 \prec j_2$ is assumed. This is described by introducing a fictitious activity j' whose processing time is $q + 2$ and which overlaps both of j_1 and j_2 . In particular, the constraint that no idle time is permitted between j_1 and j_2 can be represented by setting $q = 0$.

As a generalization of the precedence constraint, the temporal constraint is often introduced (e.g., [4, 9]), which specifies minimum and maximum start-start time lags between activities and can be described by

$$s_{j_2} - s_{j_1} \geq \delta_{j_1 j_2} \quad (25.3)$$

for an appropriate constant $\delta_{j_1 j_2}$ (which can be either positive or negative). Similarly, temporal constraints for completion-start, start-completion and completion-completion time lags are allowed. These temporal constraints can handle the above situations (ii), (iii) and (iv); however, they cannot represent the immediate precedence relations between activities and their setup activities. On the other hand, constraints of type (25.3) can be represented by using immediate precedence constraints (though we need to introduce the associated dummy resources and activities).

Here, we define a digraph $D_\prec = (J, A_\prec)$, which is often referred to as an *activity-on-nodes network*, a *project network* or a *precedence graph*. J is the set of activities, and A_\prec is given by $A_\prec = A_I \cup A_P$, where

$$A_I = \{(j_1, j_2) \mid \exists r, j_1 \prec\!\prec_r j_2\}, \text{ and}$$

$$A_P = \{(j_1, j_2) \mid j_1 \prec j_2\} \setminus A_I.$$

An example of D_\prec is illustrated in Figure 25.1. We assume that D_\prec is acyclic, since otherwise there exists no schedule satisfying all the precedence constraints (unless all activities on all cycles have zero processing time).

$$D_{\prec} = (J, A_{\prec})$$

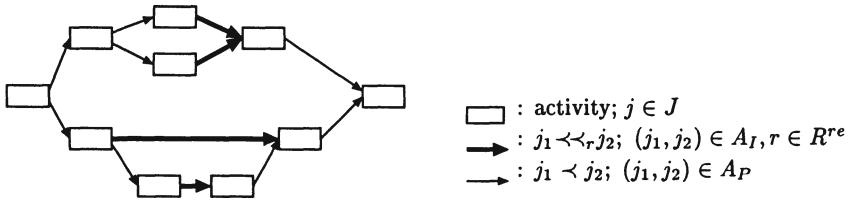


Figure 25.1 An example of $D_{\prec} = (J, A_{\prec})$.

25.2.4 Setup activities

In many problems in practice, the setup operation $setup(j_1, j_2)$ becomes necessary between changing from activity j_1 to j_2 on a machine r . While processing the setup operation, no activity can be processed on r . Although setup operations are very important in real applications, the research on this aspect has been rather limited. In our formulation, such setup operations can be handled as follows.

We introduce the setup activity $\sigma(j_2)$, which is required for the processing of activity j_2 , together with an immediate precedence constraint $\sigma(j_2) \ll_r j_2$. The activity $\sigma(j_2)$ is also processed on machine r and its mode is determined according to the activity $prev(\sigma(j_2))$ that immediately precedes $\sigma(j_2)$ on machine r . In our algorithm, we consider only those schedules in which all setup activities are processed in these specified modes.

In the subsequent discussion, setup activities $\sigma(j)$ will be also regarded as standard activities in J , unless otherwise specified.

25.2.5 Hard and soft constraints, and objective functions

In our formulation, not only resources and precedence constraints, but also any other constraints imposed on modes m_j , start times s_j , completion times c_j and/or processing times p_j of activities j can be included. In our current implementation, only linear inequality constraints in the form of

$$\sum_{j \in J} \sum_{m_j \in M_j} \alpha_{j,m_j} x_{j,m_j} + \sum_{j \in J} \beta_j s_j \leq \gamma \quad (25.4)$$

are accepted as additional constraints, where α_{j,m_j} , β_j and γ are constants, and x_{j,m_j} is 0-1 variable that takes value 1 (resp., 0) if activity j is (resp., is not) processed in mode m_j . Note that constraint (25.4) may include processing times p_j and completion times c_j of activities j , since p_j and c_j are represented as $\sum_{m_j} p_{m_j} x_{j,m_j}$ and $\sum_{m_j} p_{m_j} x_{j,m_j} + s_j$, respectively.

The constraints are classified into *hard* constraints and *soft* constraints, meaning that the former must always be satisfied for the obtained schedules to be practically meaningful, while the latter may remain violated if it is difficult to satisfy them. For each soft constraint C_ℓ , a *penalty function* $p_\ell(m, s)$ and a *weight* $w_\ell > 0$ are introduced, where $p_\ell(m, s)$ takes 0 (resp., positive) value if C_ℓ is satisfied (resp., violated), and w_ℓ is associated to represent the importance of C_ℓ . Then, we try to find an optimal schedule in the sense of minimizing the sum of weighted penalties

$$p(m, s) = \sum_{\ell} w_\ell p_\ell(m, s) \quad (25.5)$$

subject to all hard constraints. In other words, the quality of a schedule is evaluated by the amounts of violations of these soft constraints. A schedule satisfying all hard constraints is called a *feasible schedule*.

In our algorithm, only the constraints on renewable resources and precedence relations can be specified as hard, although, if necessary, some of them may be specified as soft by putting them in the category of (25.4). Therefore, the constraints of type (25.4) are always considered as soft. If the weights are set to substantially large numbers, however, the corresponding constraints behave like hard constraints even if their feasibility is not always maintained. The constraints on nonrenewable resources are also specified in the form of linear inequalities (25.4); i.e., they are considered to be soft. However, all the immediate precedence constraints are regarded as hard, and the constraints on the modes of setup activities are also hard, as mentioned in Section 25.2.4.

Except for the above default rules, it is users' responsibility to specify which constraints are soft, and how much weights should be associated with them. Although these decisions must be carefully done so that acceptable schedules may be obtained, this flexibility permits a wide applicability of our formulation.

In concluding this section, we note that the objective function is optimized in the process of minimizing (25.5). For example, makespan can be minimized by introducing the following soft constraint for the activity *sink*, which succeeds all others by precedence constraints and whose processing time is 0,

$$c_{\text{sink}} \leq 0. \quad (25.6)$$

Then minimizing the penalty of this constraint (which is simply the value of c_{sink} in this case) enables us to minimize the makespan.

In real applications, in addition to makespan, we may encounter various objective functions, related to other parameters such as earliness and tardiness. For example, the minimization of the earliness and tardiness cost of an activity can be achieved by using the following soft constraint on a renewable resource. Let d_j be the due date of activity j , and $\text{cost_due}(c_j, d_j)$ be a given cost function. We introduce a renewable resource $r_{\text{due}}(j)$, which is consumed only in the last period of processing j by amount $k = \max_t \text{cost_due}(t, d_j)$, and whose available amounts $K_{r_{\text{due}}(j), t}$ are defined as $k - \text{cost_due}(t, d_j)$ ($t = 1, 2, \dots$). If j is completed at time t , then its penalty is given by the excess of $r_{\text{due}}(j)$,

that is, $k - K_{r_{\text{due}}(j), t} = \text{cost_due}(t, d_j)$, and hence represents the earliness or tardiness cost $\text{cost_due}(c_j, d_j)$. In this way, we can handle any cost function $\text{cost_due}(c_j, d_j)$ defined by c_j and d_j .

25.3 TABU SEARCH FOR RCPSP

There have been proposed many heuristic algorithms for RCPSP [1, 2, 5, 7, 8, 10, 12, 13, 14, 15, 18, 19, 21, 22]. Among them, some effective RCPSP algorithms (e.g., genetic algorithms by Hartmann [7, 8], an ant colony optimization algorithm by Merkle et al. [13] and a tabu search algorithm by Baar et al. [1]) represent a solution by a permutation of all the activities (which is often called a *list*), from which a schedule is generated by the serial or parallel schedule generation scheme [10]. The list representation is also used in our algorithms. The algorithms [1, 7, 8, 13] are developed for the standard RCPSP, that is, only constraints on renewable resources and precedence relations are considered, the available amounts of resources do not vary with periods, and the objective is to minimize makespan. In [19], Sampson and Weiss considered a generalization of RCPSP such that any objective function and constraints can be handled, and proposed a local search algorithm. In their algorithm, a solution is represented by a *shift vector*, each element of which gives the start time lag of an activity from the maximum completion time of its predecessors. A schedule constructed from a shift vector always satisfies the precedence constraints, but may violate other constraints. The feasibility is sought by minimizing the penalty function like (25.5). More recently, Möhring et al. have developed a Lagrangian-based heuristic algorithm [14] for the time-indexed integer programming formulation. Their approach can be easily extended to general RCPSPs with other objective functions than the makespan, and with time-dependent resource availabilities and requirements.

Our algorithm is developed for a generalized formulation of RCPSP, and is based on tabu search. All the schedules considered in our algorithm are generated from lists (i.e., permutations of activities), as in [1, 7, 8, 13], but require more careful consideration since we handle more general constraints, which are classified into hard and soft.

Tabu search [6] is a local search algorithm with a modification that the search continues to the best neighboring solution even if local optimum is attained. In tabu search, to avoid cycling of solutions, a set of solutions including those recently visited are kept in *tabu list*, and excluded from the candidate list of the next solution.

In this section, we first define basic components of local search such as search space and neighborhood, and then describe how we implement our tabu search.

25.3.1 Representation of solutions

As mentioned in Section 25.2, a schedule can be represented by a vector (m, s) . However, this representation may not give rise to an efficient search algorithm, since changing the start time s_j or the mode m_j of one activity j (but not

other variables) may cause a lot of constraint violations which are not straightforward to adjust. To avoid this complication, we represent a solution by a list $\pi = (\pi(1), \pi(2), \dots, \pi(|J|))$ (a permutation of all activities) and a vector of mode selections $\mathbf{m} = (m_j \mid j \in J)$ (in [1, 7], a solution is represented only by π , as single-mode RCPSP is considered). The corresponding schedule is then constructed by algorithm CONSTRUCT whenever needed. In other words, CONSTRUCT is a mapping from the space of (\mathbf{m}, π) into the space of schedules. In the subsequent discussion, we sometimes do not distinguish a solution (\mathbf{m}, π) from its schedule constructed by CONSTRUCT.

The algorithm CONSTRUCT tries to construct a feasible (i.e., all hard constraints are satisfied) schedule from a given (\mathbf{m}, π) . Its algorithmic details will be explained later in this section. However, it should be noted that finding a feasible schedule may not be easy if we allow immediate precedence constraints. In fact, if immediate precedence constraints are allowed, the problem of determining the existence of a feasible schedule is NP-hard even if availability and requirement of every renewable resource do not vary with periods. In our implementation of CONSTRUCT, a program parameter T is introduced, which is a prespecified upper bound on the start times s_i of activities i ; i.e., only those schedules satisfying $s_i \leq T$ can be constructed in CONSTRUCT. In the initial part of tabu search, we generate solutions (\mathbf{m}, π) randomly and apply CONSTRUCT to find feasible schedules. If no feasible schedule is found after trying a certain number of random solutions, we simply conclude the failure. In practice, however, such situation rarely occurs (if a sufficiently large value is specified as T), and even if so, we are usually able to relax some of the hard constraints into soft ones so that meaningful results may still be attained.

As explained in Section 25.2.5, some soft constraints may be given large weights and are expected to behave like hard constraints. In this case, as they are distinguished only by their penalty weights w_ℓ , the border between hard and soft constraints becomes somewhat fuzzy. However, we found in practical applications that distinguishing two types of constraints is usually very easy. From the viewpoint of constructing an appropriate model of the problem at hand, this distinction appears to be very important.

Now we describe algorithm CONSTRUCT to construct a feasible schedule from a given solution (\mathbf{m}, π) . For simplicity of description, we first assume that there are no setup activities (how to handle setup activities will be described at the end of this subsection). In CONSTRUCT, activities are sequentially scheduled in the order of π by setting the start time $s_{\pi(i)}$ of each $\pi(i)$ to the earliest time t ($\leq T$) such that the feasibility is met within the activities scheduled so far. As described in Step 2, this t is temporarily set to the minimum value obtained by considering all the activities preceding $\pi(i)$ (by precedence and immediate precedence constraints) and renewable resource requirements. Then, if t is greater than the prespecified T , CONSTRUCT returns “failure”; otherwise the activity $\pi(i)$ is scheduled at $s_{\pi(i)} = t$. Note that the obtained schedule may not be feasible since it may not satisfy immediate precedence constraints (even if $t \geq c_j$ holds for the immediately preceding activities j).

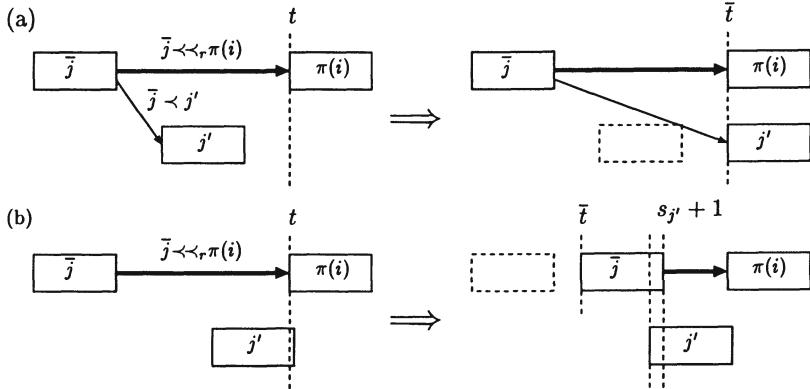


Figure 25.2 Examples of backtracking in Step 3, where immediate precedence constraint $\bar{j} \prec\prec_r \pi(i)$ (bold arrow) is violated: (a) if $\bar{j} \prec j'$, then j' will be delayed; (b) otherwise, \bar{j} will be delayed.

If the schedule is feasible, we move to the next activity $\pi(i+1)$. If it is not feasible, however, there are an immediate precedence constraint $\bar{j} \prec\prec_r \pi(i)$, and an activity j' , such that j' has already been scheduled, its start time $s_{j'}$ satisfies $c_{\bar{j}} \leq s_{j'} < t$, and it prevents $\pi(i)$ from being scheduled at time t . The situation is illustrated in Figure 25.2 (a) and (b), respectively. In these cases, the constraint $\bar{j} \prec\prec_r \pi(i)$ cannot be met by delaying the start time of $\pi(i)$, and a backtracking takes place (in Step 3) to delay the start time of activity j' or \bar{j} . The results are also illustrated in Figure 25.2 (a) and (b). In case (b), it is also possible to resolve the violation of $\bar{j} \prec\prec_r \pi(i)$ by delaying the start time of j' till that of $\pi(i)$ as in case (a). However, we do not adopt this, because when such j' is not unique, delaying \bar{j} seems more effective to reduce the number of such j' .

In our search, we only generate (m, π) for which

$$\pi^{-1}(j_1) < \pi^{-1}(j_2) \quad \text{for all pairs } j_1 \prec j_2 \quad (25.7)$$

holds, where $i = \pi^{-1}(j)$ is defined by $\pi(i) = j$. In other words, no activity is scheduled until all of its predecessors are scheduled. This property will facilitate and simplify some part of the algorithm.

CONSTRUCT

Input: A solution (m, π) .

Output: A feasible schedule (m, s) or “failure” (in the latter case, (m, π) is regarded to have the penalty of infinity).

Step 0 (initialization): $c_j := 0$ ($j \in J$), and $i := 1$.

Step 1 (lower bound of the start time of $\pi(i)$): $\bar{t} := \max_{j \prec \pi(i)} c_j$ ($\bar{t} := 0$ if there is no j with $j \prec \pi(i)$).

Step 2 (checking the feasibility of $\pi(i)$): Calculate the minimum t ($\leq T$) that satisfies the following three conditions (t is the start time of activity $\pi(i)$):

- [1] $t \geq \bar{t}$.
- [2] All hard renewable resource constraints are satisfied for activities $\pi(1), \pi(2), \dots, \pi(i)$.
- [3] For any immediate precedence constraint $j_1 \prec_r j_2$ such that $\pi^{-1}(j_1), \pi^{-1}(j_2) < i$ and $r \in R_{m_{j_1}}^{re} \cap R_{m_{j_2}}^{re} \cap R_{m_{\pi(i)}}^{re}$, it holds $t < c_{j_1}$ or $t \geq s_{j_2}$ (i.e., $\pi(i)$ does not interfere with $j_1 \prec_r j_2$).

If there exists no such t , return “failure”. Otherwise (i.e., such t exists), check whether there are an immediate precedence constraint $\bar{j} \prec_r \pi(i)$ and an activity j' such that $\pi^{-1}(j') < i$, $r \in R_{m_{\bar{j}}}^{re} \cap R_{m_{j'}}^{re} \cap R_{m_{\pi(i)}}^{re}$ and $c_{\bar{j}} \leq s_{j'} < t$ (see the left hand side of Figure 25.2 (a) and (b)). If yes, then go to Step 3; otherwise go to Step 4.

Step 3 (backtracking): The start time of j' or \bar{j} is delayed.

- (a) If $\bar{j} \prec j'$, then let $i := \pi^{-1}(j')$ and $\bar{t} := t$ (this is illustrated in Figure 25.2 (a)).
- (b) Otherwise, let $i := \pi^{-1}(\bar{j})$ and $\bar{t} := s_{j'} + 1 - p_{m_{\bar{j}}}$ (this is illustrated in Figure 25.2 (b)).

In either case, return to Step 2, and reschedule this new $\pi(i)$ and all the activities after $\pi(i)$ in the list, with the updated \bar{t} .

Step 4 (setting the start time of $\pi(i)$): Let $s_{\pi(i)} := t$. If $i < |J|$, then $i := i + 1$ and return to Step 1; otherwise halt after outputting the constructed schedule (\mathbf{m}, \mathbf{s}) .

If there is no immediate precedence constraint, no backtracking occurs in CONSTRUCT. In general, CONSTRUCT returns a feasible schedule or “failure” in finite steps, since the start time of the new $\pi(i)$ ($= j'$ or \bar{j}) is increased after backtracking in Step 3, while those of activities $\pi(i')$ remain the same for all $i' < i$. It is easily shown that CONSTRUCT can be executed in $O(|J|^2|R^{re}|)$ time if restricted to the standard RCPSP.

$$D_{\prec}^* = (J, A_{\prec}^*)$$

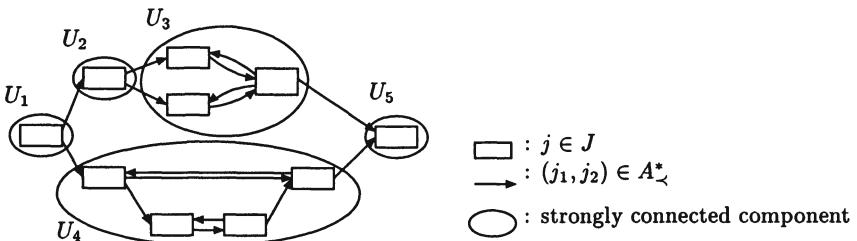


Figure 25.3 The digraph $D_{\prec}^* = (J, A_{\prec}^*)$ obtained from $D_{\prec} = (J, A_{\prec})$ of Figure 25.1.

Now let \mathcal{U} be the set of strongly connected components in digraph $D_{\prec}^* = (J, A_{\prec}^*)$, which is obtained from $D_{\prec} = (J, A_{\prec})$ (defined in Section 25.2.3) by adding arc (j_2, j_1) for each arc $(j_1, j_2) \in A_I$ (see Figure 25.3). A precedence relation \prec on \mathcal{U} is defined by

$$U_1 \prec U_2 \Leftrightarrow U_1 \neq U_2 \text{ and } \exists j_1 \in U_1, \exists j_2 \in U_2 \text{ such that } j_2 \text{ is reachable from } j_1 \text{ in } D_{\prec}^*.$$

In order to suppress the number of activities rescheduled after backtracking in Step 3, we generate in our algorithm only those π that satisfy the following condition:

$$\text{All activities in one component } U \in \mathcal{U} \text{ are consecutively ordered in } \pi. \quad (25.8)$$

If π satisfies conditions (25.7) and (25.8), backtracking in Step 3 occurs only within each component (i.e., once all activities in a component U are scheduled, their start times are fixed in the rest of computation of CONSTRUCT). This will facilitate the computation of CONSTRUCT substantially.

To process setup activities $\sigma(j)$ in the modes specified by its immediate predecessor $prev(\sigma(j))$ on the machine $r(\sigma(j))$, the procedure CONSTRUCT is modified as follows.

- Regardless of the mode $m_{\sigma(j)}$ of the input (m, π) , the mode of a setup activity $\sigma(j)$ is determined in Step 2 by the activity $prev(\sigma(j))$ and the machine resource $r(\sigma(j))$.
- Once $\sigma(j)$ is scheduled, the immediate precedence constraint

$$prev(\sigma(j)) \prec \prec_{r(\sigma(j))} \sigma(j) \quad (25.9)$$

is introduced so that no activity between $prev(\sigma(j))$ and $\sigma(j)$ on $r(\sigma(j))$ is permitted. In case the start time of $\sigma(j)$ is canceled by backtracking in Step 3, constraint (25.9) is also removed.

25.3.2 Search space of tabu search

Our tabu search operates on the following search space:

$$\Lambda = \left\{ (m, \pi) \mid m_j \in M_j \text{ for all } j \in J, \pi \in \Pi \right\}, \quad (25.10)$$

where

$$\Pi = \{ \pi \mid \pi \text{ satisfies (25.7) and (25.8)} \}. \quad (25.11)$$

The quality of a solution (m, π) is evaluated by the penalty to the schedule (m, s) constructed from (m, π) by CONSTRUCT.

A feasible schedule is called *active*, if no activity can be made start earlier (while keeping others the same) without violating the feasibility [20]. When there are no immediate precedence constraints, the schedules obtained by CONSTRUCT are always active. However, there may exist active schedules that

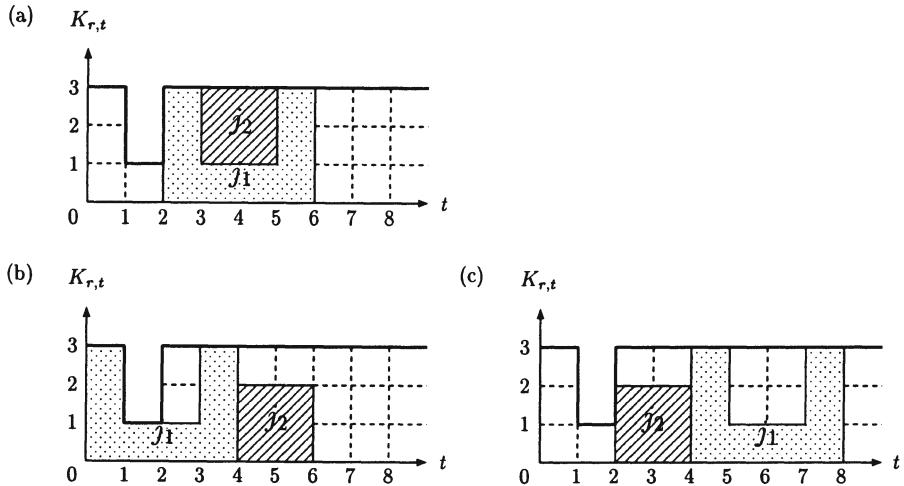


Figure 25.4 Examples of active schedules. The schedule (a) cannot be constructed by CONSTRUCT, while schedules (b) and (c) are constructed from lists (j_1, j_2) and (j_2, j_1) , respectively.

cannot be constructed by CONSTRUCT. To see this, we consider an example of two activities j_1 and j_2 that consume a renewable resource r as illustrated in Figure 25.4. The resource r is available by amount $K_{r,t} = 3$ in all periods $(t - 1, t]$, except that we have $K_{r,2} = 1$. Activity j_1 has processing time 4, and consumes r by amounts 3, 1, 1 and 3 in each period of the processing, respectively, while the processing time of activity j_2 is 2, and the requirement of r by j_2 is 2 in all processing periods. The renewable resource constraint r is hard, and there is no precedence constraint between j_1 and j_2 . Then we can see that the active schedule of Figure 25.4 (a) cannot be constructed by CONSTRUCT for any list. In fact, if a list π is (j_1, j_2) (resp., (j_2, j_1)), then the constructed schedule is Figure 25.4 (b) (resp., (c)), since each activity is sequentially scheduled at the earliest start time.

This observation tells that the set of schedules constructed from search space (25.10) by CONSTRUCT may not contain any optimal schedule $(\mathbf{m}^*, \mathbf{s}^*)$ of the given problem instance, if the constraints are complicated. Furthermore, the optimal schedule is not necessarily active in general, since the objective function represented by soft constraints (as explained in Section 25.2.5) may not be *regular* [20] in our RCPSP model. However, if the availabilities and requirements of all renewable resources do not vary with periods, and there is no immediate precedence constraint (as in the standard RCPSP), then it can be shown that any active schedule (\mathbf{m}, \mathbf{s}) can be constructed by CONSTRUCT from the solution $(\mathbf{m}, \pi) \in \Lambda$ such that all activities j are listed in π in the nondecreasing order of their start times s_j . In this case, for any regular objective function, our search space contains an optimal solution that provides an optimal schedule.

To guarantee the existence of optimal schedules for all instances of our generalized RCPSP, it seems necessary to use the direct representations $(\mathbf{m}, \boldsymbol{\pi})$ as solutions to search. However, the local search over this search space appears less efficient, as mentioned in the beginning of Section 25.3.1, and we face the trade-off between completeness and efficiency. As we shall see in Section 25.4, our algorithm, relying on the search space Λ of (25.10), turns out to be reasonably powerful, even though the search space may not contain optimal solutions for the full generality of RCPSP.

25.3.3 Neighborhood

We define the neighborhood $N(\mathbf{m}, \boldsymbol{\pi})$ of the current solution $(\mathbf{m}, \boldsymbol{\pi})$ as the set of solutions obtained from $(\mathbf{m}, \boldsymbol{\pi})$ by applying one of the following three types of moves:

change_mod(j, m'_j), where $j \in J$ is a non-setup activity and $m'_j \in M_j$ satisfies $m'_j \neq m_j$ (where m_j is the current mode of j): Modify \mathbf{m} by changing the mode m_j of j into another mode m'_j . For a setup activity $\sigma(j)$, this move is not defined, since its mode is uniquely determined in CONSTRUCT from its immediate predecessor $prev(\sigma(j))$.

shift_aft(j_1, j_2), where $j_1 (= \pi(i_1))$ and $j_2 (= \pi(i_2))$, which belong to strongly connected components U_1 and U_2 in digraph D^*_\prec ($U_1, U_2 \in \mathcal{U}$), respectively, satisfy

$$i_1 < i_2, j_1 \not\prec j_2, \text{ and } U_1 \not\prec U_2. \quad (25.12)$$

If $U_1 = U_2$, then shift j_1 to the position immediately after j_2 , together with all the activities $j' = \pi(i')$ such that $i_1 < i' < i_2$ and $j_1 \prec j'$; otherwise (i.e., $U_1 \neq U_2$), shift all activities in U_1 to the positions immediately after the last activity in U_2 , together with all activities $j' = \pi(i') \in U'$ such that $i_1 < i' < i_2$ and $U_1 \prec U'$. The relative positions among all the activities moved in this process are preserved. Notice that the resulting list $\boldsymbol{\pi}'$ also belongs to the Π of (25.11).

shift_bef(j_1, j_2), where $j_1 (= \pi(i_1) \in U_1)$ and $j_2 (= \pi(i_2) \in U_2)$ satisfy (25.12): Modify $\boldsymbol{\pi}$ by shifting j_2 to the position before j_1 , together with some other activities, in a symmetric manner to *shift_aft* so that the resulting list also belongs to the Π of (25.11).

Since it appears too time consuming to generate all solutions in neighborhood $N(\mathbf{m}, \boldsymbol{\pi})$ in each iteration, we try only those moves which may lead to a reduction of the penalty (25.5). To do this, for each of the currently violated constraint C_ℓ , we first calculate the move set $MV_\ell(\mathbf{m}, \boldsymbol{\pi})$, which consists of the moves that temporarily reduce the penalty of C_ℓ (its precise definition will be given later). The whole set is then denoted as

$$MV(\mathbf{m}, \boldsymbol{\pi}) = \bigcup_{C_\ell \text{ is violated}} MV_\ell(\mathbf{m}, \boldsymbol{\pi}).$$

Let $\tilde{N}(\mathbf{m}, \boldsymbol{\pi})$ denote the set of solutions obtained by applying the moves in $MV(\mathbf{m}, \boldsymbol{\pi})$ to $(\mathbf{m}, \boldsymbol{\pi})$. The neighborhood $N(\mathbf{m}, \boldsymbol{\pi})$ is reduced to $\tilde{N}(\mathbf{m}, \boldsymbol{\pi})$. Since the size of $\tilde{N}(\mathbf{m}, \boldsymbol{\pi})$ depends on the number of violated constraints, $\tilde{N}(\mathbf{m}, \boldsymbol{\pi})$ may still be too large if there are many violated constraints. In such a case, we also try a strategy that decreases the size of neighborhood by randomly choosing a subset of moves from $MV(\mathbf{m}, \boldsymbol{\pi})$ (some computational results will be reported in Sections 25.4.2.1 and 25.4.2.2).

The definition of $MV_\ell(\mathbf{m}, \boldsymbol{\pi})$ depends on the type of constraint C_ℓ . We now describe how we constructed $MV_\ell(\mathbf{m}, \boldsymbol{\pi})$ in our implementation, but other choices may also be possible. Let us first define the digraph $G(\mathbf{m}, \boldsymbol{\pi}) = (J, A(\mathbf{m}, \boldsymbol{\pi}))$: The vertex set J is the set of activities, and arcs (j_1, j_2) in $A(\mathbf{m}, \boldsymbol{\pi})$ imply that j_1 is an obstacle to schedule j_2 before s_{j_2} in CONSTRUCT. In other words, if j_1 had not been scheduled before j_2 , the start time of j_2 would be earlier than s_{j_2} . There are four possibilities for why j_2 could not be scheduled before s_{j_2} :

- a. there is a predecessor of j_2 whose completion time is s_{j_2} ,
- b. if j_2 was scheduled at $t' (< s_{j_2})$, some hard renewable resource constraints would be violated,
- c. if j_2 was scheduled at $t' (< s_{j_2})$, some immediate precedence constraint would be violated, or
- d. backtracking in Step 3 delayed the start time of j_2 to s_{j_2} .

The cases a–c correspond to the conditions 1–3 in Step 2 of CONSTRUCT, respectively. We define

$$A(\mathbf{m}, \boldsymbol{\pi}) = A_P(\mathbf{m}, \boldsymbol{\pi}) \cup A_R(\mathbf{m}, \boldsymbol{\pi}) \cup A_I(\mathbf{m}, \boldsymbol{\pi}) \cup A_B(\mathbf{m}, \boldsymbol{\pi}),$$

where A_P , A_R , A_I and A_B are given as follows, corresponding to the above four cases.

$$A_P(\mathbf{m}, \boldsymbol{\pi}) = \{(j_1, j_2) \mid j_1 \prec j_2 \text{ and } c_{j_1} = s_{j_2}\},$$

$$A_R(\mathbf{m}, \boldsymbol{\pi}) = \left\{ (j_1, j_2) \mid \begin{array}{l} \text{For some } t' < s_{j_2}, j_2 \text{ could not be started at } \\ t' \text{ in CONSTRUCT because of the lack of a } \\ \text{resource which was also consumed by } j_1. \end{array} \right\},$$

$$A_I(\mathbf{m}, \boldsymbol{\pi}) = \left\{ (j_1, j_2) \mid \begin{array}{l} \text{For some } t' < s_{j_2} \text{ and } j \in J, j_2 \text{ could not be} \\ \text{started at } t' \text{ in CONSTRUCT due to imme-} \\ \text{diate precedence constraint } j \ll_r j_1 \text{ such that} \\ c_j \leq t' < s_{j_1}. \end{array} \right\},$$

and

$$A_B(\mathbf{m}, \boldsymbol{\pi}) = \left\{ (j_1, j_2) \mid \begin{array}{l} \text{a backtracking was executed in Step 3 (a) of} \\ \text{CONSTRUCT with } \boldsymbol{\pi}(i) = j_1 \text{ and } j' = j_2, \text{ or} \\ \text{in Step 3 (b) with } j' = j_1 \text{ and } \bar{j} = j_2. \end{array} \right\}.$$

In our code, for each solution $(\mathbf{m}, \boldsymbol{\pi})$, the digraph $G(\mathbf{m}, \boldsymbol{\pi})$ is constructed in the process of applying CONSTRUCT to $(\mathbf{m}, \boldsymbol{\pi})$.

In the tabu search algorithm developed for the standard RCPSP by Baar et al. [1], a digraph similar to our $G(\mathbf{m}, \boldsymbol{\pi})$ is used to define the neighborhood. But there are differences in the definition of the arc set, even if restricted to the standard RCPSP. The arc set of [1] is defined by $\{(j_1, j_2) \mid c_{j_1} = s_{j_2}\}$. Thus, even when $\pi^{-1}(j_1) > \pi^{-1}(j_2)$ holds (i.e., j_1 is scheduled later than j_2), the digraph of [1] has arc (j_1, j_2) , if $c_{j_1} = s_{j_2}$ holds, while ours does not. As a result, our neighborhood appears to represent more directly the conflicts encountered in the process of scheduling activities.

Now, for a constraint C_ℓ currently violated by solution $(\mathbf{m}, \boldsymbol{\pi})$, we implement the following three types (i)(ii)(iii) of move sets $MV_\ell(\mathbf{m}, \boldsymbol{\pi})$ in our code, as these appear to be effective to handle the constraints of many existing problems. To handle other types of constraints, however, we may need to introduce different move sets; this is a topic of our future research.

(i) C_ℓ is a renewable resource constraint for $r \in R^{re}$ in period $(t - 1, t]$. To reduce the penalty of C_ℓ , we try to change the modes or the start times of the activities j that consume r in $(t - 1, t]$. Let J' denote the set of such activities j . For each of $j \in J'$, let $\mathbf{v}^-(j) = (v_1^-, v_2^-, \dots, v_{n^-}^- = j)$ and $\mathbf{v}^+(j) = (j = v_1^+, v_2^+, \dots, v_{n^+}^+)$ be two maximal simple directed paths to j and from j in $G(\mathbf{m}, \boldsymbol{\pi})$, respectively (if there is more than one candidate, choose arbitrarily), where we say that a simple directed path \mathbf{v} is *maximal* if it is not contained in any other simple directed path to j and from j , respectively. Let m_v be the current mode of activity v . The move set is defined by

$$MV_\ell(\mathbf{m}, \boldsymbol{\pi}) = \bigcup_{j \in J'} CM(j) \cup SB(j) \cup SA(j), \quad (25.13)$$

where

$$CM(j) = \left\{ \text{change_mod}(v, m'_v) \mid \begin{array}{l} v \text{ is a non-setup activity in} \\ \mathbf{v}^-(j), m'_v \in M_v \text{ and } m'_v \neq m_v \end{array} \right\},$$

$$SB(j) = \{shift_bef(v^-, j) \mid v^- \text{ is in } \mathbf{v}^-(j) \text{ and } (v^-, j) \text{ satisfies (25.12)}\},$$

and

$$SA(j) = \{shift_aft(j, v^+) \mid v^+ \text{ is in } \mathbf{v}^+(j) \text{ and } (j, v^+) \text{ satisfies (25.12)}\}.$$

Here, for the activities in $\mathbf{v}^+(j)$, the moves *change_mod* are not considered, since they have less influence on the start time of j .

(ii) C_ℓ is a constraint related to the mode m_j of an activity j . C_ℓ is typically a nonrenewable resource constraint. In general, it is described by a linear inequality (25.4) with $\alpha_{j, m_j} \neq 0$. In this case, we try to change the mode of j . If j is a setup activity, we try to change its mode by shifting its position in $\boldsymbol{\pi}$, as its mode is determined by its immediate predecessor (as described in Section 25.2.4). The move set is then given by

$$\begin{aligned}
MV_\ell(\mathbf{m}, \boldsymbol{\pi}) = & \left\{ change_mod(j, m'_j) \mid \begin{array}{l} j \text{ is a non-setup activity,} \\ m'_j \in M_j \text{ and } \alpha_{j,m'_j} \neq \alpha_{j,m_j} \end{array} \right\} \\
& \cup \bigcup_{\sigma(j) \in J''} SB(\sigma(j)) \cup SA(\sigma(j)),
\end{aligned} \tag{25.14}$$

where $J'' = \{j \mid j \text{ is a setup activity such that } \alpha_{j,m_j} \neq 0\}$, and move sets $SB(\sigma(j))$ and $SA(\sigma(j))$ are defined in the same manner as in case (i).

(iii) C_ℓ is a constraint given by a linear inequality (25.4), for which $\beta_j \neq 0$ holds for some activity j . If $\beta_j > 0$ holds, for example, the penalty of C_ℓ can be decreased by making an activity j start earlier (see e.g., (25.6)). As in cases (i) and (ii), maximal simple paths $v^-(j)$ and $v^+(j)$ are also utilized. Let J^+ and J^- denote $\{j \mid \beta_j > 0\}$ and $\{j \mid \beta_j < 0\}$, respectively. Then, the move set is defined by

$$\begin{aligned}
MV_\ell(\mathbf{m}, \boldsymbol{\pi}) = & \bigcup_{j \in J^+} \left\{ shift_bef(v_i, v_{i+1}) \mid \begin{array}{l} (v_i, v_{i+1}) \text{ is an arc in} \\ v^-(j) \text{ and satisfies} \\ (25.12) \end{array} \right\} \\
& \cup \bigcup_{j \in J^-} SA(j) \cup \bigcup_{j \in J^+ \cup J^-} CM(j).
\end{aligned} \tag{25.15}$$

The moves of type *shift_bef* in the first set is different from $SB(j)$ of case (i). Although other definitions are also possible, this is chosen in our implementation, because in case (iii), move set $SB(j)$ sometimes becomes quite small and is not effective; e.g., if C_ℓ is (25.6) for the minimization of makespan, then $SB(j = sink)$ becomes empty.

In passing, it is noted that, if the problem being solved is a standard RCPSP (in which constraint (25.6) is employed), a maximal simple path $v^-(sink) = (v_1^-, v_2^-, \dots, v_{n^-}^- = sink)$ chosen in (iii) to determine the move set is always a critical (i.e., longest) path, since it satisfies $s_{v_1^-} = 0$ and $c_{v_i^-} = s_{v_{i+1}^-}$ for all $i = 1, 2, \dots, n^- - 1$. Since it is required to make such a critical path non-critical in order to decrease the makespan, the idea of eliminating a critical path is often used in algorithms for the jobshop scheduling problem [3], and also for the standard RCPSP (e.g., [1]). The moves *shift_bef* in (25.15) have similar effects, and, in such special cases, our neighborhood may be said to embody similar ideas.

25.3.4 Example

We show an example to illustrate solutions $(\mathbf{m}, \boldsymbol{\pi})$, their schedules (\mathbf{m}, \mathbf{s}) , digraph $G(\mathbf{m}, \boldsymbol{\pi})$ and how $MV_\ell(\mathbf{m}, \boldsymbol{\pi})$ are actually constructed.

Figure 25.5 gives an example consisting of two machines and three jobs. Each job i ($i = 1, 2, 3$) consists of two activities $j(i, 1)$ and $j(i, 2)$ with precedence

$$D_{\prec}^* = (J, A_{\prec}^*)$$

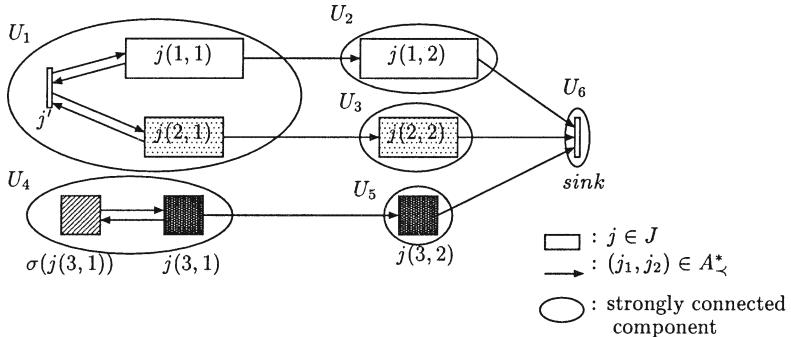


Figure 25.5 The digraph $D_{\prec}^* = (J, A_{\prec}^*)$ of the example.

constraint $j(i, 1) \prec j(i, 2)$. Activities $j(1, 1)$, $j(2, 2)$ and $j(3, 2)$ are processed on machine 1, and the rest are processed on machine 2. The processing times of the activities in job i are defined to be $4 - i$. For simplicity, all activities have only one mode. In addition to the resource constraints of two machines, r_1 and r_2 , and the above three precedence constraints, this problem has a constraint that $j(1, 1)$ and $j(2, 1)$ must start at the same time; hence a fictitious activity j' and a fictitious renewable resource r_0 are introduced, and two immediate precedence constraints $j' \ll_{r_0} j(1, 1)$ and $j' \ll_{r_0} j(2, 1)$ are added (see Section 25.2.3). Furthermore, to start the processing of $j(3, 1)$, the setup operation of 1 period is required on machine r_2 , unless $j(1, 2)$ is processed just before $j(3, 1)$ on machine r_2 . As explained in Section 25.2.4, the setup activity $\sigma(j(3, 1))$ is introduced together with its immediate precedence constraint $\sigma(j(3, 1)) \ll_{r_2} j(3, 1)$. The mode set of $\sigma(j(3, 1))$ is given by {dummy, setup}, where “dummy” has processing time 0, and “setup” is associated with the setup operation. Setup activity $\sigma(j(3, 1))$ is in mode “dummy” (resp., “setup”), if $j(1, 2)$ is (resp., is not) immediately preceding $\sigma(j(3, 1))$ on machine r_2 .

The digraph D_{\prec}^* (defined in Section 25.3.1) of this problem is shown in Figure 25.5. Activities j' , $j(1, 1)$ and $j(2, 1)$, and activities $\sigma(j(3, 1))$ and $j(3, 1)$ belong to the same strongly connected components U_1 and U_4 , respectively. Others belong to distinct components U_i , $i = 2, 3, 5, 6$, each consisting of a single activity.

For this problem, we now consider three soft constraints, respectively. For each of them, we explain how our algorithm determines the move sets, and reduces the penalties of these constraints.

Let us first assume that the current list is

$$\pi_0 = (\sigma(j(3, 1)), j(3, 1), j', j(1, 1), j(2, 1), j(3, 2), j(1, 2), j(2, 2), \text{sink}),$$

whose digraph $G(m, \pi_0)$ is given in Figure 25.6. Figure 25.6 may also be viewed as the Gantt chart of the schedule constructed from π_0 by CONSTRUCT (in

$$G(\mathbf{m}, \boldsymbol{\pi}_0 = (\sigma(j(3,1)), j(3,1), j', j(1,1), j(2,1), j(3,2), j(1,2), j(2,2), \text{sink}))$$

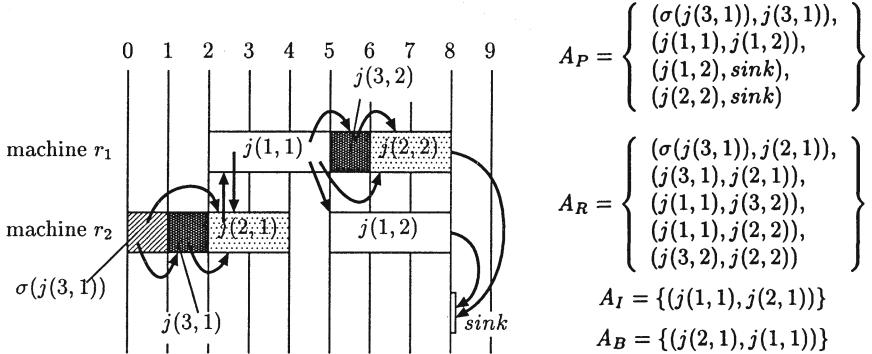


Figure 25.6 The digraph $G(\mathbf{m}, \boldsymbol{\pi}_0) = (J, A(\mathbf{m}, \boldsymbol{\pi}_0))$.

which fictitious activity j' is omitted). Notice that activities $\sigma(j(3,1))$ and $j(3,1)$, and activities $j', j(1,1)$ and $j(2,1)$ are contained in one strongly connected components, respectively, and hence consecutively ordered in $\boldsymbol{\pi}_0$. Hence $\boldsymbol{\pi}_0$ belongs to Π of (25.11). In the computation of CONSTRUCT, the mode of $\sigma(j(3,1))$ is set to “setup”, as $j(1,2)$ is not processed just before $\sigma(j(3,1))$ on machine r_2 . Activity $j(1,1)$ was first scheduled at time 0, but later delayed by the backtracking in Step 3 due to the violation of $j' \prec_{r_0} j(2,1)$ (this fictitious activity j' is scheduled at time 0); thus $G(\mathbf{m}, \boldsymbol{\pi})$ has arc $(j(2,1), j(1,1)) \in A_B$.

First, we assume that there is a soft renewable resource constraint C_{t_1} for $r' \in R^{re}$ and period $(5, 6]$, where r' is consumed by only activity $j(3,2)$ and not available in period $(5, 6]$ (but available in all other periods). As we can see from Figure 25.6, constraint C_{t_1} is violated by the current solution $(\mathbf{m}, \boldsymbol{\pi}_0)$. In this case, move set (i) in Section 25.3.3 applies, and two simple paths $v^-(j(3,2)) = (\sigma(j(3,1)), j(3,1), j(2,1), j(1,1), j(3,2))$ and $v^+(j(3,2)) = (j(3,2), j(2,2), \text{sink})$, which are respectively maximal, are chosen. Based on these, the move set (25.13) is determined as

$$\begin{aligned} MV_{t_1}(\mathbf{m}, \boldsymbol{\pi}_0) &= \{shift_bef(j(2,1), j(3,2)), shift_bef(j(1,1), j(3,2))\} \\ &\cup \{shift_aft(j(3,2), j(2,2))\}. \end{aligned}$$

Three moves $shift_bef(\sigma(j(3,1)), j(3,2))$, $shift_bef(j(3,1), j(3,2))$ and $shift_aft(j(3,2), \text{sink})$ are excluded because precedence constraints $\sigma(j(3,1)) \prec j(3,2)$, $j(3,1) \prec j(3,2)$ and $j(3,2) \prec \text{sink}$ violate (25.12).

As an example of move, let us apply $shift_bef(j(2,1), j(3,2))$ to $\boldsymbol{\pi}_0$. The resulting list becomes

$$\boldsymbol{\pi}_1 = (\sigma(j(3,1)), j(3,1), j(3,2), j', j(1,1), j(2,1), j(1,2), j(2,2), \text{sink}).$$

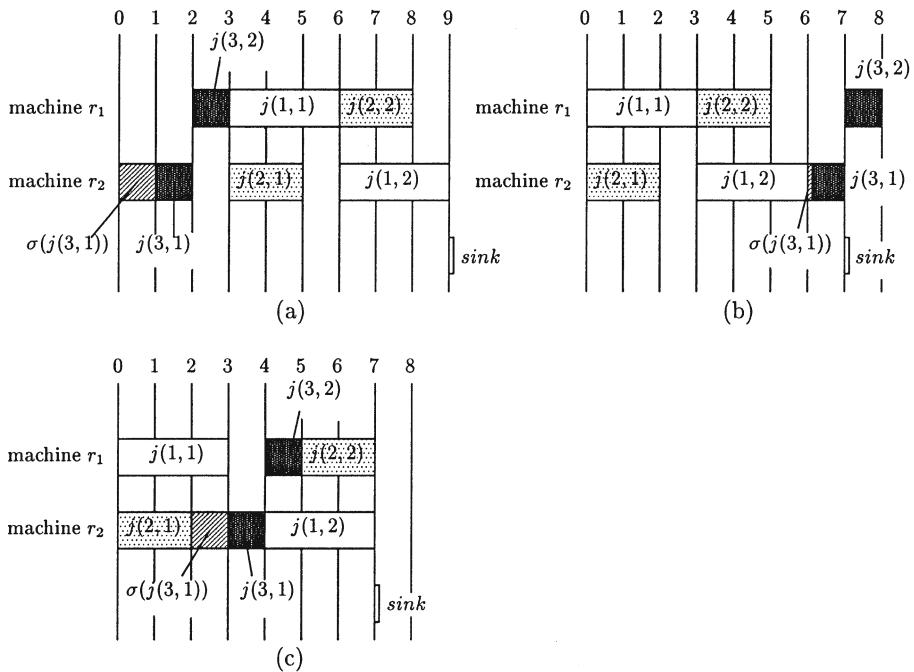


Figure 25.7 The schedules constructed from (a) (m, π_1) , (b) (m, π_2) and (c) (m, π_3) by CONSTRUCT.

Recall that activities j' , $j(1,1)$ and $j(2,1)$ belong to the same strongly connected component U_1 , and were moved after $j(3,2)$ by condition (25.8). Figure 25.7 (a) illustrates the schedule constructed from (m, π_1) by CONSTRUCT, in which C_{ℓ_1} is satisfied.

As the second example, we consider the minimization of the cost of the setup operation $\sigma(j(3,1))$ (which is positive if this operation is executed, but is 0 otherwise). As explained in Section 25.2.5, this is described by a soft nonrenewable resource constraint

$$C_{\ell_2} : x_{\sigma(j(3,1)), \text{setup}} \leq 0.$$

Assume again that the current solution is (m, π_0) (note that we do not consider constraint C_{ℓ_1} in this example). C_{ℓ_2} is violated by (m, π_0) , and move set (ii) of Section 25.3.3 is generated. Here, $J'' = \{\sigma(j(3,1))\}$ holds, and we choose two maximal simple directed paths $v^-(\sigma(j(3,1))) = (\sigma(j(3,1)))$ and $v^+(\sigma(j(3,1))) = (\sigma(j(3,1)), j(3,1), j(2,1), j(1,1), j(1,2), \text{sink})$. The move set $MV_{\ell_2}(m, \pi_0)$ of (25.14) is given by

$$MV_{\ell_2}(\mathbf{m}, \pi_0) = \left\{ \begin{array}{l} shift_aft(\sigma(j(3,1)), j(2,1)), shift_aft(\sigma(j(3,1)), j(1,1)), \\ shift_aft(\sigma(j(3,1)), j(1,2)) \end{array} \right\},$$

as $\sigma(j(3,1)) \prec j(3,1) \prec sink$ holds. If we apply a move $shift_aft(\sigma(j(3,1)), j(1,2))$ to π_0 , then the new list becomes

$$\pi_2 = (j', j(1,1), j(2,1), j(1,2), \sigma(j(3,1)), j(3,1), j(3,2), j(2,2), sink),$$

since $\sigma(j(3,1))$ and $j(3,1)$ are contained in the same strongly connected component, and $\sigma(j(3,1)) \prec j(3,2)$ holds. From this list, the schedule that requires no setup operation is obtained by CONSTRUCT, which is illustrated in Figure 25.7 (b).

Finally, we consider the minimization of makespan by introducing a soft constraint C_{ℓ_3} of type (25.6). C_{ℓ_3} is violated by the solution (\mathbf{m}, π_0) , and move set of type (iii) is generated, where $J^+ = \{sink\}$ and $J^- = \emptyset$. Let a maximal simple directed path $v^-(sink) = (\sigma(j(3,1)), j(3,1), j(2,1), j(1,1), j(1,2), sink)$ be chosen for our purpose. The move set $MV_{\ell_3}(\mathbf{m}, \pi_0)$ of (25.15) is then given by

$$MV_{\ell_3}(\mathbf{m}, \pi_0) = \{shift_bef(j(3,1), j(2,1)), shift_bef(j(2,1), j(1,1))\}.$$

The moves $shift_bef(\sigma(j(3,1)), j(3,1))$ and $shift_bef(j(1,1), j(1,2))$ are not considered because $\sigma(j(3,1)) \prec j(3,1)$ and $j(1,1) \prec j(1,2)$ hold. By applying a move $shift_bef(j(3,1), j(2,1))$ to π_0 , for example, we obtain the list

$$\pi_3 = (j', j(1,1), j(2,1), \sigma(j(3,1)), j(3,1), j(3,2), j(1,2), j(2,2), sink).$$

Recall again that activities j' , $j(1,1)$ and $j(2,1)$, and activities $\sigma(j(3,1))$ and $j(3,1)$, respectively, belong to the same strongly connected components. Using this, the makespan is reduced from 8 to 7. The resulting schedule is illustrated in Figure 25.7 (c).

25.3.5 Implementation of tabu search

In our tabu search, an initial solution is randomly selected from the search space Λ of (25.10), and then repeatedly replaced by its best non-tabu neighbor until a stopping criterion is achieved; e.g., the predetermined number of iterations or computational time is reached, or a satisfactory schedule is found.

The tabu list in our algorithm is implemented as follows. For each move, we define its *attribute* by the pair of move type (“change_mod” or “shift”) and the corresponding activity; e.g., $(change_mod, j)$, $(shift, j_1)$ and $(shift, j_2)$ are the attributes of $change_mod(j, m'_j)$, $shift_aft(j_1, j_2)$ and $shift_bef(j_1, j_2)$, respectively. The tabu list T keeps the attributes of all the moves executed in τ recent iterations, where τ is a program parameter called *tabu tenure*. As stated in Step k (3) below, it prohibits all moves that possess an attribute in the tabu list.

The value of τ has a large influence on the performance of tabu search. In our algorithm, τ is adaptively controlled by the method introduced in the CSP code of [17]. Although we do not give detailed computational results here, this control mechanism of τ performs well also in our RCPSP code.

The outline of our algorithm for RCPSP is described as follows.

Tabu search algorithm for RCPSP

Step 0 (initialization): Generate an initial solution $(\mathbf{m}^{(1)}, \boldsymbol{\pi}^{(1)})$ randomly. Initialize the incumbent schedule $(\mathbf{m}^*, \mathbf{s}^*)$ to the schedule obtained from $(\mathbf{m}^{(1)}, \boldsymbol{\pi}^{(1)})$ by applying CONSTRUCT. Set $k := 1$.

Step k (k -th iteration):

- (1) Calculate the move set $MV(\mathbf{m}^{(k)}, \boldsymbol{\pi}^{(k)})$ and the neighborhood $\tilde{N}(\mathbf{m}^{(k)}, \boldsymbol{\pi}^{(k)})$ for the current solution $(\mathbf{m}^{(k)}, \boldsymbol{\pi}^{(k)})$.
- (2) Apply CONSTRUCT to each solution $(\mathbf{m}, \boldsymbol{\pi}) \in \tilde{N}(\mathbf{m}^{(k)}, \boldsymbol{\pi}^{(k)})$, and evaluate the resulting schedule (\mathbf{m}, \mathbf{s}) by its penalty $p(\mathbf{m}, \mathbf{s})$.
- (3) For the non-tabu neighbor $(\mathbf{m}', \boldsymbol{\pi}')$ having the smallest penalty, let $(\mathbf{m}^{(k+1)}, \boldsymbol{\pi}^{(k+1)}) := (\mathbf{m}', \boldsymbol{\pi}')$. If $p(\mathbf{m}', \mathbf{s}') < p(\mathbf{m}^*, \mathbf{s}^*)$, then update $(\mathbf{m}^*, \mathbf{s}^*) := (\mathbf{m}', \mathbf{s}')$.
- (4) If the stopping criterion is met, then output $(\mathbf{m}^*, \mathbf{s}^*)$ and halt; otherwise, let $k := k + 1$ and go to Step k .

25.4 COMPUTATIONAL EXPERIMENTS

Our algorithm was coded in C, and tested on many benchmarks of RCPSP, and some practical problems. In this section, we report some of such computational results. All experiments was conducted on a workstation Sun Ultra 2 (300MHz, 1G bytes memory).

25.4.1 Benchmarks from PSPLIB

Kolisch et al. [11] generated a large number of RCPSP instances systematically, which are available in the Project Scheduling Problem LIBRARY (PSPLIB) together with their optimum or best known values. All instances are from the standard RCPSP in the sense that the available amounts of resources do not change with periods, and that no immediate precedence constraints are considered. We solved four types of instances j60.sm, j90.sm, j120.sm and j30.mm, which are 2110 instances in total. Statistics of these instances such as the number of activities, the number of modes in which each activity can be processed, and the number of resources are shown in Table 25.1. All instances have precedence constraints as well as resource constraints, and the objective is to minimize the makespan.

For each instance in j60.sm, j90.sm, j120.sm and j30.mm, we ran our algorithm until 5000 iterations are reached, respectively. Computational results are summarized in Table 25.2. As listed in Table 25.2, the best known schedules (and sometimes even better ones) were found for many instances. (By running our algorithm repeatedly with different initial solutions, we could increase the number of instances with improved best known values. Table 25.3 shows the

instance type	#activities	#modes	#renewable resources	#nonrenewable resources
j60.sm	60	1	4	0
j90.sm	90	1	4	0
j120.sm	120	1	4	0
j30.mm	30	3	2	2

Table 25.1 Properties of four instance types in PSPLIB.

instance type	#instances	#instances whose best known values are found ¹	CPU time per run (in seconds)		
			min	average	max
j60.sm	480	370 (1)	3.2	13.6	78.5
j90.sm	480	367 (9)	5.5	29.3	212.3
j120.sm	600	202 (40)	8.4	109.4	575.8
j30.mm	550	360 (50)	7.2	16.8	41.3

- [1] The best known values are as of August 2000. Figures in parenthesis are the numbers of instances whose best known values were improved by our code.

Table 25.2 Computational results for benchmarks in PSPLIB.

current status, as of August 2000, of the number of instances whose best values were improved by our codes, and the number of instances whose optimal solutions were found; i.e., these makespans are equal to their lower bounds.)

In the last three columns in Table 25.2, the minimum, average and maximum computational times required for each run (i.e., 5000 iterations) are listed. It seems that the deviation of the computational time is quite large. This is because the size of neighborhood varies from instance to instance. As mentioned in Section 25.3.3, the neighborhood $\tilde{N}(\mathbf{m}, \boldsymbol{\pi})$ is based on the type of constraints violated, and in the case of the standard RCPSP, its size highly depends on the number of activities in a critical path (since the move set is given by (25.15)). Figure 25.8 shows the relationship between the computational time and the

instance type	#instances whose best values are improved by our code	#instances with proven optimality
j60.sm	4	0
j90.sm	14	4
j120.sm	79	33
j30.mm	126	0

Table 25.3 The number of instances whose best known values were improved by our code (as of August 2000).

makespan of the best schedule obtained by our code for 600 instances in j120.sm. From this, we observe a tendency that more computational time is required for larger makespan (since the critical path tends to contain more activities).

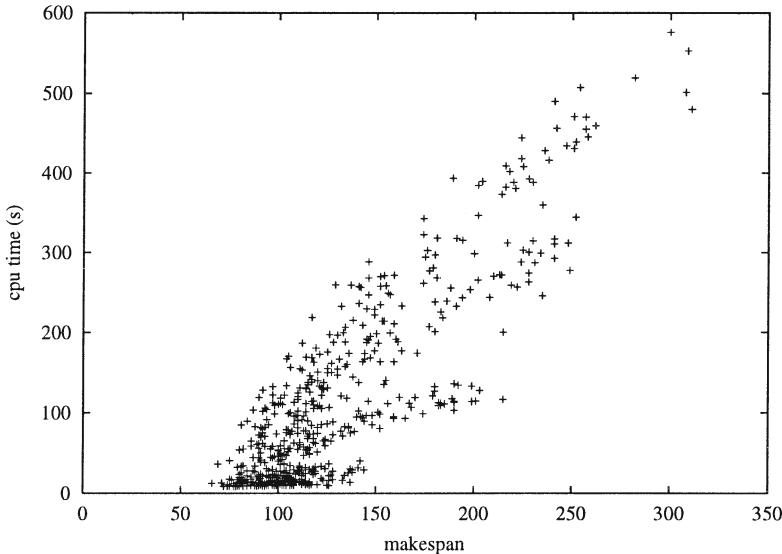


Figure 25.8 The relationship between computational time and makespan.

Next, we compare our results for j120.sm with those obtained by three existing RCPSP algorithms; genetic algorithm by Hartmann (H) [8], Lagrangian heuristic by Möhring et al. (MSSU) [14] and ant colony optimization algorithm by Merkle et al. (MMS) [13]. Table 25.4 gives the average deviation of the best obtained values from the critical path based lower bounds. For each instance, H, MMS and our algorithm have been executed for 5000 iterations, respectively. Each run of MSSU and our algorithm consumed the average and maximum computational times of 65 and 711 seconds (on a Sun Ultra 2 with 200MHz clock pulse), and 109.4 and 575.8 seconds (on a Sun Ultra 2 with 300MHz clock pulse), respectively. According to [8, 13], algorithms H and MMS required 3.07 seconds (on a Pentium 133MHz) and 25 seconds (on a Pentium III 500MHz) for 1000 iterations, respectively. These results indicate that our tabu search is competitive with other powerful codes in its solution quality, although our algorithm is slower. In closing, we emphasize that our code is designed for the generalized RCPSP. In fact, among the tested four algorithms, only our code can be applied to instances in j30.mm, which are multi-mode RCPSPs.

25.4.2 Problems from real-world applications

The generalized RCPSP can encompass a wide variety of problems encountered in real life applications. In this subsection, we consider two such examples,

algorithm	deviation (%)
Genetic algorithm by Hartmann [8]	35.60
Our algorithm	35.86
Lagrangian heuristic by Möhing et al. [14]	36.2
Ant colony optimization by Merkle et al. [13]	36.65

Table 25.4 Average deviations from the critical path based lower bounds over all 600 instances in j120.sm.

which cannot be formulated as the standard RCPSP. We explain below how they are formulated as extended RCPSP instances and how we apply our code, together with their computational results. The first problem can be found in spinning mills in textile industry, for example, while the second problem can be found in some types of chemical plants.

25.4.2.1 Production scheduling with setup activities. We are given a set of activities J , a set of machines R , and the duration of the schedule T (days). J is partitioned into $|R|$ sets J_r such that $J = \cup_{r \in R} J_r$, and all activities $j \in J_r$ must be processed on machine r , each requiring consecutive p_j days (thus, each activity has only one mode). For each activity j , its due date d_j is also specified, and its cost function $\text{cost_due}(c_j, d_j)$ is given, which can take any form (in our experiment, it contains both earliness and tardiness penalties). To change the activity on machine r from j_1 to j_2 (where $j_1, j_2 \in J_r$), the setup operation $\text{setup}(j_1, j_2)$ of q consecutive days on the same machine r as well as the setup cost are required. The q is common to all setup operations, but the setup cost, $\text{cost_set}(j_1, j_2)$, depends on j_1 and j_2 . Since setup operations consume the setup resource, at most K_t machines can process setup operations in each day $t = 1, 2, \dots, T$. The K_t takes different values, depending on whether t is a weekday or not. Each machine must process exactly one activity (which may be a setup operation) everyday (i.e., there is no idle time). To make this possible, we assume that the data satisfy the constraints

$$\sum_{j \in J_r} p_j + q(|J_r| - 1) = T, \quad r \in R.$$

The objective is to minimize the total cost

$$\sum_{j \in J} \text{cost_due}(c_j, d_j) + \sum_{r \in R} \sum_{i=1}^{|J_r|-1} \text{cost_set}(a(r, i), a(r, i+1)), \quad (25.16)$$

where $a(r, i)$ denotes the i -th activity processed on machine r .

To formulate this problem as an extended RCPSP, we introduce

- the renewable resource r for each machine r ,

- the mode set with only one mode m_j for each $j \in J$, except that the setup activity $\sigma(j)$ for $j (\in J_r)$ has the mode set,

$$\{setup(j', j) \mid j' \neq j, j' \in J_r\} \cup \{\text{dummy}\},$$

and

- the renewable resource r_{set} common to all setup activities.

As explained in Section 25.2.4, each setup activity $\sigma(j)$ ($j \in J_r$) has the immediate precedence constraint $\sigma(j) \prec_r j$, and $\sigma(j)$ must be processed in mode “ $setup(j', j)$ ” (resp., “dummy”) if j' is immediately preceding $\sigma(j)$ on r (i.e., $j' = a(r, i)$ and $j = a(r, i + 1)$ for some i) (resp., if j is the first activity on r). Finally, $|J|$ constraints on completion times c_j ,

$$c_j \leq T, \quad j \in J, \quad (25.17)$$

are added so that no machine has idle time.

As noted in Section 25.2.5, the minimization of total cost (25.16) is attained via soft constraints. The minimization of the earliness and tardiness cost $\sum_j cost_due(c_j, d_j)$ is realized by soft renewable resource constraints, as described at the end of Section 25.2.5. The cost of setup activities, where setup activity $\sigma(j)$ in mode $setup(j', j)$ consumes $cost_set(j', j)$ of the resource, is described by a nonrenewable resource constraint,

$$\sum_{r \in R} \sum_{j \in J_r} \sum_{\substack{j' \neq j \\ j' \in J_r}} cost_set(j', j) x_{\sigma(j), setup(j', j)} \leq 0.$$

The left hand side represents the total cost of setup activities (i.e., the penalty of this constraint).

There may exist no feasible schedule to this problem. To obtain a practical schedule in such a case, we regard each setup resource constraint of amount K_t as a soft one with a large weight. In practice, its violation (i.e., too many setups in a day) may be resolved by allocating more manpower.

The instance we solved has $|J| = 77$, $|R| = 19$, $T = 30$ and $q = 1$, and each machine $r \in R$ is assigned 2–7 activities, and the processing times p_j vary between 1 and 25 days. As to setup activities, at most three machines are allowed on each weekday, but no machine is allowed on weekend or holiday. The cost $cost_set(j_1, j_2)$ of each setup activity in mode $setup(j_1, j_2)$ takes one of the two values $\{0, 1\}$, and the cost function $cost_due(c_j, d_j)$ for $j (\in J)$ is defined by

$$cost_due(c_j, d_j) = \begin{cases} 10, & c_j \leq d_j - 5, \\ 2k, & c_j = d_j - k, \quad 1 \leq k \leq 4, \\ 0, & c_j = d_j, \\ 10, & c_j = d_j + 1, \\ 16, & c_j = d_j + 2, \\ 18, & c_j = d_j + 3, \\ 20, & c_j \geq d_j + 4. \end{cases}$$

	Values of ρ [%]					
	5%	10%	30%	50%	80%	100%
number of runs in which the optimum schedule could be found	30/30	30/30	27/30	22/30	14/30	11/30
average CPU time (sec) required to find the optimum schedule	17.5	9.1	—	—	—	—

Table 25.5 Effect of the neighborhood reductions.

Since this problem has many soft constraints, the size of neighborhood $\tilde{N}(m, \pi)$ (see Section 25.3.3) is rather large. To demonstrate the effect of the random neighborhood reduction stated in Section 25.3.3, we executed six tabu search algorithms in which $\rho = 5, 10, 30, 50, 80$ and 100% of solutions in $\tilde{N}(m, \pi)$ are randomly sampled in each iteration, respectively. We ran our code 30 times for each value of ρ , setting the computational time to 180 seconds for each run. Table 25.5 shows the number of runs (out of 30) for each ρ in which we could find the optimum schedule (the optimality was proved by integer programming, as explained later), as well as the average computational time required to find the optimum schedule if we could find it in all 30 runs. From this result, we can see that the neighborhood reduction is quite effective, and the best value of ρ is 10% for this instance. However, at the current stage, some preliminary experiments are needed to find an appropriate value of ρ .

This problem can be formulated as an integer program (IP) by introducing $|J_r|!$ 0-1 variables for each machine r , corresponding to all permutations of activities in J_r . By solving the resulting integer program by a commercial package SOPT, we found the optimum solution with its optimality in one second (not including the computational time for generating and storing the data, which takes about ten seconds). Although this integer programming approach seems more effective for this small instance, our approach will become more useful for larger instances, since the number of 0-1 variables in IP formulation increases exponentially with the number of activities. To show this, we solved three more instances I_8, I_9 and I_{10} , in which the maximum numbers of activities assigned to one machine (i.e., $\max_r |J_r|$) are 8, 9 and 10, respectively. The IP approach could find optimal solutions for instances I_8 and I_9 in 2 and 21 seconds (requiring about 1 and 14 minute(s) for generating and storing the data), respectively, but could not solve instance I_{10} because the data size exceeds the memory bound of the computer. On the other hand, our approach could handle all instances I_8, I_9 and I_{10} . We ran our code 30 times for instances I_8, I_9 and I_{10} , consuming 50.6, 85.7 and 90.8 seconds, respectively, for each of the 30 runs. For instances I_8 and I_9 , we could obtain the optimal solutions (which were obtained by IP approach). Optimal solutions for instances I_{10} are not known.

25.4.2.2 Parallel machine shop scheduling with setup activities. The next example is the parallel shop scheduling problem treated in [16]. This problem consists of a set of jobs J , which are classified into G groups, and a set of parallel machines R . For each job $j \in J$, the set of machines R_j on which j can be processed and the total amount D_j of demanded days for processing j are specified. Each job can be processed on more than one machine with preemption, and be also processed on several machines in parallel. Jobs are scheduled on a day basis, and the schedule horizon is T days. This problem has the following constraints:

- Each machine can process only one job at a time, and runs even on weekends and holidays.
- Once the processing of a job is initiated, it must be continued for at least three days.
- Switching jobs within the same group on a machine takes no time, but changing a job into another job in a different group requires the setup construction of q days.
- There is no precedence constraint between jobs.
- The switching operations and setup constructions are possible only on weekdays, and their numbers which can be carried out in a day are at most K_{sw} and K_{set} , respectively.

The objectives are to minimize the weighted sum of the total amount (days) of unsatisfied demands and the number of switchings.

Since a job may be processed on more than one machine, for each pair of job j and machine $r \in R_j$, we prepare activity $a(j, r)$ with mode set

$$\{t \mid 3 \leq t \leq T\} \cup \{\text{dummy}\},$$

where we consider that job j is processed on machine r for t days (resp., is not processed on r) if activity $a(j, r)$ takes mode “ t ” (resp., “dummy”). We denote by $t_{a(j, r)}$ the number of days given to activity $a(j, r)$ on machine r . Furthermore, for each activity $a(j, r)$, we introduce setup activity $\sigma(a(j, r))$ whose mode set is $\{\text{setup, dummy}\}$. Hence, the number of activities is $(\sum_j |R_j|) \times 2$ in total. In addition to $|R|$ machine resources, two renewable resources r_{set} and r_{sw} are introduced to keep the numbers of setup constructions and switchings in a day within K_{set} and K_{sw} (or 0 in a holiday or weekend), respectively. All of these renewable resource constraints are considered hard. To obtain schedules whose horizon is T (days) and in which no machine has idle time, we impose the following constraints:

$$c_{a(j, r)} \leq T, \quad \text{for all pairs } j \text{ and } r \in R_j, \quad (25.18)$$

$$\sum_{j \in \{j \mid r \in R_j\}} t_{a(j, r)} + t_{\sigma(a(j, r))} \geq T, \quad \text{for all } r. \quad (25.19)$$

As explained in Section 25.2.5, these additional constraints (25.18) and (25.19) are treated as soft constraints C_ℓ with large weights w_ℓ in our code (w_ℓ is set to 100 in our experiments).

In general, a penalty function for a soft inequality constraint $C_\ell : f_\ell(\mathbf{m}, \mathbf{s}) \leq 0$ (including (25.18) and (25.19)) may be naturally defined by

$$p_\ell(\mathbf{m}, \mathbf{s}) = \max\{0, f_\ell(\mathbf{m}, \mathbf{s})\}. \quad (25.20)$$

However, it is observed in our experiment that allowing the search into infeasible region of soft constraints is useful to find many different feasible schedules. Thus, to allow some amount of violations (occasionally), the following penalty function appears to be more appropriate:

$$p_\ell(\mathbf{m}, \mathbf{s}) = \max \left\{ 0, \frac{f_\ell(\mathbf{m}, \mathbf{s})}{|f_\ell(\mathbf{m}, \mathbf{s})|} \right\}, \text{ or} \quad (25.21)$$

$$p_\ell(\mathbf{m}, \mathbf{s}) = \max \left\{ 0, \frac{f_\ell(\mathbf{m}, \mathbf{s})}{|f_\ell(\mathbf{m}, \mathbf{s})|} + \delta \cdot f_\ell(\mathbf{m}, \mathbf{s}) \right\}, \quad (25.22)$$

where $\delta > 0$ is a small value. The penalty function defined by (25.21) depends only on whether $f_\ell(\mathbf{m}, \mathbf{s})$ is positive (i.e., the constraint is violated) or not, and (25.22) is a compromise between (25.20) and (25.21).

The soft constraints associated to the objective function can be written by

$$\sum_{r \in R_j} t_{a(j,r)} \geq D_j, \quad \text{for all } j, \quad (25.23)$$

corresponding to minimizing the total amount of unsatisfied demands, and

$$m_{a(j,r)} = m_{\sigma(a(j,r))} = \text{"dummy"}, \quad \text{for all } a(j,r), \quad (25.24)$$

corresponding to minimizing the number of switchings (a switching operation is required if an activity is processed in non-dummy mode), respectively.

We solved an instance with $|J| = 11$, $|R| = 4$, $G = 7$, $T = 31$, $K_{\text{set}} = 2$, $K_{\text{sw}} = 3$, $q = 2$ and the objective function of

$0.5 \times (\text{the total amount of unsatisfied demands}) + (\text{the number of switchings})$.

Thus, we set the weights of soft constraints (25.23) and (25.24) to 0.5 and 1, respectively. We tried penalty functions (25.20), (25.21) and (25.22) with $\delta = 10^{-5}$ for constraints (25.18) (25.19), respectively. For each case, we ran our tabu search 30 times consuming 300 seconds in each run. The size of neighborhood is randomly reduced to $\rho = 50\%$ as in Section 25.4.2.1. The computational results are shown in Table 25.6, where the minimum, average and maximum penalty values are indicated. “—” in the last column means that no feasible schedule could be found in some runs, and such runs are excluded in the calculation of the average penalty. Among these three, penalty function (25.22) brought the best results. A best schedule (whose penalty value is 15.0) obtained by our

penalty functions for constraints (25.18) and (25.19)	min	average	max
$\max\{0, f_t(\mathbf{m}, \mathbf{s})\}$	24.0	30.1 ¹	— ²
$\max\left\{0, \frac{f_t(\mathbf{m}, \mathbf{s})}{ f_t(\mathbf{m}, \mathbf{s}) }\right\}$	16.0	19.3	23.0
$\max\left\{0, \frac{f_t(\mathbf{m}, \mathbf{s})}{ f_t(\mathbf{m}, \mathbf{s}) } + \delta \cdot f_t(\mathbf{m}, \mathbf{s})\right\}$	15.0	16.5	20.0

[1] the average of those runs in which feasible schedules could be found.

[2] In two runs (out of 30), we could not find any feasible schedule.

Table 25.6 Computational results with different penalty functions.

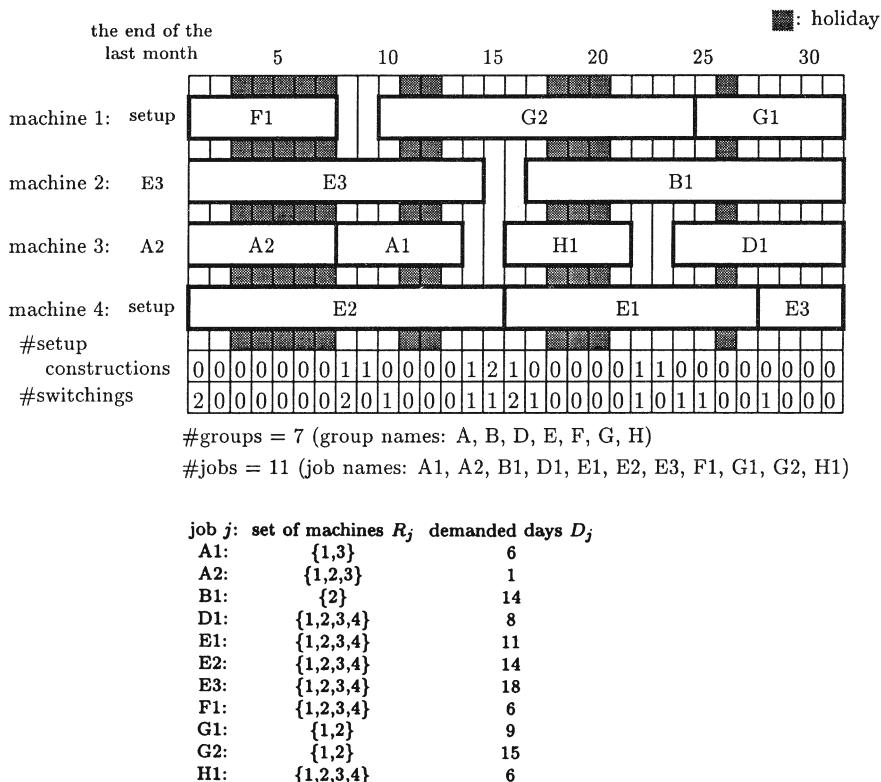


Figure 25.9 An example of schedules obtained by our RCPSP code.

experiments is illustrated in Figure 25.9, in which all hard constraints as well as (25.18) and (25.19) are satisfied. Note that job E3 is processed on two machines 2 and 4, in this schedule.

To evaluate our results, we formulated this problem as a mixed integer program, and solved it by using a commercial package CPLEX. We could find feasible schedules with penalties 16.5 and 16.0 in about 50 minutes and 9 hours, respectively, but no better schedule could be found in 10 hours of computational time. Thus, the schedules obtained by our code appear to be quite good.

25.5 CONCLUSION

In this paper, we first gave a generalization of RCPSP as a framework for handling complicated practical scheduling problems. Then, we proposed a tabu search algorithm, in which a list of activities is used to represent a solution so that only feasible schedules are searched efficiently. Our code was tested on a number of benchmarks of RCPSP and performed well. In addition to these benchmarks, we solved problems from real applications, in which setup activities are required, available amounts of resources may vary with periods, and objective functions may not be regular. For these problems, we succeeded to find practically usable schedules. We also demonstrated the effectiveness of the random neighborhood reduction. However, its performance highly depends on the sampling rate ρ , which is currently preset to an appropriate value by preliminary experiment; it would be a topic of future research to devise an automatic control mechanism of ρ .

Since our algorithm was developed as a general problem solver, for specific problems, it may not be able to attain as good performance as those algorithms specially designed for them (as observed in the computational results of Section 25.4.1). However, this would not deny the importance of our code, because it is easy to try, and can find satisfactory schedules in many practical problems, which cannot be formulated as the standard RCPSP.

As a possible avenue of future research, we point out that it may be possible to improve our code by customizing some of its components to the given problem; the following changes may be worth considering.

- As mentioned in Section 25.3.1, in general, optimal schedules may not be constructible from the solution space (25.10) by CONSTRUCT. Applying constructive methods other than CONSTRUCT may be able to overcome this limitation.
- In our code, the neighborhood is defined as a set of solutions obtained by applying only those moves which are effective to remove some violations of constraints. We discussed in Section 25.3.3 a few simple rules to select such moves for some typical constraints. To increase applicability, we can use other move sets specialized to the given problems.

Acknowledgments: We appreciate Prof. S. Morito of Waseda University for the data in Section 25.4.2.2. We are also grateful to Prof. T. Uno of

Tokyo Institute of Technology, Prof. M. Yagiura of Kyoto University, and three anonymous referees for their valuable comments and suggestions. This research was partially supported by the Grant-in-Aid for Scientific Research of Priority Areas “Algorithm Engineering as a New Paradigm: A Challenge to Hard Computation Problems” of the Ministry of Education, Science, Sports and Culture of Japan. The first author was supported by Research Fellowship of the Japan Society for the Promotion of Science for Young Scientists.

References

- [1] T. Baar, P. Brucker, and S. Knust. Tabu Search Algorithms and Lower Bounds for the Resource-Constrained Project Scheduling Problem. In: *Meta-heuristics: Advances and Trends in Local Search Paradigms for Optimization*, S. Voss, S. Martello, I. Osman, and C. Roucairol, editors, Kluwer, pages 1–18, 1998.
- [2] C.E. Bell and J. Han. A New Heuristic Solution Method in Resource-Constrained Project Scheduling. *Naval Research Logistics*, 38:315–331, 1991.
- [3] J. Błażewicz, W. Domschke, and E. Pesch, The Job Shop Scheduling Problem: Conventional and New Solution Techniques. *European Journal of Operational Research*, 93:1–33, 1996.
- [4] P. Brucker, A. Drexl, R. Möhring, K. Neumann, and E. Pesch. Resource-Constrained Project Scheduling: Notation, Classification, Models, and Methods. *European Journal of Operational Research*, 112:3–41, 1999.
- [5] J-H. Cho and Y-D. Kim. A Simulated Annealing Algorithm for Resource Constrained Project Scheduling Problems. *Journal of the Operational Research Society*, 48:736–744, 1997.
- [6] F. Glover. Tabu Search — Part I. *ORSA Journal on Computing*, 1:190–206, 1989.
- [7] S. Hartmann. A Competitive Genetic Algorithm for Resource-Constrained Project Scheduling. *Naval Research Logistics*, 45:733–750, 1998.
- [8] S. Hartmann. Self-Adapting Genetic Algorithms with an Application to Project Scheduling. Manuskripte aus den Instituten für Betriebswirtschaftslehre der Universität Kiel #506, 1999.
- [9] W. Herroelen, B. De Reyck, and E. Demeulemeester. Resource-Constrained Project Scheduling: A Survey of Recent Developments. *Computers and Operations Research*, 25:279–302, 1998.
- [10] R. Kolisch. Serial and Parallel Resource-Constrained Project Scheduling Methods Revisited: Theory and Computation. *European Journal of Operational Research*, 90:320–333, 1996.

- [11] R. Kolisch and A. Sprecher. PSPLIB — A Project Scheduling Library. *European Journal of Operational Research*, 96:205–216, 1997.
- [12] J-K. Lee and Y-D. Kim. Search Heuristics for Resource Constrained Project Scheduling. *Journal of the Operational Research Society*, 47:678–689, 1996.
- [13] D. Merkle, M. Middendorf, and H. Schmeck. Ant Colony Optimization for Resource-Constrained Project Scheduling. In: *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 893–900, 2000.
- [14] R.H. Möhring, A.S. Schulz, F. Stork, and M. Uetz. Solving Project Scheduling Problems by Minimum Cut Computations. Technical Report #680, Fachbereich Mathematik, Technische Universität, Berlin, 2000.
- [15] M. Mori and C.C. Tseng. A Genetic Algorithm for Multi-Mode Resource Constrained Project Scheduling Problem. *European Journal of Operational Research*, 100:134–141, 1997.
- [16] S. Morito, J. Imaizumi, and J.W. Park. A Mathematical Programming Approach to a Tightly Constrained Scheduling Problem (in Japanese). In: *Proceedings of the 1996 Production Scheduling Symposium*, pages 85–90, 1996 .
- [17] K. Nonobe and T. Ibaraki. A Tabu Search Approach to the Constraint Satisfaction Problem as a General Problem Solver. *European Journal of Operational Research*, 106:599–623, 1998.
- [18] J.H. Patterson. A Comparison of Exact Approaches for Solving the Multiple Constrained Resource, Project Scheduling Problem. *Management Science*, 30:854–867, 1984.
- [19] S.E. Sampson and E.N. Weiss. Local Search Techniques for Generalized Resource Constrained Project Scheduling Problem. *Naval Research Logistics*, 40:665–675, 1993.
- [20] A. Sprecher, R. Kolisch, and A. Drexl. Semi-Active, Active, Non-Delay Schedules for the Resource-Constrained Project Scheduling Problem. *European Journal of Operational Research*, 80:94–102, 1995.
- [21] P.R. Thomas and S. Salhi. A Tabu Search Approach for the Resource Constrained Project Scheduling Problem. *Journal of Heuristics*, 4:123–139, 1998.
- [22] M.G.A. Verhoeven. Tabu Search for Resource-Constrained Scheduling. *European Journal of Operational Research*, 106:266–276, 1998.

26 ANALYSING THE RUN-TIME BEHAVIOUR OF ITERATED LOCAL SEARCH FOR THE TRAVELLING SALESMAN PROBLEM

Thomas Stützle¹ and Holger H. Hoos²

¹Darmstadt University of Technology, Computer Science Department, Intellectics Group
Alexanderstr. 10, 64283 Darmstadt, Germany
stuetzle@informatik.tu-darmstadt.de

²University of British Columbia, Computer Science Department
2366 Main Mall, Vancouver, BC, V6T 1Z4 Canada
hoos@cs.ubc.ca

Abstract: Iterated Local Search (ILS) is currently the most successful metaheuristic for solving large Travelling Salesman Problems (TSPs). In this paper we study the behaviour of ILS algorithms by measuring and analysing run-time distributions. Our analysis shows that currently available ILS algorithms suffer from stagnation behaviour when searching for very high quality solutions. Based on this observation we propose two enhanced ILS algorithms for the TSP which avoid this stagnation behaviour and achieve significantly improved performance. More generally, our study demonstrates how the analysis of run-time distributions can provide the basis for characterising and improving the performance of state-of-the-art metaheuristics, such as Iterated Local Search.

26.1 INTRODUCTION

Metaheuristics strongly involve random decisions during the search process. Such random decisions are due to random initial solutions, randomised tie breaking criteria, randomised sampling of neighbourhoods, probabilistic acceptance criteria, etc. Because of their nondeterministic nature, the run-time required by such an algorithm to achieve a specific goal—like finding an optimal

solution to a given problem instance—is a random variable. Obviously, knowledge on the distribution of this random variable (the run-time distribution, RTD) can give valuable information for the analysis and the characterisation of an algorithm’s behaviour, provide a basis for the comparison of algorithms, and give hints on possible improvements of an algorithm’s performance [13]. To obtain empirical knowledge on an algorithm’s RTD when applied to a specific problem instance, one can run the algorithm several times, estimate *a posteriori* the RTD and possibly approximate the empirical RTD by a distribution function known from probability theory. Analysing the RTDs for all instances from a particular problem class, such as a specific type of symmetric TSPs, can reveal characteristics that correspond to the typical run-time behaviour of the given algorithm.

In this work we investigate the run-time behaviour of Iterated Local Search (ILS) for the well known Travelling Salesman Problem (TSP). ILS [5, 16, 22, 28] is a simple and powerful metaheuristic which has proven to be among the best performing approximation algorithms for the TSP [3, 16, 21]. By means of an RTD-based analysis, we show that several basic ILS algorithms for the TSP suffer from a type of stagnation behaviour which severely compromises ILS performance for large run times. Based on this observation we derive improved ILS variants and give computational results which demonstrate their strongly improved performance. More generally, our study exemplifies how an RTD-based analysis can be used to characterise the behaviour of algorithms and often facilitates improvements of their performance.

The remainder of this article is structured as follows. In Section 26.2 we introduce the TSP; Section 26.3 gives details on the ILS algorithms used in our study and Section 26.4 describes the RTD-based analysis of these algorithms’ run-time behaviour. We then present computational results with two improved ILS variants in Section 26.5 and complete our presentation with some concluding remarks in Section 26.6.

26.2 THE TRAVELLING SALESMAN PROBLEM

The TSP is an \mathcal{NP} -hard optimisation problem which is extensively studied in the literature [16, 18, 26]. It has become a standard test-bed for new algorithmic ideas and a good performance on the TSP is often taken as a proof for the usefulness of an algorithmic approach. Intuitively, the TSP is the problem of a salesman who wants to find a shortest possible round-trip that will allow him to visit a given set of customer cities each exactly once, starting from and ending in his home town. The TSP can be represented by a complete, weighted graph $G = (N, E, d)$ where N is the set of $n = |N|$ nodes (cities), E is the set of edges fully connecting the nodes, and d is a weight function which assigns to each edge $(i, j) \in E$ a number d_{ij} representing the distance between cities i and j . The TSP is the problem of finding a minimal length Hamiltonian circuit of the graph, where a Hamiltonian circuit is a closed tour visiting each node of G exactly once. For symmetric TSPs, the distances between the cities are independent of the direction of traversing the

```

procedure Iterated Local Search
   $s_0 = \text{GenerateInitialSolution}$ 
   $s^* = \text{LocalSearch}(s_0)$ 
  repeat
     $s' = \text{Perturbation}(s^*, \text{history})$ 
     $s^{*\prime} = \text{LocalSearch}(s')$ 
     $s^* = \text{AcceptanceCriterion}(s^*, s^{*\prime}, \text{history})$ 
  until termination condition met
end

```

Figure 26.1 Pseudocode of Iterated Local Search (ILS). The *history* component indicates a possible dependence of the procedures on the search history.

edges, that is, $d_{ij} = d_{ji}$ for every pair of nodes. In the asymmetric TSP (ATSP) at least for one pair of nodes i, j we have $d_{ij} \neq d_{ji}$. All TSP instances used in this empirical study are from TSPLIB, a benchmark library for the TSP accessible at <http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/-TSPLIB95/TSPLIB.html>. These instances have been used in many other algorithmic studies and partly stem from applications.

26.3 ITERATED LOCAL SEARCH

Iterated Local Search is a simple yet powerful metaheuristic. The essence of ILS [4, 21, 22, 28] is to build a biased random walk in the space of the local optima (with respect to some local search algorithm). This walk is built by iteratively perturbing a locally optimal solution, then applying a local search algorithm to obtain a new locally optimal solution, and finally using an acceptance criterion for deciding from which of these solutions to continue the search. (In principle, any procedure that given a candidate solution computes potentially improved solutions can be used. The central idea behind ILS is the iterative application of the improvement procedure by means of some – possibly rather complex – perturbation mechanism.) The general idea of ILS has been rediscovered by many authors and has lead to many different names for ILS like *iterated descent* [4], *large-step Markov chains* [22], *iterated Lin-Kernighan* [15], *chained local optimisation* [21]. To implement an ILS algorithm, the following four sub-procedures have to be defined: *GenerateInitialSolution* generates a starting point for the search, *LocalSearch* implements a (possibly complex) improvement procedure, *Perturbation* computes a perturbation of a given locally optimal solution, and *AcceptanceCriterion* chooses from which of two candidate solutions the search is continued. Given these procedures, Figure 26.1 shows the general algorithmic outline of ILS.

One of the first detailed descriptions of the ideas underlying ILS is given in [22], although earlier descriptions of the basic ideas underlying the approach exist [5, 6]. Some of the first ILS implementations have shown that the approach is very promising and especially on the TSP, some of the best performing approximation algorithms are based on ILS [2, 3, 16, 21].

It is worth noting that there are other algorithms which show similarities to ILS. In [9] proposals similar to the use of perturbations were put forward under the name *random breakup* [9]. Furthermore, *Basic Variable Neighborhood Search* (Basic VNS) [10, 11] can be seen as an ILS algorithm that always applies a specific type of perturbation to the best solution found so far. In Basic VNS, the perturbation steps are essentially random moves in dynamically changing neighborhoods. Although some ILS and VNS implementations may share common features, they are based on two fundamentally different ideas: While VNS is based on the notion of dynamically changing the neighborhood during search, ILS is based on generating a randomized walk in a subspace defined by the locally optimal solutions of a given problem.

26.3.1 ILS for the symmetric TSP

In the following we discuss the particular implementation choices for the four procedures `GenerateInitialSolution`, `LocalSearch`, `Perturbation`, and `AcceptanceCriterion` for the ILS algorithms we applied to the TSP.

26.3.1.1 Choice of LocalSearch. In principle, any local search algorithm may be applied, but the final performance of ILS will be strongly determined by this particular choice. We consider three possible local search algorithms for the TSP: (i) 2-opt, (ii) 3-opt, and the Lin-Kernighan heuristic (LK) [19]. 2-opt and 3-opt try to find an improved tour by exchanging two or at most three edges, respectively. They are much simpler to implement than LK and therefore practitioners might favor 2-opt and 3-opt over LK when a fast implementation is required. However, typically the best performance with respect to solution quality (i.e., length of tours) is obtained using LK [16]. In the following, we will refer to the ILS algorithms using 2-opt, 3-opt, and LK local search as Iterated 2-opt (I2opt), Iterated 3-opt (I3opt), and Iterated LK (ILK), respectively.

In the experiments we used our own 2-opt and 3-opt implementations, while the LK implementation was kindly provided by Martin, see e.g. [21]. (The run-time of this LK implementation increased very strongly when applied to highly clustered TSP instances. Therefore, we did not apply it to such instances.) Our 2-opt and 3-opt implementations use standard speed-up techniques which are described in [7, 16, 22]. In particular, we perform a fixed radius nearest neighbor search within candidate lists of the 40 nearest neighbors for each city and use don't look bits. Initially, all don't look bits are turned off (set to 0). If for a node no improving move can be found, its don't look bit is turned on (set to 1) and the node is not considered as a starting node for finding an improving move in the next iteration. When an edge incident to a node is changed by a move, the node's don't look bit is turned off again. These speed-up techniques together achieve that 2-opt's and 3-opt's run-time grows subquadratically with instance size.

26.3.1.2 Choice of Perturbation. For Perturbation we use the so called *double-bridge* move that is the standard choice for ILS applications to the TSP [16,

21, 22]. It perturbs a given tour s by cutting it at four appropriately chosen edges into four subtours $s_1 - s_2 - s_3 - s_4$ (which occur in this order in s) and reconnecting them in the order $s_4 - s_3 - s_2 - s_1$. This double-bridge move is a specific 4-opt move that cannot be reversed directly by none of the three local search algorithms.

In preliminary experiments we found that best performance is obtained if the edges cut by the double-bridge move are not chosen completely at random. Instead, we proceed as follows: in I2opt and I3opt we first choose an edge (i, j) at random. The other three cut-points are then selected such that they are all within a candidate set of the $\max\{n/2, 500\}$ nearest neighbours of city i . A very similar approach has been shown to be effective on large TSP instances [3, 27]. Alternatively, length restrictions on the newly introduced edges are imposed in the original ILK code of [22]. In our experiments with ILK, a newly introduced edge has to be shorter than 25 times the average edge length of the current solution.

Additionally, the interaction between the double-bridge move and the setting of the don't look bits after Perturbation is important to achieve a good tradeoff between solution quality and run-time [16]. For ILK we follow [16] and reset only the don't look bits of the cities directly affected by the double-bridge move to zero. Yet, for I2opt and I3opt the performance with such a resetting strategy was found to perform not very well with respect to solution quality [28]. A good compromise between run-time and solution quality was found by resetting all the don't look bits of a number of cities (here chosen as 20) preceding and following each single cut-point.

26.3.1.3 Choice of AcceptanceCriterion. Common knowledge in ILS applications to the TSP and to many other combinatorial optimization problems appears to be that the best performance is achieved by accepting better quality solutions only [15, 16, 20, 21]. Consequently, we will use this acceptance criterion as the basis of our analysis; denoted $\text{Better}(s^*, s^{*'})$, it returns $s^{*''}$ if tour $s^{*''}$ is shorter than s^* , and s^* otherwise (see Figure 26.1 for the definition of s^* and $s^{*''}$). If side-moves are allowed, that is, a new solution is also accepted if it has the same length as the current one, we call the acceptance criterion $\text{BetterEqual}(s^*, s^{*''})$. Only occasionally an improved performance has been reported when using acceptance criteria that accept worse solutions with a small probability [12, 22, 27].

26.3.1.4 Choice of GenerateInitialSolution. The initial solutions for all the ILS algorithms studied here are generated by the nearest neighbour heuristic. Compared to random initial tours, this choice reduces the run-time for the first local search application.

26.3.2 ILS for the ATSP

When applying ILS to the ATSP, the ILS algorithm is very much the same as for symmetric TSPs, except that for LocalSearch we apply a particular 3-opt

algorithm which we called **reduced 3-opt**. Note that in 2-opt and some 3-opt moves subtours have to be traversed in the opposite direction, in which case for the ATSP the length of that subtour would have to be re-computed from scratch. To avoid this, reduced 3-opt only applies one specific 3-opt move which does not lead to a reversal of any subtour. For Perturbation, again the double-bridge move can be applied because it does not reverse any subtour. The acceptance criterion and the choice of the initial solution are the same as for the symmetric TSP.

Surprisingly, there are no results for the application of ILS to the ATSP reported in the literature, except for [31], where a specific code optimisation problem is formulated as an ATSP and subsequently solved by transforming the resulting ATSP instances to the symmetric TSP and applying an iterated 3-opt algorithm for symmetric TSPs.

26.4 RTD-BASED EMPIRICAL ANALYSIS OF ILS

In this section we analyse the run-time behaviour of ILS algorithms for the TSP, in particular, of I2opt, I3opt, and ILK for the symmetric TSP and Ired3-opt for the ATSP. Measuring run-time distributions (RTDs) is potentially very helpful for analysing the performance of ILS, because ILS algorithms make heavily use of random choices in the search (such as random solution perturbations, randomised generation of initial solutions, etc.).

26.4.1 Measuring run-time distributions

To empirically measure the RTD for an algorithm applied to a given problem instance, we run the algorithm multiple times on that same problem instance and collect some elementary data: in each run it suffices to record the solution quality (for the TSP: length of the current tour) whenever a new best solution is found and the computation time needed to obtain it. Based on these data it is easy to *a posteriori* estimate the empirical, cumulative probability distribution of the run-time required for reaching different solution quality bounds in the following way: Given a desired solution quality c_b , we determine from the data recorded during the experiments for each single trial j the time $rt(j)$ required to reach a solution of quality better or equal to c_b . By sorting the $rt(j)$ in non-decreasing order, the empirical cumulative RTD can be derived. Formally, if k is the total number of trials, and l is the number of successful trials (i.e., the number of trials in which the desired solution quality c_b was reached), the empirical cumulative RTD can be defined as:

$$\widehat{RT}_b(t) = \begin{cases} j/k & \text{for } rt_{j-1} \leq t < rt_j \text{ and } j \in \{1, \dots, l\} \\ l/k & \text{for } rt_l \leq t \end{cases}$$

where rt_1, rt_2, \dots, rt_l are the sorted run-times $rt(j)$ and $rt_0 = 0$. It should be noted that the empirical RTD reaches one if and only if all trials were successful.

Using this procedure, we can determine *a posteriori* empirical RTDs for several bounds c_b . For the TSP instances used here we are in the fortunate

position that optimal solutions are known; in this case it is best to fix the solution quality bound c_b relative to the optimal solution quality, that is, to require the algorithm to get within a certain percentage deviation from the optimum. Of course, it is also possible to apply the RTD-based methodology to problems for which no optimal solutions are available. In that case, one would look at solution quality bounds defined as a percentage excess over (tight) lower bounds, over best-known solutions, or the best solution found in a number of trials.

Ideally, we would like to run the algorithm for very large CPU times to observe limiting behaviour, where an additional improvement in solution quality by allowing still considerably longer computation times becomes negligible. (Obviously, one can stop a trial if a known optimum is found.) In practice, we have to limit the computation time of the single trials to some reasonable upper bound, because of the possibility that some runs may never reach the desired solution quality. To limit the overall computation time in our RTD analysis we only tested (with one exception) instances with less than 1000 cities, running our algorithms 100 times on each instance, and then determined the RTDs with respect to various bounds on the solution quality. These RTDs are given in Figures 26.2 to 26.6 and are discussed in the next few sections.

26.4.2 Stagnation behaviour and effectivity of restart

Intuitively, a stochastic search algorithm shows stagnation behaviour if for long runs, the probability of finding a solution at least as good as a given bound c_b can be improved by restarting the algorithm at some appropriately chosen cut-off time t_r . Interestingly, it is quite easy to detect such stagnation behaviour from an empirical RTD. Based on a well-known result from probability theory, the probability of finding a solution (of the desired quality) by running a stochastic search algorithm k times for time t is the same as when running the algorithm once for time $k \cdot t$ if and only if this algorithm has an exponential RTD. Hence, to detect stagnation behaviour, we just have to compare the empirical RTD against an exponential distribution. Therefore, in our RTD plots, we also show the cumulative distribution function of an exponential distribution (indicated by $f(x)$ in the plots and defined as $f(x) = 1 - e^{-\lambda x}$, $x \geq 0$) which approximates the empirical RTDs measured with respect to very high quality c_b .

As we have argued, comparing an empirical RTD against an exponential distribution can help to visually as well as analytically detect stagnation behaviour of stochastic search algorithms. Furthermore, if such stagnation behaviour is present, this approach can also be used to determine optimal cutoffs for restarting the algorithm. In fact, a cutoff t_r^* is optimal if the solution probability obtained by restart, given by $P^*(t) = 1 - (1 - \widehat{RT}_b(t))^{(t/t_r^*)}$ is maximal for all t . (For technical reasons, we assume here that t is a multiple of t_r^* — if this does not hold, a slightly more complicated formula has to be used, but the basic result is the same.) Interestingly, using this definition, all the points of $P^*(t)$ lie on a curve that can be graphically determined as the leftmost

exponential distribution that coincides with one of the points $(rt_j, j/k)$ where $j \in \{1, \dots, \min\{l, k - 1\}l\}$. (For technical reasons, if $k = l$, the point $(rt_k, 1)$, can not be considered.) In our RTD plots, stagnation behaviour can easily be visually detected by observing that an empirical RTD falls below this “ideal” exponential RTD.

26.4.3 Operation counts

RTDs can be obtained by measuring directly CPU-time or by using representative operation counts as a more machine independent measure of an algorithm’s performance [1]. A natural choice for a “high-level” operation count for the ILS algorithms applied here is the number of local searches in a given amount of time. To establish the relation between CPU-time and local search iterations, in Table 26.1 we give the average CPU-time measured on a single 266MHz Pentium II CPU and 320MB RAM under Redhat Linux 5.2 (programs are coded in C and were compiled using gcc, version egcs-2.91.66, with the flag `-O2`) when running `I2opt`, `I3opt`, and `Ired3-opt` for 25,000 iterations and `ILK` for 1,000 iterations on the instances used in this study. The actual times are averaged over 5 independent ILS runs on each instance.

For most of our experiments, we report actual CPU-times measured either on a single 167MHz UltraSparc I processor and 192MB RAM (using gcc version 2.7.2 with the same compiler flags,) or the 266MHz Pentium II processor on which the timings in Table 26.1 are based.

I2opt							
eil51	3.07	kroA100	5.39	lin105	5.43	u159	7.07
d198	9.10	lin318	13.25	pcb442	16.59	rat783	27.94
pr1002	34.66	pcb1173	39.42	d1291	41.17	f11577	46.96
I3opt							
d198	68.0	lin318	97.2	pcb442	64.4	rat783	118.5
pr1002	145.2	pcb1173	140.0	d1291	113.9	f11577	124.8
pr2392	180.1	pcb3038	209.7	f13795	208.3		
ILK							
lin318	714.7	pcb442	577.9	att532	433.42		
rat783	163.5	pcb1173	337.8	pr2392	482.44		
reduced 3-opt							
ry48p	27.8	ft70	53.8	kro124p	50.4		
ftv170	58.1	rbg443	285.6				

Table 26.1 CPU-time taken for running `I2opt`, `I3opt`, and `Ired3-opt` on a 266MHz Pentium II CPU for 25,000 iterations.

26.4.4 RTDs for symmetric TSPs

26.4.4.1 Iterated 2-opt. Figure 26.2 shows the empirically observed RTDs for the `I2opt` algorithm on the four TSPLIB instances `d198`, `lin318`, `pcb442`, and `rat783`. Except for instance `d198`, `I2opt` could only very rarely find the

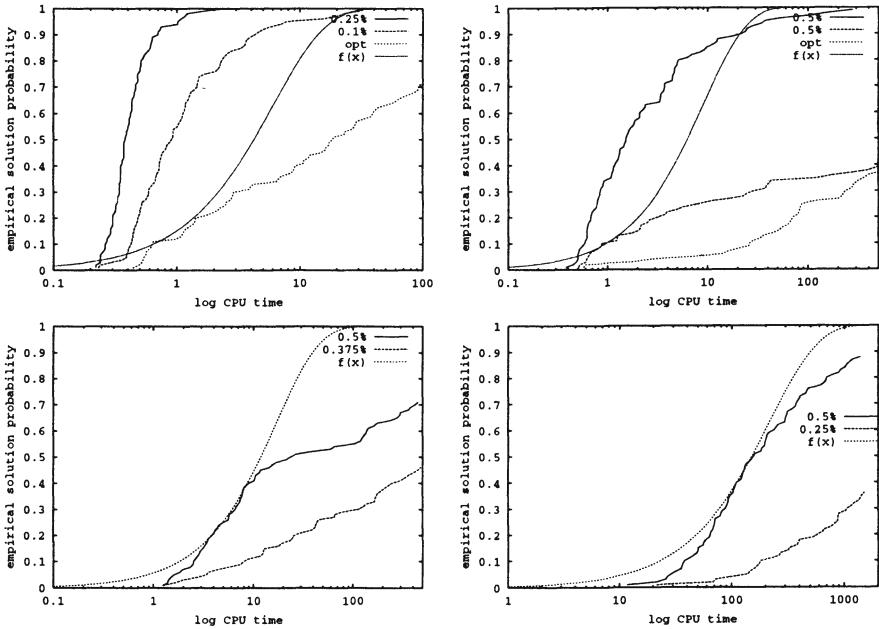


Figure 26.2 Run-time distributions of I2opt on four symmetric TSP instances. RTDs are identified by the desired solution quality (e.g., ‘opt’ means optimal solution is required). Given are the distributions for d198 (upper left side), lin318 (upper right side), pcb442 (lower left side), and rat783 (lower right side). $f(x)$ indicates an exponential distribution. See the text for more details.

optimal solutions for these instances within reasonable run-time. Thus, we report mainly RTDs which are obtained by imposing some weaker bounds on the solutions quality. In the plots, an exponential distribution is fitted to the lower part of the RTDs for high quality solutions (where high quality is here meant as high relative to I2opt). As can be observed in the plots, the exponential matches the lower part of the RTDs well, yet, for larger run-times the RTDs fall strongly below the indicated exponential. Hence, the I2opt algorithm is severely affected by stagnation behaviour and by using restarts, the solution probability can be significantly improved.

26.4.4.2 Iterated 3-opt. Figure 26.3 shows the empirical run-time distributions for the four TSPLIB instances d198, lin318, pcb442, and rat783. Additionally, exponential distributions, which may indicate stagnation behaviour, are fitted to RTDs for high-quality solutions.

The RTDs for high quality solutions indicate that I3opt also suffers from stagnation behaviour. Consider, for example, instance pcb442: After 25 seconds, an optimal solution is found with probability 0.41, but in 33 of the 100 trials the algorithm fails to find an optimal solution even after 1000 seconds.

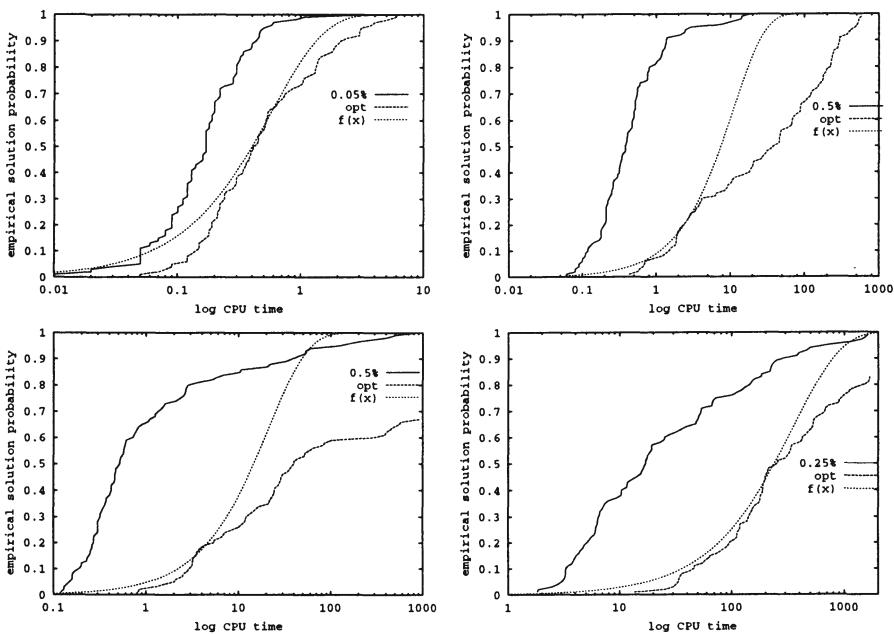


Figure 26.3 Run-time distributions of iterated 3-opt on four symmetric TSP instances. RTDs are identified by the desired solution quality (e.g., ‘opt’ means optimal solution is required). Given are the distributions for d198 (upper left side), lin318 (upper right side), pcb442 (lower left side), and rat783 (lower right side).

A simple calculation shows that if we restart I3opt on this instance after 25 seconds, with 40 restarts (that is, after 1000 seconds) one would estimate to obtain a probability larger than 0.9999 for finding an optimal solution—a much larger probability than obtained by using I3opt without restarts. (As discussed in Section 26.4, the exponential distribution shown in Figure 26.3 gives an indication of how the solution probability would develop when using restarts with an optimal cutoff-time.) Similar observations also apply to the other instances. This stagnation behaviour is especially striking on the instances lin318, pcb442, and rat783. On instance d198, the run-time distribution for finding the optimal solution is much closer to an exponential distribution, although for larger run-times it is slightly less steep. Because I2opt shows strong stagnation behaviour on this latter instance, we may conclude that the more powerful 3-opt local search is responsible for the better behaviour on that instance.

With I3opt we also tested the influence of the initial solution on the run-time distributions for very high quality solutions, in particular, optimal solutions using the same four instances as in Figure 26.3. We found that the influence of the initial solution was not significant. The only exception was instance pcb442, where after a time limit of 1000 seconds in 55 of 100 trials an optimal

solution was found from random starts, while with nearest neighbour starts an optimal solution was found in 67 of 100 trials.

26.4.4.3 ILK. One could object that the observation of stagnation behaviour in the previous two sections may be due to the choice of the 2-opt or 3-opt local search algorithms which show worse performance than LK. To test this hypothesis we have ran ILK on the same instances except the smallest one, which was easily solved.

Because ILK is known to be one of the best performing approximate algorithms for symmetric TSP instances, we expect higher quality solutions to be obtained more frequently than with either 2-opt or 3-opt. This expectation is confirmed by the empirical RTDs given in Figure 26.4. (Instance `lin318` is not shown here, because it was solved to optimality in all runs. On `att532`, `I3opt` shows a similarly strong stagnation behaviour like ILK, yet the maximal probability of finding the optimum was significantly lower.) For instances `pcb442` and `rat783`, the empirical probability of finding an optimal solution is significantly higher than for `I3opt`. Instances `pcb442` and `rat783` are solved in every trial to optimality by the ILK algorithm. The run-length distribution of `pcb442` can be closely fitted by a modified exponential distribution with distribution function $RT(x) = \max\{0, 1 - e^{-\lambda(x-\Delta x)}\}$, where Δx adjusts for the fact that a certain minimal number of iterations have to be performed to obtain a reasonable chance of finding the optimal solution. (The curve in that plot has been fitted with the C. Grammes implementation of the Marquart-Levenberg algorithm available in gnuplot. The fitted parameters are $\Delta x = 51.26$ and $\lambda = 0.00524$.) It appears that this instance is easily solved using the LK heuristic, but it is “relatively hard” to solve when using 2-opt or 3-opt (compare the run-time distribution of ILK to that of iterated 3-opt in Figure 26.3). On instance `rat783`, the RTD for ILK again falls slightly below the exponential distribution, which indicates stagnation behaviour and suggests that the algorithm’s performance could possibly be further improved. On the other two instances `att532` and `pr2392` stagnation behaviour is also clearly visible.

26.4.5 RTDs for asymmetric TSPs

The run-time distributions for the ATSP instances shown in Figure 26.5 indicate findings resembling those for the symmetric instances. Here, the stagnation behaviour of iterated reduced 3-opt appears to be even more severe. Often, the optimal solution is obtained very early in a run or, with a few exceptions, is not found at all within the given computation time. Surprisingly, even on the smallest instance `ry48p` with only 48 nodes, the algorithm shows stagnation behaviour; however, it appears that for very large run-times one might be able to find optimal solutions in all runs.

Interestingly, instance `rbg443`, with 443 cities the largest ATSP instance of TSPLIB, is easily solved with the acceptance criterion $\text{BetterEqual}(s^*, s^{*'})$. In this case an optimal solution is found, on average, in 35 seconds (on an UltraSparc 167MHz processor and using candidate lists of length 20) or 5200

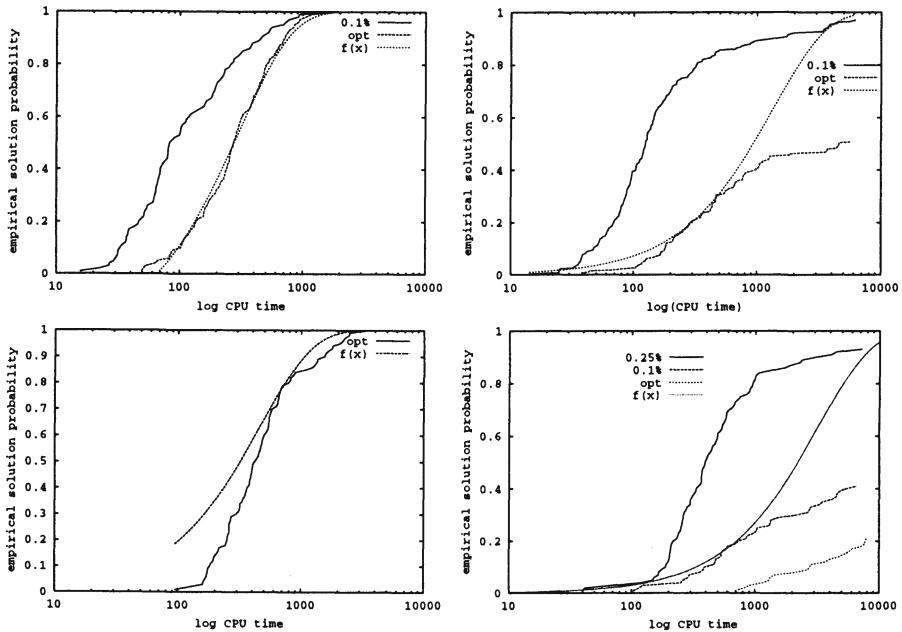


Figure 26.4 Run-time distributions of ILK on four symmetric TSP instances. RTDs are identified by the desired solution quality (e.g., ‘opt’ means optimal solution is required). Given are the distributions for *pcb442* (upper left side), *att532* (upper right side), *rat783* (lower left side), and *pr2392* (lower right side).

local searches, with the maximally required run-time to solve this instance being roughly 150 seconds. The same instance could only be solved in 90% of the runs within 1000 seconds with the acceptance criterion *Better*(s^* , $s^{* \prime}$). In Figure 26.6 we have plotted the run-time distributions for these two acceptance criteria. Such significant differences between these two acceptance criteria could only be observed on the four ATSP instances *rbg323*, *rbg358*, *rbg403*, and *rbg443*. Considering this fact, we used the acceptance criterion *BetterEqual*(s^* , $s^{* \prime}$) for all other ATSP experiments.

26.4.6 Discussion

While it is known that ILS algorithms yield very good quality solutions to the TSP very quickly, the RTD-based analysis revealed unambiguously that for large run-times the standard ILS algorithms suffer from stagnation behaviour which severely compromises their performance if very high quality solutions or optimal solutions are required. In fact, the comparison with an exponential distribution shows that, in the simplest case, the long term behaviour of ILS algorithms can be significantly improved by occasional restarts after a fixed number of iterations (cutoff time). For optimal cutoff-times, the ILS

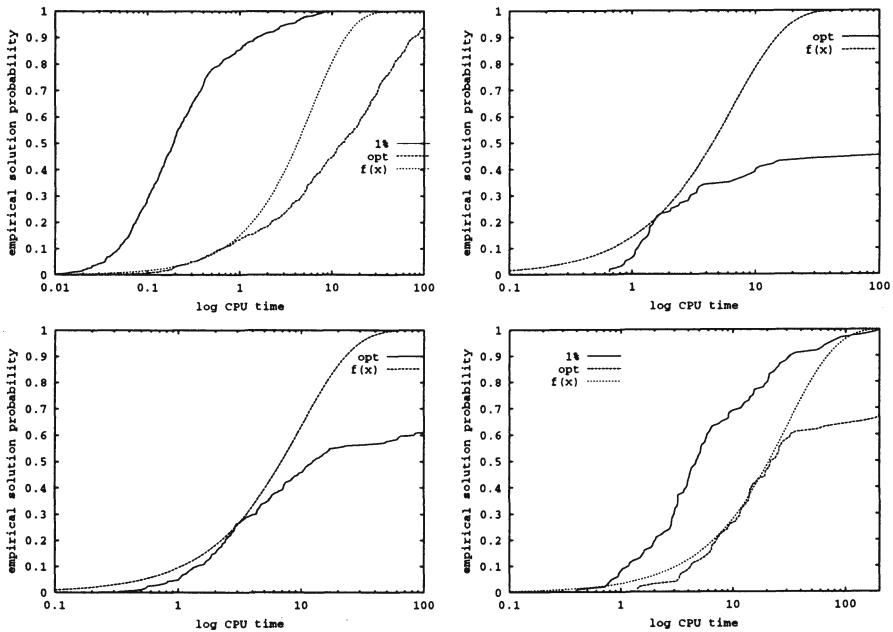


Figure 26.5 Run-time distributions of iterated reduced 3-opt on four ATSP instances. RTDs are identified by the desired solution quality (e.g., ‘opt’ means optimal solution is required). Given are the run-time distributions for ry48p (upper left side), ft70 (upper right side), kro124p (lower left side) and ftv170 (lower right side).

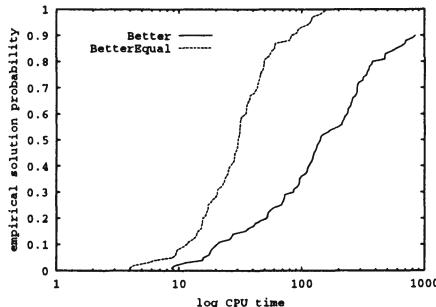


Figure 26.6 Run-time distributions for instance rbg443 using iterated reduced 3-opt. Given are two run-time distribution, one using acceptance criterion $\text{BetterEqual}(s^*, s'')$, the other using $\text{Better}(s^*, s'')$.

algorithm’s solution probability will develop analogously to the indicated exponential distributions. This fact is exemplified in Figure 26.7 for an I3opt algorithm applied to instance att532.

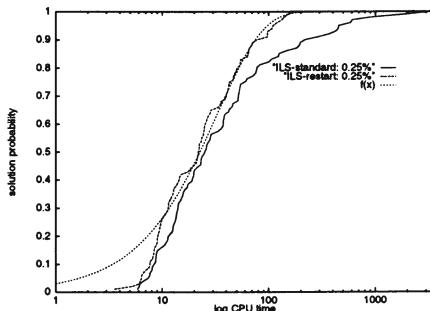


Figure 26.7 Example of improvements by using restart. Given is the run-time distribution for reaching a solution within 0.25% of the optimum on instance att532 for I3opt (indicated by ILS-standard) and an ILS algorithm using restart after an appropriately chosen cutoff time (indicated by ILS-restart). The improved ILS algorithm is able to follow the exponential distribution (indicated by $f(x)$) which approximates ILS-standard's RTD in the left part. See text for more details.

Our RTD-based analysis indicates that for more effective local search algorithms, the problem size at which ILS begins to show strong stagnation behaviour appears to be much higher. For example, the I2opt algorithm showed very strong stagnation behaviour on instance d198 if optimal solutions are required, while I3opt's RTD showed a much better behaviour. Yet, on the larger pcb442 instance both, I2opt and I3opt, were severely affected by the stagnation behaviour, while ILK could solve this instance rather easily. Yet, when applied to even larger TSP instances, ILK showed stagnation behaviour as well. Hence, one possible conclusion from these observations is that the ILS algorithms will—when applied to non-trivial instances—show stagnation behaviour when applied to medium-sized to large instances. The critical instance size at which severe stagnation begins to occur seems to depend on the effectiveness of the local search algorithm.

In summary, the run-time distributions arising from the application of ILS algorithms to the TSP suggest that these algorithms can be further improved. In the following we will present several possibilities how such improvements can be achieved, followed by an empirical investigation of these new ILS variants.

26.5 IMPROVEMENTS OF ILS ALGORITHMS FOR THE TSP

In this section we will present two improved ILS algorithms for the TSP and then give results from an extensive experimental comparison of these algorithms.

26.5.1 Improved ILS algorithms

One conclusion from the stagnation behaviour we observed for ILS is that additional diversification features are needed to increase its performance. There are two particular ways to increase the diversification in ILS algorithms: One possibility is to use acceptance criteria which accept worse solutions or even introduce occasional restarts. A second possibility is to increase the strength of the solution modifications implemented by Perturbation; for the TSP this can be done by cutting the current tour at more than four points than done in the doublebridge move.

We note that yet another way of exploring more distant solutions is used in Basic VNS [11], where the number of edges modified in each perturbation is varied systematically. In general, Basic VNS accepts only improving solutions and varies solely the perturbation size. Worse solutions are accepted by the recently introduced skewed VNS (SVNS) [10]: a solution $s^{* \prime}$ which is worse than the incumbent solution, is accepted if $c(s^{* \prime}) - \alpha \rho(s^*, s^{* \prime}) < c(s^*)$, where $c(s^*)$ is the cost of solution s^* , α is some parameter and ρ a distance measure.

Here, we follow a different approach by proposing and experimentally investigating two improved ILS variants that are based on the idea of an extended acceptance criteria.

26.5.1.1 Soft restarts. As argued before, the simplest strategy to avoid stagnation behaviour is to restart the algorithm from new initial solutions after some predefined cutoff value. Yet, optimal cutoff values may depend strongly on the particular TSP instance. Hence, it appears to be a better idea to make the decision to restart depending on search progress by applying so-called “soft restart criteria”. In particular, we restart the ILS algorithm if no improved solution could be found for i_r iterations, where i_r is a parameter. This *soft restart criterion* assumes that after i_r iterations without improvement the search has become stuck and that additional diversification, here implement as the extreme case of restarting an algorithm from scratch, is needed to escape from there. The restart of the algorithm can easily be modelled by the acceptance criterion $Restart(s^*, s^{* \prime}, history)$, where the *history* component captures, e.g., the very simple use of search history underlying the soft restart criterion. Let i_{last} be the last iteration in which a better solution has been found since the last restart and i be the iteration counter. Then $Restart(s^*, s^{* \prime}, history)$ is defined as

$$Restart(s^*, s^{* \prime}, history) = \begin{cases} s^{* \prime} & \text{if } c(s^{* \prime}) < c(s^*) \\ s' & \text{if } c(s^{* \prime}) \geq c(s^*) \text{ and } i - i_{last} > i_r \\ s^* & \text{otherwise,} \end{cases} \quad (26.1)$$

where s' is a new initial solution.

26.5.1.2 Fitness-distance diversification-based ILS. A disadvantage of restarting ILS from new initial solutions is that previously obtained high

quality solutions are lost. Additionally, ILS algorithms need some initial time t_{init} to reach very high quality solutions (that is to dig deep enough in the search space), which is “wasted” with each restart; furthermore, t_{init} typically increases with instance size. To avoid these disadvantages, it might be better to use a less radical and more directed diversification when the ILS algorithm is deemed to be stuck. Based on this intuition, we propose a new diversification method for ILS that attempts to *find a good quality solution beyond a certain minimal distance from the current search point without using restart*.

We implemented this idea as follows. Let s_c^* be the current candidate solution from which the search should escape, and let $d(s, s')$ be the distance between two tours s and s' measured as the number of edges that differ between s and s' . Then the following steps are repeated until we obtain a solution beyond a minimal distance d_{min} from s_c^* :

- (1) Generate p copies of s_c^* .
- (2) To each of the p solutions, apply Perturbation followed by LocalSearch.
- (3) Choose the best q solutions, $1 \leq q \leq p$, as candidate solutions.
- (4) Let s be the candidate solution with maximal distance to s_c^* .
If $d(s, s_c^*) \leq d_{min}$, then goto (2); otherwise return s .

The goal of step (3) is to obtain good quality solutions, while the goal of step (4) is to choose a candidate solution at a maximal distance from the current solution. The steps (2) to (4) are then iterated until a solution s^* is found for which the requirement in step (4) is satisfied. In fact, to avoid getting stuck in infinite loops we stop this iterative process if we have not found a solution s^* beyond d_{min} after a maximal number of iterations (here 30).

The parameter d_{min} is critical for the performance of ILS-FDD. We estimate d_{min} as the average distance d_{avg} between an number of local optima (100 local optima when applying 2-opt or 3-opt, 50 when applying LK) w.r.t. the local search algorithm applied in the ILS. Then we set alternatingly $d_{min} = 1/4 \cdot d_{avg}$ and $d_{min} = 1/2 \cdot d_{avg}$. A second choice is which solution to take as s_c^* . Here, s_c^* is always taken as the best solution found since the start of the algorithm (only when using LK this is done every second diversification). In fact, these two latter choices are inspired by the results on the fitness landscape analysis of the TSP, where it was shown that (i) local optima cluster together in a small part of the search space (measuring the average distance between local optima), and that (ii) the better the solution quality the closer they are—on average—to a globally optimal solution [8] (choice of s_c^*).

26.5.2 Experimental results

This section presents the results of an experimental comparison of three ILS algorithms, the standard ILS algorithm used for the run-time analysis, the ILS algorithm using soft restarts (referred to as ILS-restart), and the ILS algorithm

with fitness-distance based diversification (referred to as ILS-FDD). In ILS-restart and ILS-FDD, a restart (diversification) is initiated if for i_r , iterations no improved solution is found, where $i_r = 3 \cdot n$ for I2opt, $i_r = n$ for I3opt, and finally $i_r = 1/3n$ for ILK; n is the number of cities of a TSP instance. For ILS-FDD we use $p = 20$, $q = 15$ (parameters were chosen in an ad hoc manner without fine-tuning).

In Tables 26.4 to 26.6 we report computational results for the ILS algorithms based on 2-opt, 3-opt, and LK, respectively, Table 26.7 shows the results for ILS applied to the ATSP; the optimal tour lengths for the instances considered here are shown in Table 26.2. In each table we report the frequency of finding optimal solutions, the average percentage deviation from the optimum and the average run-time to find the best solution in each trial. For instances with $n \leq 1000$, the results are based on 100 independent trials, for $1000 < n \leq 2000$ on 25 trials, and for $n > 2000$ on 10 independent trials for symmetric TSPs when running I2opt or I3opt; when running ILK the results are based on 25 trials for $n \leq 1000$ and 10 trials, otherwise. The maximal CPU-time t_{max} allowed for the algorithms on the single instances when running I2opt and I3opt are indicated in Table 26.3. t_{max} was chosen in roughly such a way that larger instances are given more run-time. When running ILK, we stopped after 5000 LK applications on instances lin318 and pcb442, after 10000 LK applications on att532 and rat783, after 20000 on pcb1173, and after 25000 on pr2392.

eil51	426	kroA100	21282	lin105	14379	u159	42080
d198	15780	lin318	42029	pcb442	50778	att532	27686
rat783	8806	pr1002	259045	pcb1173	56892	d1291	50801
f11577	22249	pr2392	378032	pcb3038	137694	f13795	28772

Table 26.2 Optimal tour lengths for symmetric TSP benchmark instances used in the experiments.

eil51	20	kroA100	60	lin105	60	u159	60
d198	120	lin318	120	pcb442	300	att532	600
rat783	900	pr1002	1200	pcb1173	1200	d1291	1200
f11577	2400	pr2392	3600	pcb3038	7200	f13795	7200

Table 26.3 Time limits for symmetric TSPs when running I2opt or I3opt.

Generally, these experimental results show that our improved ILS algorithms indeed obtain significantly better solution qualities. With both ILS extensions the frequency of finding optimal solutions is strongly increased and the average solution quality improves considerably. The only exception are the results obtained with ILS-restart using 2-opt on the instances rat783 and pr1002, although instance rat783 has shown apparent stagnation behaviour when using

Instance	ILS 2-opt			ILS-Restart			ILS-FDD			
	f_{opt}	Δ_{avg}	t_{avg}	f_{opt}	Δ_{avg}	t_{avg}	f_{opt}	Δ_{avg}	t_{avg}	
eil51	0.20	0.19	< 0.1	1.0	0.0		0.4	1.0	0.0	0.2
kroA100	1.0	0.0	< 0.1	1.0	0.0		< 0.1	1.0	0.0	0.2
lin105	1.0	0.0	< 0.1	1.0	0.0		< 0.1	1.0	0.0	0.2
u159	1.0	0.0	7.3	1.0	0.0		0.4	1.0	0.0	0.3
d198	0.59	0.032	18.5	1.0	0.0		18.8	1.0	0.0	9.4
lin318	0.03	0.30	15.9	0.21	0.11		51.8	0.94	0.008	37.8
pcb442	0.01	0.62	118.6	0.0	0.31		131.2	0.05	0.15	185.2
rat783	0.0	0.49	536.2	0.0	0.60		415.0	0.0	0.29	613.9
pr1002	0.0	0.55	922.7	0.0	0.81		750.2	0.0	0.25	974.6
pcb1173	0.0	1.10	787.3	0.0	0.90		693.5	0.0	0.54	918.3
d1291	0.0	0.68	523.4	0.0	0.27		712.4	0.0	0.12	789.8
f1577	0.0	0.82	996.6	0.0	0.11		1262.0	0.0	0.05	1463.1

Table 26.4 Comparison of I2opt, ILS-restart, and ILS-FDD on symmetric TSPs. For each instance (the number in the instance identifier is the problem size), we report the frequency of finding the known optimal solution (f_{opt}), the average percentage deviation from the optimum, the average CPU-time t_{avg} to find the best solution in a run, and the maximally allowed computation time t_{max} . The algorithms were run on a 266MHz Pentium II CPU.

I2opt. Most probably this result is due to the fact that the parameter i_r was chosen too low. Yet, notice that even for these two instances ILS-FDD significantly improves over standard I2opt. Similarly, when applying I3opt, for rat783 the frequency of finding the optimum is lower with ILS-restart than with I3opt. For the ATSP, all instances are well solved by using ILS-restart (see Table 26.7) and therefore we did not apply ILS-FDD on these instances since apparently they do not provide a real challenge.

The improvement in solution quality achieved by the two enhanced ILS algorithms depends strongly on instance size. In particular, for small problems ILS-restart and ILS-FDD perform similar to standard ILS if the latter does not show any or only very weak stagnation behaviour (notice that I2opt shows very strong stagnation behaviour on instance eil51 which is the smallest instance we tested). Yet, with increasing instance size the performance improvement obtained with ILS-restart and ILS-FDD becomes more significant.

Similarly, the improvement of ILS-FDD over ILS-restart becomes more marked for larger instances. On all instances with more than 400 cities, ILS-FDD shows either better average performance and a higher frequency of finding the optimal solution or, if both algorithms find the optimal solution in all runs, a much lower average run-time. It should also be noted that the performance of ILS-FDD using 3-opt local search is particularly good on instances which are known to be hard for other algorithms. This is the case for f11577 and f13795 which show a pathological clustering of cities. On these instances, ILS-restart also performs surprisingly well; this is probably due to the fact that they contain deep local minima from which the standard ILS algorithm has strong

Instance	ILS 3-opt			ILS-Restart			ILS-FDD		
	f_{opt}	Δ_{avg}	t_{avg}	f_{opt}	Δ_{avg}	t_{avg}	f_{opt}	Δ_{avg}	t_{avg}
d198	1.0	0.0	1.1	1.0	0.0	0.8	1.0	0.0	1.5
lin318	0.65	0.10	13.9	1.0	0.0	7.1	1.0	0.0	13.7
pcb442	0.56	0.12	34.9	1.0	0.0	46.5	1.0	0.0	30.8
att532	0.22	0.055	91.6	0.74	0.0096	214.6	0.96	0.002	202.3
rat783	0.71	0.029	238.8	0.51	0.018	384.9	1.0	0.0	159.5
pr1002	0.56	0.11	389.7	1.0	0.0	578.2	1.0	0.0	207.2
pcb1173	0.0	0.26	461.4	0.0	0.040	680.2	0.56	0.011	652.9
d1291	0.08	0.29	191.2	0.68	0.012	410.8	1.0	0.0	245.4
f1577	0.12	0.52	494.6	0.92	0.00008	477.6	1.0	0.0	294.1
pr2392	0.0	0.23	1538.5	0.0	0.22	2008.5	0.3	0.027	2909.1
pcb3038	0.0	0.22	3687.6	0.0	0.20	4824.3	0.0	0.099	5535.9
f13795	0.2	0.36	3601.9	0.2	0.0035	3080.9	0.9	0.0003	3506.7

Table 26.5 Comparison of I3opt, ILS-restart, and ILS-FDD on symmetric TSPs. For each instance (the number in the instance identifier is the problem size), we report the frequency of finding the known optimal solution (f_{opt}), the average percentage deviation from the optimum, the average CPU-time t_{avg} to find the best solution in a run, and the maximally allowed computation time t_{max} . The algorithms were run on a 266MHz Pentium II CPU.

Instance	ILK			ILS-Restart			ILS-FDD		
	f_{opt}	Δ_{avg}	i_{avg}	f_{opt}	Δ_{avg}	i_{avg}	f_{opt}	Δ_{avg}	i_{avg}
lin318	1.0	0.0	299.8	1.0	0.0	165.8	1.0	0.0	300.5
pcb442	1.0	0.0	325.1	1.0	0.0	289.8	1.0	0.0	297.3
att532	0.68	0.029	1834.6	1.0	0.0	3767.5	1.0	0.0	2194.1
rat783	0.96	0.01	2087.9	1.0	0.0	3577.2	1.0	0.0	2596.5
pcb1173	0.1	0.0065	2071.3	0.2	0.0018	8403.0	0.8	0.0004	5629.9
pr2392	0.1	0.11	10924.3	0.0	0.049	9843.8	0.4	0.014	13028.4

Table 26.6 Comparison of ILK, ILS-restart, and ILS-FDD on symmetric TSPs. For each instance, we report the frequency of finding the known optimal solution (f_{opt}), the average percentage deviation from the optimum, the average number of LK applications i_{avg} to find the best solution in a run, and the maximally allowed number of LK applications.

difficulties to escape. Also note that on these instances it appears preferable to run 3-opt, because the run-time of LK is very high on strongly clustered instances (see also the discussion in [16]).

26.5.3 Related work

Recently, several high performing new approaches and improved implementations of known approaches have been presented for the TSP. Among these algorithms we find the genetic local search approach of Merz and Freisleben [23, 24], a new genetic local search approach using a repair-based crossover operator and brood selection by Walters [30], a specialized local search

Instance	opt	ILS 3-opt			ILS-Restart			t_{max}
		f_{opt}	Δ_{avg}	t_{avg}	f_{opt}	Δ_{avg}	t_{avg}	
ry48p	14422	0.95	0.03	24.7	1.0	0.0	9.2	120
ft70	38673	0.56	0.056	2.2	1.0	0.0	7.4	300
kro124p	36230	0.68	0.056	22.8	1.0	0.0	11.3	300
fvt170	2755	0.92	0.10	35.3	1.0	0.0	45.3	300
rbg443	2720	1.0	0.0	31.7	1.0	0.0	36.3	900

Table 26.7 Comparison of Ired3-opt and ILS-restart on ATSPs. For each instance, we report how often the known optimal solution is found in a given number of trials (f_{opt} /no. trials), the average percentage deviation from the optimum, the average CPU-time t_{avg} to find the best solution in a run, and the maximally allowed computation time t_{max} . The algorithms were run on a 167MHz UltraSparc I CPU.

algorithm for the TSP called Iterative Partial Transcription (ITP) by Möbius et.al. [25], and the ILS approach by Katayama and Narisha [17] which uses a newly designed solution modification mechanism inspired by genetic algorithms. The improved ILS algorithms and, in particular, ILS-FDD compare very well to these approaches [29]. ILS-FDD with 3-opt also compares well with the so far best performance results obtained by Merz and Freisleben with their genetic local search approach. In [24] they report average times for finding optimal solutions on instances lin318, pcb442, att532, rat783 in 25, 35, 131, and 61 seconds, respectively on a 300MHz Pentium II CPU running Solaris (averages are taken over 30 runs). We obtained the same solution quality (except att532 on which 4 of 100 runs did not yield the optimum) at slightly larger run-times on the two larger instances. Yet, the very good performance of our ILK-FDD variant suggests that ILS-FDD, when combined with a much faster LK implementation like Johnson's, Merz's or the code by Applegate, Cook, Bixby and Chvatal, (the Concorde code is publically available via <http://www.keck.caam.rice.edu/concorde.html>) can achieve still a much improved performance over the 3-opt version. Also, note that only the ITP algorithm appears to come close to the results with ILS-FDD on the two instances f11577 and f13795. Overall, our computational results show that ILS and, in particular, the proposed ILS extensions are highly competitive with more complex algorithms for the TSP.

26.6 CONCLUSIONS

In this paper, we presented a novel method for the systematic empirical analysis of metaheuristics based on run-time distributions. While this methodology has been previously applied to combinatorial decision problems, such as SAT [14], here, we use it for investigating the behaviour of Iterated Local Search algorithms for the well-known Traveling Salesman Problem.

Our RTD-based analysis revealed that standard ILS implementations which accept only shorter tours often show stagnation behaviour for the TSP. This

stagnation behaviour can be noted in the fact that from some point on in the search process, relatively little progress is made with respect to finding higher quality solutions. This phenomenon is observed not only for relatively simple ILS algorithms based on 2-opt and 3-opt local search, but also for the ILK algorithm, currently one of the best performing approximation algorithms for large symmetric TSP instances, as well as for ILS algorithms for the asymmetric TSP. Based on this observation, we proposed improved variants of ILS algorithms for the TSP. In an experimental analysis we verified that our strategies to overcome stagnation behaviour are effective and that the modified algorithms show a significantly improved performance.

These results demonstrate that the RTD-based approach provides a good basis for analysing and improving the performance of ILS algorithms in particular, and other meta-heuristics in general. We expect that by systematically applying this methodology to other algorithms and problem domains, further new insights into the behaviour of meta-heuristics for hard combinatorial problems can be obtained, which will facilitate the development of new, improved algorithms and their successful application.

Acknowledgments: We would like to thank Olivier Martin for making available his ILK implementation and for helpful comments on this work. This work was in part supported by a Marie Curie Fellowship awarded to Thomas Stützle (CEC-TMR Contract No. ERB4001GT973400) and a Postdoctoral Fellowship awarded by the University of British Columbia to Holger H. Hoos.

References

- [1] R.K. Ahuja and J.B. Orlin. Use of Representative Operation Counts in Computational Testing of Algorithms. *INFORMS Journal on Computing*, 8:318–330, 1996.
- [2] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. *Finding Tours in the TSP*. Preliminary version of a book chapter available via www.keck.caam.rice.edu/concorde.html, 2000.
- [3] D. Applegate, W. Cook, and A. Rohe. Chained Lin-Kernighan for Large Traveling Salesman Problems. Technical Report No. 99887, Forschungsinstitut für Diskrete Mathematik, University of Bonn, 1999.
- [4] E.B. Baum. Iterated Descent: A Better Algorithm for Local Search in Combinatorial Optimization Problems. Unpublished Manuscript, 1986.
- [5] E.B. Baum. Towards Practical ‘Neural’ Computation for Combinatorial Optimization Problems. In: *Neural Networks for Computing*, J.S. Denker, editor, pages 53–58, American Institute of Physics, 1986.
- [6] J. Baxter. Local Optima Avoidance in Depot Location. *Journal of the Operational Research Society*, 32:815–819, 1981.

- [7] J.L. Bentley. Fast Algorithms for Geometric Traveling Salesman Problems. *ORSA Journal on Computing*, 4:387–411, 1992.
- [8] K.D. Boese. *Models for Iterative Global Optimization*. PhD Thesis, University of California at Los Angeles, Computer Science Department, 1996.
- [9] F. Glover. Future Paths for Integer Programming and Links to Artificial Intelligence. *Computers and Operations Research*, 13:533–549, 1986.
- [10] P. Hansen and N. Mladenović. Variable Neighborhood Search: Principles and Applications. To appear in: *European Journal of Operational Research*.
- [11] P. Hansen and N. Mladenović. An Introduction to Variable Neighborhood Search. In: *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, S. Voss, S. Martello, I.H. Osman, and C. Roucairol, editors, pages 433–458. Kluwer, 1999.
- [12] I. Hong, A.B. Kahng, and B.R. Moon. Improved Large-Step Markov Chain Variants for the Symmetric TSP. *Journal of Heuristics*, 3:63–81, 1997.
- [13] H.H. Hoos and T. Stützle. Evaluating Las Vegas Algorithms — Pitfalls and Remedies. In: *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 238–245, Madison, Morgan Kaufmann, 1998.
- [14] H.H. Hoos and T. Stützle. Towards a Characterisation of the Behaviour of Stochastic Local Search Algorithms for SAT. *Artificial Intelligence*, 112:213–232, 1999.
- [15] D.S. Johnson. Local Optimization and the Travelling Salesman Problem. *Lecture Notes in Computer Science*, 443:446–461, 1990.
- [16] D.S. Johnson and L.A. McGeoch. The Travelling Salesman Problem: A Case Study in Local Optimization. In: *Local Search in Combinatorial Optimization*, E.H.L. Aarts and J.K. Lenstra, editors, pages 215–310, Wiley, 1997.
- [17] K. Katayama and H. Narihisa. Iterated Local Search Approach using Genetic Transformation to the Traveling Salesman Problem. In: *Proceedings of the 1999 Genetic and Evolutionary Computation Conference*, vol. 1, pages 321–328, Orlando, Morgan Kaufmann, 1999.
- [18] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys (editors). *The Travelling Salesman Problem*. Wiley, 1985.
- [19] S. Lin and B.W. Kernighan. An Effective Heuristic Algorithm for the Travelling Salesman Problem. *Operations Research*, 21:498–516, 1973.
- [20] O. Martin and S.W. Otto. Partitioning of Unstructured Meshes for Load Balancing. *Concurrency: Practice and Experience*, 7:303–314, 1995.

- [21] O. Martin and S.W. Otto. Combining Simulated Annealing with Local Search Heuristics. *Annals of Operations Research*, 63:57–75, 1996.
- [22] O. Martin, S.W. Otto, and E.W. Felten. Large-Step Markov Chains for the Traveling Salesman Problem. *Complex Systems*, 5:299–326, 1991.
- [23] P. Merz and B. Freisleben. Genetic Local Search for the TSP: New Results. In: *Proceedings of ICEC'97*, pages 159–164, IEEE Press, 1997.
- [24] P. Merz and B. Freisleben. Fitness Landscapes and Memetic Algorithm Design. In: *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glover, editors, pages 245–260, McGraw-Hill, 1999.
- [25] A. Möbius, B. Freisleben, P. Merz, and M. Schreiber. Combinatorial Optimization by Iterative Partial Transcription. *Physical Review E*, 59:4667–4674, 1999.
- [26] G. Reinelt. *The Traveling Salesman: Computational Solutions for TSP Applications*. Springer-Verlag, 1994.
- [27] A. Rohe. *Parallele Heuristiken für sehr große Traveling Salesman Probleme*. Master's Thesis, Fachbereich Mathematik, Universität Bonn, 1997.
- [28] T. Stützle. *Local Search Algorithms for Combinatorial Problems — Analysis, Improvements, and New Applications*. Ph.D. Thesis, FB Informatik, TU Darmstadt, 1998.
- [29] T. Stützle, A. Grün, S. Linke, and M. Rüttger. A Comparison of Nature Inspired Heuristics on the Traveling Salesman Problem. *Lecture Notes in Computer Science*, 1917:661–670, 2000.
- [30] T. Walters. Repair and Brood Selection in the Traveling Salesman Problem. *Lecture Notes in Computer Science*, 1498:813–822, 1998.
- [31] C. Young, D.S. Johnson, D.R. Karger, and M.D. Smith. Near-Optimal Intraprocedural Branch Alignment. In: *Proceedings of the 1997 Symposium on Programming Languages, Design, and Implementation*, pages 183–193, 1997.

27 POPMUSIC — PARTIAL OPTIMIZATION METAHEURISTIC UNDER SPECIAL INTENSIFICATION CONDITIONS

Éric D. Taillard¹ and Stefan Voss²

¹University of Applied Sciences of Western Switzerland
EIVD campus at Yverdon
Route de Cheseaux 1
CH-1400 Yverdon-les-Bains, Switzerland
Eric.Taillard@eivd.ch

²Technische Universität Braunschweig
Institut für Wirtschaftswissenschaften, Informationsmanagement
Abt-Jerusalem-Straße 7,
D - 38106 Braunschweig, Germany
stefan.voss@tu-bs.de

Abstract: This article introduces POPMUSIC, a meta-heuristic that has been successfully applied to various combinatorial optimization problems. This meta-heuristic is especially useful for designing heuristic methods for large combinatorial problems that can be partially optimized. The basic idea is to optimize sub-parts of solutions until a local optimum is reached. Implementations of the technique to large centroid clustering and to the problem of balancing mechanical parts are shown to be very efficient.

27.1 INTRODUCTION

When presented with large combinatorial optimization problems to be solved, a natural reflex is to decompose these large problems into independent sub-problems that are solved with an appropriate procedure. In this way, large problems can be efficiently tackled since the complexity of the global method grows slowly, typically in $O(n)$ or $O(n \cdot \log(n))$, where n is the problem size.

However, proceeding like this may lead to solutions of moderate quality since the sub-problems might have been created in a somewhat arbitrary fashion. Indeed, it is not easy to find an appropriate way to decompose a problem *a priori*. The basic idea of POPMUSIC is to locally optimize sub-parts of a solution, *a posteriori*, once a solution to the problem is available. These local optimizations are repeated until a local optimum is found. So, POPMUSIC can be seen as a local search working with a special, large neighbourhood. This is why POPMUSIC has been called LOPT (standing for *Local Optimizations*) in the work of [27, 28] and LNS (standing for *Large Neighbourhood Search*) in the work of [23].

POPMUSIC can also be seen as an application of the Proximate Optimality Principle (POP, see e.g. [10]). In a tabu search context, the POP notion is exploited by performing a number of iterations at a given level before restoring the best solution found to initiate the search at the next level. In that context, a level corresponds to the optimization of a sub-problem in the POPMUSIC terminology.

In contrast to other meta-heuristics that have been inspired by a natural process (such as simulated annealing, genetic algorithms or ant systems) which metaphor is translated for designing an optimization technique, POPMUSIC has been synthesized by analyzing principles used in heuristic methods designed for various combinatorial optimization problems. Therefore, other methods than those quoted above are based on POPMUSIC principles. The aim of this paper is to introduce the basic concept of this method and to show that it may successfully be applied to some known combinatorial optimization problems outperforming previous methods from the literature. Furthermore, we show that a number of different methods, presented under various names such as LNS [23], shuffle [2], MIMAUSA [16], VNDS [12] or hybrid tabu search/branch & bound [5] share similar ideas within the same general framework. Therefore, this paper tries to extract the essence of these methods and to synthesize a number of ideas under the same framework that are very close, in a fashion similar to what was done for AMP (*Adaptive Memory Programming*) [28, 29] for meta-heuristics working with a memory.

The paper is organized as follows. In Section 27.2, POPMUSIC principles are presented. Then, Section 27.3 reviews methods that are based on these principles. Applications for vehicle routing problems [26, 19, 20, 23] and clustering [27] are reviewed. Moreover, similar ideas have already been used in other applications, for example in scheduling [2] (*Shuffle* procedure), direct flight network [5] (*Hybrid tabu search/branch & bound* application). Finally, a new application to the balancing of mechanical parts is proposed in Section 27.4 and shown to be efficient.

27.2 POPMUSIC

Local search methods have been used for a long time to improve the quality of a given solution of a combinatorial optimization problem. They consist in defining a subset of *neighbour* solutions for each feasible solution of the

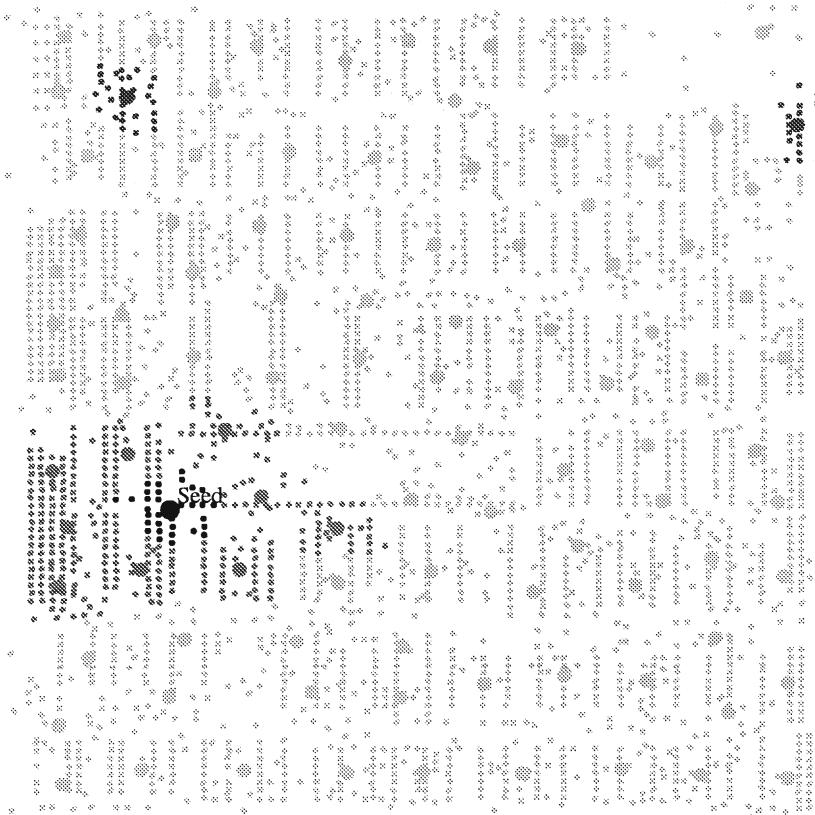


Figure 27.1 Example of two parts that are very loosely related (on the top, in dark grey) and a sub-problem created with 10 parts around a Seed part (in black) for the best solution known to a minimum sum-of-squares problem instance with 3038 entities and 100 clusters.

problem under consideration. Having defined a neighbourhood, local searches attempt to find a good solution, starting from an initial solution (whose quality is not necessarily good) by moving in the solution space from one solution to a neighbour one.

Very often, local search methods use neighbourhoods whose size are polynomial in the problem size. To illustrate, let us suppose that we want to solve a clustering problem consisting in classifying a number n of elements into p classes. The elements have characteristics (e.g. co-ordinates on the Euclidean plane). These characteristics are used to define a (dis-)similarity measure between elements (e.g. the distance on the plane). The objective of a clustering problem is to find classes, or clusters that are homogeneous and well separated (e.g. classes that minimize the sum of all distances between elements belonging to the same class).

A very well known method for clustering problems is the k -means algorithm [13] that consists in moving one element from one class into another class. This neighbourhood is relatively small (in $O(np)$). However, for problem instances of large size (n and p), this neighbourhood contains a majority of solutions that are useless to enumerate. Let us illustrate this by the solution to a Euclidean problem presented in Figure 27.1 (in this figure, small disks are entities and large ones are centres of clusters). Considering the two classes on the top of the figure (in dark grey), it is clear that moving an element into the other class cannot improve the solution since both clusters are homogeneous and well separated.

On one hand, the k -means neighbourhood is too large. On the other hand, it is too small to generate solutions of good quality [4]. The aim of POPMUSIC is precisely to suggest how to generate neighbourhoods that are better adapted to problems of large size. These neighbourhoods are large and are not explicitly enumerated. Generally, they are implicitly enumerated with the help of an optimization procedure, either an exact one or a heuristic one.

For large problems, it is often possible to consider that the solutions are composed of parts, sometimes called chunks [30]. Coming back to our clustering example, it is logical to consider a cluster as a part of a solution. Later, we are going to give other examples of part definitions for other combinatorial optimization problems.

So, let us suppose that a solution S can be represented as a set of parts s_1, \dots, s_p . Moreover, some parts are more in relation with some other parts. In Figure 27.1, the well separated clusters in dark grey at the top are very loosely connected and the clusters in medium grey around the cluster in black at the bottom left of the figure are closely related. So, let us suppose that a relatedness measure can be defined between two parts. The central idea of POPMUSIC is to select a part s_i , called *seed part* and a number $r < p$ of parts s_{i_1}, \dots, s_{i_r} that are mostly related with seed part s_i to form a sub-problem called R_i . For the clustering example, the sub-problem R_i is itself a clustering problem, but smaller than the initial one.

In addition, let us suppose that parts and sub-problems can be defined in such a way that to every improvement of the sub-problem corresponds an improvement of the solution of the whole problem. If all these conditions are met for the combinatorial optimization problem considered, it is possible to state a local search optimization frame that consists in trying to improve all sub-problems that can be defined, until the solution does not contain a sub-problem that can be improved. In the POPMUSIC frame that follows, the set O of parts corresponds precisely to seed parts that have been used to define sub-problems that have been unsuccessfully optimized. Once O contains all the parts of the complete solution, then all sub-problems have been examined without success and the process stops.

Our POPMUSIC frame has one parameter, r , that controls the size of the sub-problems to be optimized and can be sketched as follows:

POPMUSIC(r)

- [1] Input : Solution S composed of parts s_1, \dots, s_p
- [2] Set $O = \emptyset$
- [3] While $O \neq \{s_1, \dots, s_p\}$ repeat
 - (a) Select $s_i \notin O$
 - (b) Create a sub-problem R_i composed of the r parts s_{i_1}, \dots, s_{i_r} most related to s_i
 - (c) Optimize R_i
 - (d) If R_i has been improved, update S (and corresponding parts) and set $O \leftarrow \emptyset$
Else set $O \leftarrow O \cup \{s_i\}$

Basically, the technique is a gradient method that starts from a given initial solution and that stops in a local optimum relative to a large neighbourhood structure. in [23]. Indeed, the neighbourhood structure contains all solutions S' that differ from S only with respect to sub-problem R_i , $i = 1, \dots, p$, i.e., the size of the neighbourhood is determined by the number of solutions to the sub-problems. Naturally, this number may be huge and grows exponentially with parameter r (for $r = p$, $R_i = S$ and the technique just optimizes S directly). It should be noted that optimizing sub-problem R_i may still require the solution of a hard (combinatorial optimization) problem. So, the optimization step cannot be performed by explicit enumeration of the neighbourhood (as in “classical” local search) but either by implicit enumeration methods (such as branch & bound) or by a heuristic method. The optimization procedure may also contain a number of parameters p_1, p_2, \dots . In this case, POPMUSIC has additional parameters, stated as $\text{POPMUSIC}(r, p_1, \dots)$ in the following.

Let us comment on the general POPMUSIC framework step by step. First, the technique can be seen as a “simple” local search. So, it can be embedded in a meta-heuristic that uses a local search as building block. Typical examples are AMP or MIMAUSA. In fact, the method of Rochat and Taillard [20] for vehicle routing problems is precisely an AMP that embeds an optimizer based on POPMUSIC. In MIMAUSA, the sub-problems are optimized using an exact method based on branch & bound and the value of POPMUSIC parameter r is automatically tuned in such a way to optimize the largest sub-problems that the exact method is able to solve in a reasonable computing time.

When designing a method based on POPMUSIC for a given problem, the first choice that has to be made is to define the *parts* of a solution. To illustrate what a part can be, let us take the example of the vehicle routing problem. A solution to this problem is a set of vehicle tours visiting customers. Each customer belongs to one, and only one tour. Therefore, the tours are disjoint and it is quite natural to consider a tour as a part of a solution. This approach was adopted in [26, 19] where the optimizers used are heuristic methods based

on tabu search. In [23], the optimizer used is an exact algorithm based on constraint logic programming and a part of a solution can be defined as one customer.

Once the parts of a solution have been defined, POPMUSIC tries to improve the solution by locally improving a sub-problem composed of a *seed part* and few parts that mostly interact with the chosen seed part. The number of parts considered for optimizing the sub-problem so defined is given by r , the parameter of the method.

The mechanism proposed for avoiding to try to improve a sub-problem that has already been tried to be improved without success is to maintain a set O of seed parts that has defined a sub-problem already treated without success. Once all parts of a solution are contained in O , the solution is a local optimum and the process stops. In case a sub-problem R_i has been successfully improved, a number of parts from s_{i_1}, \dots, s_{i_r} have been changed and the solution is perhaps not locally optimized in the neighbourhood of the parts that have been modified. Therefore, O is emptied before continuing the process. This mechanism is especially relevant if a part structure s_1, \dots, s_p does not survive after an update due to the current solution properties (as a result of an improvement).

A meta-heuristic just provides guidelines for designing optimization methods. The art of the designer is to find how to implement the points that are not specified precisely in the meta-heuristic for the specific problem he wants to solve. The main points that are left free by POPMUSIC are the following:

- [1] definition of the *parts* of a solution;
- [2] *selection* procedure of a part not in O ;
- [3] *relatedness* function between parts; and
- [4] subproblem *optimizer*.

In the next section, we review some methods that are using POPMUSIC principles by specifying the choices made for the four free points quoted above. This allows a very synthetic description of the methods.

27.3 SOME METHODS USING POPMUSIC PRINCIPLES

27.3.1 Vehicle routing

The most basic vehicle routing problem (VRP) is to find a set of vehicle tours starting from and coming back to a depot, in such a way that the tours cover all the customers once and only once, with the constraint that the sum of the customer demands on a tour is not higher than the capacity of the vehicles (see [3] for more details about the VRP). Two possibilities for defining parts of a solution have been given in the literature. In [26, 19, 20] a part is a vehicle tour and in [23] a part is a customer. So, a solution can be seen either as a set of tours or as a set of travel from one customer to another. Depending on the

point of view, quite different methods based on POPMUSIC principles can be derived.

In case a vehicle tour defines a part, it is clear that POPMUSIC can be used only for problem instances involving several tours (at least 3, but typically more than 6). In the problem instances considered in [26, 19, 20], each customer has a co-ordinate on the Euclidean plane. The relatedness between two tours has been defined as follows: First, the rectangular (x, y) co-ordinates of the centre of gravity of each tour is computed. Then the rectangular co-ordinates are transformed into polar ones (ρ, ϕ) , the depot being the reference point. The relatedness between two vehicle tours with centre of gravity (ρ_1, ϕ_1) and (ρ_2, ϕ_2) is then defined by the difference in their angles: $(\phi_1 - \phi_2)$ modulo 2π . A sub-problem is created by randomly selecting a seed tour and by considering then an independent VRP containing only the customers of the r closest tours of the seed tour. Since the aim of [26] was to design a parallel method, several independent sub-problems were considered at a time and optimized using a basic tabu search.

In [19], a POPMUSIC based method is embedded in a kind of variable neighbourhood search (VNS, [12]) which is itself used as the intensification mechanism of a tabu search. In this application, a part is a vehicle tour, the customers have co-ordinates on the Euclidean plane and the angle of the centre of gravity of the tours is also used for defining the relatedness between tours. First, the POPMUSIC method is called with parameter $r = 4$. Once all sub-problems have been optimized with a basic tabu search, the POPMUSIC method is called with parameter r increased by 2 units and so on till the sub-problem created contains all the customers of the original problem. At this point, a regular tabu search working on the whole problem runs for a given number of iterations. Then the VNS-POPMUSIC intensification mechanism is called again. In [20], the whole method is embedded in an adaptive memory programme and used as optimizer.

The approach of [23] is quite different from [26, 19, 20] since a part is a customer (or, more precisely, the trips arriving to and departing from a customer). The relatedness between two customers is defined in two ways, depending on the objective function to optimize. If the goal is to minimize the total distance travelled by the vehicles, then the relatedness between two customers is the inverse of the distance as defined in the original problem. If the objective is to minimize the number of vehicles used to service all customers (with total distance minimization as secondary objective), a penalty to the ordinary distance is added in case both customers belong to different tours. In order to add some randomness in the sub-problem created, the computed relatednesses are randomly perturbed.

The choice of the seed customer is always random. Once a subset of customers has been chosen, these customers are removed from the problem and optimally re-inserted with constraint logic programming. The process is also embedded in a VNS frame: At the beginning, parameter r is set to 1. Once α sub-problems have been unsuccessfully solved (where α is a parameter), r is

incremented. Note that none of the methods discussed here uses exactly the stopping criterion described in the POPMUSIC frame. An interesting research issue would be to study methods more closely related to the frame proposed in this article and to try to use other relatedness measures for building sub-problems to optimize.

Another application of POPMUSIC principles to the VRP is provided in [21]. Similarly to the previous approach customers are removed from the problem and re-inserted with constraint logic programming. Embedded in a VNS frame they also allow for an arc exchange neighbourhood. That is, based on their ideas we see that the definition of parts may be changed throughout a POPMUSIC application especially when it is combined with the VNS frame.

27.3.2 Clustering

The basic foundation of POPMUSIC was given by Taillard [27] for centroid clustering problems where the method was introduced under the acronym LOPT, standing for *Local OPTimizations*. Cluster analysis is to partition a set of entities into subsets, or clusters, such that the subsets are homogeneous and separated from one another, considering measurements describing the entities. This problem is very old and appears in a very large number of practical applications (see, e.g. [22]). A class of centroid clustering problems can be stated as follows: Given n entities e_i with weights w_i ($i = 1, \dots, n$) it is searched p centres c_j ($j = 1, \dots, p$) minimizing $\sum_{i=1}^n [\min_j w_i \cdot d(e_i, c_j)]$, where $d(e_i, c_j)$ measures the dissimilarity between e_i and c_j . Different centroid clustering problems are commonly used: If the centres can be placed only on entities, the problem is known as the p -median problem. If the entities are given by co-ordinates on the Euclidean plane, the distance measure being the Euclidean distance and the centres can be placed anywhere on the plane, this is the multi-source Weber problem. If the entities are given by co-ordinates in a Euclidean space, the distance measure being the square of the Euclidean distance and the centres can be placed anywhere in the space, this is the minimum sum-of-squares clustering problem.

For clustering problems, POPMUSIC-based methods have been designed by making the following choices: A part of a solution is a cluster of entities. For centroid clustering, the relatedness between two parts is defined as the inverse of the distance between the centre of the clusters. The selection of a seed part not in O is done at random. Finally, the optimization procedure is a descent method that uses a candidate list strategy and that stops after a given number of iterations I (see [10] for details about candidate lists). This heuristic method has a parameter. Therefore, our POPMUSIC method has two parameters: r , as described in the frame, and I , the parameter of the optimization procedure.

After having seen the POPMUSIC-based method designed by Taillard [27] in 1996, Hansen and Mladenović [12] have proposed variable neighbourhood decomposition search, which may be viewed as a version of POPMUSIC in which parameter r is systematically varied. In Table 27.1, we compare our POPMUSIC heuristic (with two parameter sets) with the reduced variable

<i>p</i>	Best known	Quality [% above best]				CPU time [s. SPARC10]			
		RVNS	VNDS	POP1	POP2	RVNS	VNDS	POP1	POP2
100	47685934.0	2.34	0.73	1.19	0.44	153	1132	145	505
150	30524769.8	3.13	1.44	1.16	0.58	153	1676	111	355
200	21875113.9	2.49	1.10	1.07	0.50	160	2124	96	262
250	16621446.4	2.56	1.34	1.35	0.76	182	2954	89	234
300	13289633.4	2.50	1.57	1.58	0.78	229	3151	82	205
350	11019171.4	2.60	1.36	1.69	0.76	231	3760	75	179
400	9362179.2	3.35	1.82	1.40	0.66	165	3446	72	170
450	8101618.7	3.47	1.71	1.61	0.80	242	4152	69	163
500	7102678.4	2.85	1.86	1.70	0.90	204	4060	68	156

Table 27.1 Comparison of RVNS, VNDS, POPMUSIC(6, 40) (POP1) and POPMUSIC(10, 100) (POP2) for minimum sum-of-squares clustering instances with 3038 entities.

neighbourhood search (RVNS) and variable neighbourhood decomposition search of [12]. Comparisons are restricted to minimum sum-of-squares instances with 3038 unit weight entities located on the Euclidean plane, named pcb3038 in the TSPLIB compiled by Reinelt [18]. Further computational results are provided in [27]. Various centroid clustering problems with non-unit weight instances, larger instances up to 88900 entities, are considered in this reference. The initial solution provided to our method is obtained with DEC [27], a fast heuristic that decomposes the problem into a number of sub-problems. DEC produces solutions from 5 to 8% above best ones known. The computation times provided in Table 27.1 take into account the time to generate the initial solution. Each problem instance was solved 10 times with our method and the values are averaged in the table.

In Table 27.1, it is quite clear that our POPMUSIC implementation is more efficient than RVNS and VNDS. Even for short runs with $r = 6$ and $I = 40$ the method produces solutions of similar quality to VNDS while taking on the average 34 times less computation time. By enlarging the sub-problem size ($r = 10$) and increasing the optimization effort ($I = 100$), it is possible to halve the average gap to the best solutions known. That is, for the second parameter setting (POP2) the solution quality improves considerably while having computation times that are well below those of VNDS in all cases and even below those of RVNS for $p \geq 300$. All best solution values reported in the table have been found with our POPMUSIC implementation.

27.4 POPMUSIC FOR BALANCING MECHANICAL PARTS

27.4.1 Problem definition

The problem of balancing mechanical parts may be sketched as follows: n mechanical parts of weight $w_i (i = 1, \dots, n)$ have to be fixed on a support. Each part may occupy any of n positions on the support, given by their co-

ordinates (x_i, y_i, \dots) ($i = 1, \dots, n$), where typically the problem is considered in one, two or three dimensions. Only one part may be fixed on a given position. The goal is to position each mechanical part on the support in such a way that the centre of gravity of all the parts is as close as possible to an ideal centre of gravity, say $(0, \dots, 0)$. Mathematically, the objective is to find $\pi \in \Pi$, the set of the permutations of n elements, that minimizes the function $f(\pi)$:

$$\min_{\pi \in \Pi} f(\pi) = \frac{1}{\sum_{i=1}^n w_i} \cdot \left\| \left(\sum_{i=1}^n w_i \cdot x_{\pi_i}, \sum_{i=1}^n w_i \cdot y_{\pi_i}, \dots \right) \right\|.$$

Note that it is possible to cover the case that the number of positions differs from the number of parts by adding dummy parts of weight 0 (if the number of positions is greater than the actual number of parts) or dummy positions at the ideal centre of gravity (if the number of parts is greater than the actual number of positions).

The balancing problem is of major practical importance. Typical applications are the balancing of rotating parts [11, 7, 8] such as turbine rotors. A turbine rotor is composed of an axis around which a number of (almost) identical blades are fixed. Due to mechanical, thermic and chemical constraints, the blades are made in a special alloy that is difficult to manufacture. The result is that the blades have not exactly the same weight. The weight difference of the blades induces vibrations in the turbine that strongly depend on the position of the centre of mass of the blades. The more the centre of mass is away from the centre of the axis of the turbine, the higher the vibrations are. Therefore, it is very important to find an arrangement of the blades around the axis in such a way that the centre of gravity is as close to the rotation centre of the rotor as possible.

Another practical application of the problem is the balancing of the load of boats, airplanes or lorries.

In the literature [14, 24, 15, 17], the balancing problem is commonly explored as a special case of the quadratic assignment problem (QAP), which is NP-hard. This, however, does not settle its complexity, since transforming a problem \mathcal{P} into a hard one does not establish the complexity of \mathcal{P} .

The balancing of mechanical parts can be shown to be NP-hard by the polynomial transformation of the bi-partition problem. The latter consists in finding a partition of $2n$ elements of weight w_i , ($i = 1, \dots, 2n$) into two subsets S_1 and S_2 of n elements each, in such a way that $\sum_{i \in S_1} w_i = \sum_{i \in S_2} w_i$ [9]. Let us now consider a balancing problem in one dimension with $2n$ parts with the same weight as the weight of the partition problem and a support with n positions at co-ordinate (-1) and n positions at co-ordinate (1). There is a solution of cost 0 to this balancing problem if, and only if a bi-partition exists.

The approach of modeling the balancing problem as a special case of the quadratic assignment problem may have some additional shortcomings. Although a large number of efficient tabu search approaches have been developed for the problem, the computation of the objective function takes a time proportional to n^2 with the quadratic assignment formulation while it can be computed in $O(n)$ with the original formulation.

27.4.2 POPMUSIC adaptation

The problem of balancing mechanical parts has been successfully approached with tabu search-based methods working on the quadratic assignment formulation [24]. So, we decided to adapt a basic tabu search due to Taillard [25] to the original formulation of the problem and to use this method as the optimizer of a POPMUSIC application. The neighbourhood used in this tabu search method is to swap the positions of two parts. Naturally, swapping two parts of equal weight is forbidden since such a move does not change the position of the centre of gravity. It is quite natural to consider a part of a solution as a mechanical part. The parts are initially ranked by decreasing weights. So, let us suppose that the i^{th} heaviest part has weight w_i . With such a numbering of the parts, we define the set of the r closest parts of i as parts $i, i + 1, \dots, i + r - 1$ (the additions are made modulo n , so, the lightest part is close to the heaviest one). The rationale behind this definition of sub-problems is the following: The aim of the optimization of a sub-problem is to modify the position of r parts in such a way to move the centre of gravity in the direction of its ideal position. It is observed, for problem instances commonly used in the literature, that the swap of two parts often considerably modifies the centre of gravity, meaning also a considerable variation in the objective function evaluation. Since the sub-problems created in our POPMUSIC implementation are composed of parts having similar weights, it is possible to obtain small moves of the centre of gravity. Note that few sub-problems are composed of the lightest and heaviest parts. So, large moves of the centre of gravity are also possible. The seed parts are simply chosen in the order $1, 2, \dots, n, 1, \dots$.

The initial solution provided to the method is just a random one. After the solution is optimized with POPMUSIC, a short tabu search (500 iterations) working on the whole neighbourhood tries to improve it.

Another way to create sub-problems is to consider a part of a solution as a position instead of a mechanical part. However, we decided not to follow this idea for the following reasons: First, if the problem instance contains a cluster of r (or more) positions well separated from the other ones, the mechanical parts initially placed on this cluster cannot be moved on any other position. Second, if the weights of the mechanical parts are very different, the ruggedness of the neighbourhood used for optimizing sub-problems is higher and the sub-problems are therefore more difficult to optimize with a tabu search. Third, a sub-problem composed of positions far from the centre of gravity can move the global centre of gravity in a subset of directions poorer than a sub-problem composed of positions around the centre of gravity.

27.4.3 Numerical results

In order to show the efficiency of our POPMUSIC implementation for the balancing of mechanical part, we have considered a set of 18 problem instances of the literature with size ranging from 30 to 100 parts. These instances correspond to the balancing of turbine rotors where the parts are blades to be

regularly arranged around the rotor axis. The first set of problem instances, linear30 . . . , linear80 are artificial instances with blades of weight 1, 2, 3, . . . , n . The second set is composed of two real problems obtained from an airline company, named mas38 and mas40, used by Mason and Rönnqvist [15]. The last set is composed of ten 100-parts instances randomly generated with the same mean and variances of blade weights as observed in airplane turbines, also originating from [15].

Unfortunately, it is not possible to use the numerical results published in the literature for making significant comparisons with our POPMUSIC heuristic. First of all, it would be unfair to compare our implementation (with $O(n)$ objective function evaluation) with QAP-based methods using an evaluation in $O(n^2)$. In our tabu search implementation, the complexity of one iteration is $O(n^2)$ and performing I iteration takes a time in $O(I \cdot n^2)$. In contrast, the reverse elimination and star shape diversification methods implemented in [24] take a time in $O(n^3 + I^2n^2)$ to perform I iterations.

Besides the difficulty to compare methods with different time complexity, there is another major difficulty : The landscape associated with the neighbourhood that swaps two mechanical parts is extremely rugged (see [1] for details about landscapes and ruggedness). So, a huge difference between a good and a bad run can be observed. For example, looking at 300 tabu search executions for problem instances linear30, . . . , linear80, the worst solution is typically 250-300% above the average one, meaning that a single bad run may influence a lot the average objective function statistic usually used in the literature. This implies that a number of numerical results published are meaningless. So, significant comparisons cannot be done.

In order to get the numerical results given in Table 27.2, we have proceeded as follows. First, we run a basic tabu search for 10000 iterations and looked at the computing time (expressed in seconds on Sun Sparcstation 5 in the Table) and we try to find parameters for our POPMUSIC heuristic in such a way that its computational time is similar. We found that choosing $r = 22$ parts in the sub-problems and performing about 1070 tabu search iterations for optimizing the sub-problems was convenient. Then, we run both POPMUSIC and tabu search 300 times with the chosen stopping criterion. In Table 27.2, we provide first the name of the instance, then the number of blades, then the absolute average solution value obtained with POPMUSIC (multiplied by 10^{10} to simplify the number listed in the table), then the average solution value of tabu search relatively to POPMUSIC ones, then the computational time of both methods and finally a confidence measure of the superiority of POPMUSIC, as explained below. We see that POPMUSIC produces better solutions than tabu search for problem instances with at least 40 mechanical parts. For linear80, the average solutions produced by tabu search are nearly 50% above those produced by POPMUSIC. For the smallest instances linear30 and mas38, POPMUSIC average solution value is slightly worse than basic tabu search ones.

In order to see if there is a statistical difference in the solutions produced by both methods, we run a Mann-Whitney test (for unparametric statistics, see, e.g. [6]). The latter can be used to test if POPMUSIC has a probability higher than 1/2 to produce a solution better than tabu search. Moreover, the degree of confidence of the assumption that POPMUSIC has a probability higher than 1/2 to produce a better solution than tabu search can be evaluated (a confidence near to 0% indicates that tabu search is significantly better). Note that the distribution functions of solution values for both methods differ not only in the location of the distribution. So, the Mann-Whitney test cannot be used for comparing the average solution values. For three problem instances (linear30, mas38 and mas40) we found that 300 runs of each method were not enough for finding a significant difference between both methods (even if the relative average solution values may differ from more than 12%). So, we run both methods 10000 times for these 3 instances. Then, we can have a significant confidence degree, but unusual conclusions : For Linear30, POPMUSIC produces solutions of worse average quality than the basic tabu search, but the probability for a POPMUSIC run to produce a solution better than a tabu search run is significantly above 1/2. This just translates the fact that both distribution functions differ, but not necessarily their averages. For mas38, POPMUSIC produces solutions of worse average quality and has a probability significantly below 1/2 to produce a solution better than tabu search. For mas40, POPMUSIC is better considering both criteria.

Name	<i>n</i>	Quality		CPU [s. SPARC5]		Confidence (%) $P(POP < TS) > 1/2$
		POP	TS/POP	POP	TS	
linear30	30	57721	0.991	3.26	4.06	98.2
linear40	40	20613	1.169	6.92	6.75	99.996
linear50	50	9336	1.300	10.21	10.26	99.999
linear60	60	5080	1.437	14.11	15.13	99.999
linear70	70	3285	1.419	18.44	20.53	99.999
linear80	80	2109	1.494	21.88	26.71	99.999
mas38	38	224	0.878	4.67	5.92	0.000
mas40	40	1773	1.037	6.24	6.30	99.999
mas100-1	100	74	1.201	36.29	43.02	99.999
mas100-2	100	88	1.234	44.08	42.85	99.999
mas100-3	100	71	1.096	51.08	42.12	97
mas100-4	100	92	1.280	39.04	42.03	99.999
mas100-5	100	77	1.239	42.86	43.01	99.999
mas100-6	100	101	1.145	35.50	42.81	99.98
mas100-7	100	95	1.300	38.47	42.74	99.999
mas100-8	100	110	1.305	38.91	42.50	99.999
mas100-9	100	103	1.324	42.78	42.16	99.999
mas100-10	100	91	1.149	37.70	43.51	98.3

Table 27.2 Comparison of POPMUSIC(22, 1070) and Tabu(10000) for turbine rotor balancing instances.

27.5 CONCLUSIONS

This article presents POPMUSIC, a meta-heuristic especially designed for optimizing the solutions of large instances of combinatorial problems. The meta-heuristic suggests how to create sub-problems from a given solution in order to optimize them. In addition, POPMUSIC includes a stopping criterion similar to those used in improvement methods. In essence, POPMUSIC works as an improvement procedure defined on a very large neighbourhood. The size of this neighbourhood can be modulated through a parameter. Therefore, POPMUSIC-based methods can be easily embedded in a variable neighbourhood frame, as done for the vehicle routing problem (before the introduction of the VNS terminology).

POPMUSIC formalizes and presents under a simple frame a number of ideas “in the air” and already used in various implementations, in particular for vehicle routing problems. But other problems have been approached similarly, such as the job-shop scheduling [2] or the direct flight network problem [5].

Two methods based on POPMUSIC are reviewed in detail: First an application to centroid clustering and second an application to the problem of balancing mechanical parts. It is shown that the POPMUSIC application to the sum-of-squares clustering problem is much more efficient than other VNS-based methods, which are themselves much more efficient than methods frequently used in statistical software.

Second, the problem of balancing mechanical parts is introduced and shown to be NP-hard. A fast tabu search has been designed for this problem. This tabu search is embedded in a POPMUSIC frame and used as sub-problem optimizer. A careful statistical analysis discloses that the POPMUSIC-based method is more efficient than the fast tabu search for large instances of this problem.

Finally, let us speculate on other combinatorial optimization problems that could be tackled with a POPMUSIC-based method. A way to approach problems on a tree (Steiner tree, capacitated minimum spanning tree, ...) would be to define parts as sub-trees. Then, the relatedness between two parts may depend on the distance between two nodes belonging to different parts (either the minimum direct distance between any couple of nodes or the distance along the complete tree). The Steiner nodes most related to the considered parts and that are not used in the current solution should be added for creating a sub-problem. Another approach, more atomic, would be to consider any single node as a part and the relatedness between parts could just depend on the distance between nodes.

For graph colouring problems, a part could be a colour. A more atomic approach could be to consider one element to colour as a part. In the first case, the sub-problems are independent graph colouring problems with a limited number of colours. In the second case, the sub-problems have a limited number of elements to colour that are in relation with the other elements of the complete graph. So, the sub-problems would be colouring problems with

additional constraints. Both approaches may be transposed for frequency allocation problems.

For scheduling problems, a classical approach is to consider a machine as a part. Sub-problems are then scheduling problems with a limited number of machines but additional constraints to take into account the schedule of the operations on the other machines. Another possibility that could be interesting to investigate is to consider a job as a part.

Finally, POPMUSIC could also be used for on-line optimization problems where a new request has to be added to a solution computed with requests previously received. Indeed, a sub-problem can be built with the parts of the current solution that are most related to the new request. The sub-problem is then optimized on-line. Then, a POPMUSIC process can optimize the current solution off-line until the next request arrives.

References

- [1] E. Angel and V. Zissimopoulos. On the Landscape Ruggedness of the Quadratic Assignment Problem. Presented at the *International Workshop on Combinatorics and Computer Science LIX-CNRS*, Palaiseau, 1997.
- [2] D. Applegate and W. Cook. A Computational Study of the Job-Shop Scheduling Problem. *ORSA Journal on Computing*, 3:149–156, 1991.
- [3] M. Ball, T. Magnanti, C. Monma, and G. Nemhauser (editors). *Network Routing*. Elsevier, 1995.
- [4] J. Brimberg, P. Hansen, N. Mladenović, and É. D. Taillard. Improvements and Comparison of Heuristics for Solving the Multisource Weber Problem. *Operations Research*, 48:444–460, 2000.
- [5] K. Büdenbender, T. Grünert, and H.-J. Sebastian. A Hybrid Tabu Search/Branch-and-Bound Algorithm for the Direct Flight Network Design Problem. *Transportation Science*, 34:364–380, 2000.
- [6] W.J. Conover. *Practical Nonparametric Statistics*. Wiley, 1999.
- [7] M.S. Darlow. *Balancing of High-Speed Machinery*. Springer-Verlag, 1989.
- [8] F.F. Ehrich. *Handbook of Rotordynamics*. McGraw-Hill, 1992.
- [9] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, 1979.
- [10] F. Glover and M. Laguna. *Tabu Search*. Kluwer, 1997.
- [11] M.J. Goodwin. *Dynamics of Rotor-bearing Systems*. Unwin Hyman, 1989.

- [12] P. Hansen and N. Mladenović. An Introduction to Variable Neighborhood Search. In: *Meta-heuristics: Advances and Trends in Local Search Paradigms for Optimization*, S. Voss, S. Martello, I.H. Osman and C. Roucairol, editors, pages 422–458, Kluwer, 1999.
- [13] R.C. Jancey. Multidimensional Group Analysis. *Australian Journal on Botany*, 14:127–130, 1966.
- [14] G. Laporte and H. Mercure. Balancing Hydraulic Turbine Runners: A Quadratic Assignment Problem. *European Journal of Operational Research*, 35:378–381, 1988.
- [15] A. Mason and M. Rönnqvist. Solution Methods for the Balancing of Jet Turbines. *Computers and Operations Research*, 24:153–167, 1997.
- [16] T. Mautor and P. Michelon. MIMAUSA: A New Hybrid Method Combining Exact Solution and Local Search. *Extended Abstracts of the 2nd Metaheuristics International Conference*, page 15, Sophia-Antipolis, 1997.
- [17] L.S. Pitsoulis, P.M. Pardalos, and D.W. Hearn. Approximate Solutions to the Turbine Balancing Problem. To appear in: *European Journal of Operational Research*.
- [18] G. Reinelt. TSPLIB: A Traveling Salesman Library. *ORSA Journal on Computing*, 3:376–384, 1991.
- [19] Y. Rochat and F. Semet. A Tabu Search Approach for Delivering Pet Food and Flour in Switzerland. *Journal of the Operational Research Society*, 45:1233–1246, 1994.
- [20] Y. Rochat and É.D. Taillard. Probabilistic Diversification and Intensification in Local Search for Vehicle Routing. *Journal of Heuristics*, 1:147–167, 1995.
- [21] L.-M. Rousseau, M. Gendreau, and G. Pesant. Using Constraint-Based Operators to Solve the Vehicle Routing Problem with Time Windows. To appear in: *Journal of Heuristics*.
- [22] H. Späth. *Cluster Analysis Algorithms for Data Reduction and Classification of Objects*. Ellis Horwood, 1980.
- [23] P. Shaw. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. Technical Report, ILOG S.A., Gentilly, 1998.
- [24] L. Sondergeld and S. Voss. A Star-Shaped Diversification Approach in Tabu Search. In: *Meta-Heuristics: Theory and Applications*, I.H. Osman and J.P. Kelly, editors, pages 489–502, Kluwer, 1996.
- [25] É.D. Taillard. Robust Taboo Search for the Quadratic Assignment Problem. *Parallel Computing*, 17:443–455, 1991.

- [26] É.D. Taillard. Parallel Iterative Search Methods for Vehicle Routing Problems. *Networks*, 23:661–673, 1993.
- [27] É.D. Taillard. Heuristic Methods for Large Centroid Clustering Problems. Technical Report, IDSIA-96-96, IDSIA, Lugano, 1996.
- [28] É.D. Taillard. *Programmation à mémoire adaptative et algorithmes pseudo-gloutons : nouvelles perspectives pour les mét-heuristiques*. Habilitation Thesis, University of Versailles, 1998.
- [29] É.D. Taillard, L.M. Gambardella, M. Gendreau. and J.-Y. Potvin. Adaptive Memory Programming: A Unified View of Meta-Heuristics. To appear in: *European Journal of Operational Research*.
- [30] D.L. Woodruff. Chunking Applied to Reactive Tabu Search. In: *Meta-Heuristics: Theory and Applications*, I.H. Osman and J.P. Kelly, editors, pages 555—569, Kluwer, 1996.

28

SUBCOST-GUIDED SIMULATED ANNEALING

Mike Wright

Department of Management Science
Lancaster University
LA1 4YX, UK
m.wright@lancaster.ac.uk

Abstract: This paper describes a variant of simulated annealing for problems with compound objectives, i.e. those for which a number of diverse objectives are combined into a single objective function. The method takes account not only of the progress of the overall cost but also, to a limited extent, of the progress of the individual subcosts. Various experiments using this approach are described and results given. These results are not conclusive but suggest that there may be considerable value in using this type of approach.

28.1 INTENSIFICATION AND DIVERSIFICATION

Various metaheuristic search methods have proved highly successful at obtaining very good solutions to a wide range of complex combinatorial problems. Crucial to the success of such techniques is the balance between two important facets: *intensification* and *diversification*.

Different techniques achieve this balance in different ways. There may explicitly be two separate phases, one for each facet, as in Tabu Thresholding (TT) (Glover [4]). Other phased methods such as Strategic Oscillation (SO) (Dowsland [2]) achieve the same type of balance in a less explicit way. Tabu Search (TS) (Glover and Laguna [5]) is not always explicitly phased, but the balance may change through time, using short-term and long-term memories.

For other techniques the balance changes in a fairly predetermined way. For example, in Simulated Annealing (SA) (Dowsland [1]), the search normally starts with diversification dominant, but intensification becomes gradually more

influential as the search proceeds, by means of the temperature parameter. This effect is also seen in the Great Deluge (GD) method (Dueck [3]) and various other threshold-based approaches.

In other cases, the balance between the two facets may vary depending on the progress of the search, often by means of sudden changes. This occurs in some implementations of TS or SA (e.g. involving occasional large temperature rises) and in Variable Neighbourhood (VN) methods (Mladenovic and Hansen [7]).

Often intensification is fairly straightforward, but diversification may be more complicated. The aim is to direct the search away from previously visited regions of the solution space towards potentially fruitful new areas. The aspiration to go somewhere new can be achieved by means such as a long-term memory in TS, or by a purely (or partly) random process, as in SA. But how can we try to influence the search to move to areas that are potentially fruitful? This may be very difficult, if not impossible. In some cases the long-term memory may hold some clues, but often there is no way of telling at all.

28.2 SUBCOST-GUIDED SIMULATED ANNEALING

However, in one particular set of circumstances, there are extra clues. This is where the overall objective function is a *compound*, made up of a number of *subobjectives*. Even when the search process has no memory component at all, we have further information about any perturbation - not only the change in total cost, but also the change in each individual *subcost*.

The rationale behind a subcost-guided approach was originally intuitive. If one perturbation improves one subcost significantly even though other subcosts are increased by a greater amount, thus leading to an overall increase in cost, then it may well be possible to find another perturbation which decreases these latter subcosts without increasing the former subcosts again.

For example, in a school timetabling application, one perturbation may remove the clash for one teacher but cause clashes for two others. However, subsequent perturbations may remove these latter clashes without reintroducing the former one.

This intuitive argument has been backed up by some results (see Wright [8], [9]) for a variant of a Tabu Search / Tabu Thresholding method. These results appear to show that the approach is valuable for some very complex real problems. Wright [10] reports on some initial research into incorporating the subcost-guided approach into Simulated Annealing for school timetabling problems; this paper builds on that research, making it both wider and deeper. Some more difficult school timetabling problems have been added and the research is extended to two other types of problem. The experimental design and analysis of results have also been extended.

In *Subcost-Guided Simulated Annealing* (SGSA) the acceptance/rejection decision for a non-improving perturbation depends not only on the usual three parameters (C , the increase in total cost, T , the temperature and R , a random number between 0 and 1) but also on B , the lowest increase in any individual subcost, as long as $B < 0$. The rationale behind this is that it is intuitively

reasonable to suppose that perturbations are more likely to lead eventually to improved solutions if B is very much less than zero (i.e. a substantial improvement for that subcost).

The acceptance rules are as follows:

- If $C \leq 0$, the perturbation is accepted.
- If $C > 0$ and $B \geq 0$, the standard SA rule is used.
- If $C > 0$ and $B < 0$, the perturbation is accepted if $R < \exp(-C\exp(\theta B/C)/T)$, where θ is a parameter ≥ 0 .

This formula looks quite complicated, but it has been chosen with care, although it means that SGSA does not have the same convergence properties as SA. As B tends to zero from below, the formula tends towards the standard SA formula, and the probability of acceptance increases with $|B|/C$, meaning that perturbations with higher benefits to individual subcosts are more likely to be accepted, *ceteris paribus*. The right hand side of the inequality approaches 1 as B tends to $-\infty$, but it is always below 1. Similarly the probability of acceptance still tends to zero as either T tends to zero or C tends to ∞ , and tends to one as C tends to zero. Thus the overall cost change, the random number and the temperature still maintain their important functions. The value of θ controls the influence of the subcosts — if θ is zero the method is standard SA.

28.3 THE EXPERIMENTAL PROBLEMS USED

28.3.1 School timetabling problems

Some initial experiments had already been carried out on seven school timetabling problems (Wright [10]). This set of problems was extended by the addition of three further problems. For each problem the overall cost is the sum of subcosts relating to clashes between teachers, insufficiency of teachers within particular departments and whether or not particular lessons are timetabled in the same time-slot.

Each of the ten problems is a simplified variation of a real timetabling problem involving a large comprehensive school in the UK. The simplifications involve not only a reduction in size but also the removal of some types of subcost, for example those relating to the spread of particular types of lesson through the week. The simplification was necessary in order to make a sufficient amount of experimentation feasible within a reasonable timescale. Fuller details of the real application are given in Wright [9].

In four of these ten problems (Problems 3, 8, 9 and 10) there are significantly fewer teachers available to teach roughly the same set of classes. This implies that there are inevitably more clashes and insufficiencies within departments and thus that a good solution will have not only higher total cost, but also a large number of non-zero subcosts. This extra complexity of a good solution

suggests (though without proof) that these four problems can be considered “harder” than the other six.

28.3.2 Sports timetabling problem

The next problem used was a simplified form of the first stage of the sports timetabling problem described by Wright [8], using the real 1999 data. The total cost is the sum of hundreds of subcosts of various types, relating to whether preferred dates are given for home matches, whether gaps in the schedule are given at preferred times, the overall balance of home fixtures and other factors. Again, the simplification was necessary in order to make experimentation feasible.

28.3.3 Flowshop problem

A simple flowshop example was created, with 50 jobs of three types and four machines. Each job has a given duration and due date, and machine set-up times were involved when switching between types of job. The subcosts involved are the makespan and, for each job, a lateness cost of the form $A + Bt$, where t is the number of days by which the job is finished late, and, for some jobs, an earliness cost of the form $C + Du$, where u is the number of days by which the job is finished early. The overall cost is the sum of the subcosts.

Compared to the timetabling examples, this problem has different properties in terms of the relationships between subcosts, in such a way as to suggest that SGSA may not be in fact as useful as for this problem. This is discussed further after presentation of the results for this problem in Section 28.6.

28.4 SCHOOL TIMETABBING EXPERIMENTS

For each problem, 51 runs of 300,000 perturbations each were made for each of ten different values of $\theta(0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 5 \text{ and } 10)$. The experiments all used a simple geometric cooling schedule, with a starting temperature one hundred times the ending temperature. Initial runs were made for each of the ten problems with $\theta = 0$, in order to find a starting and ending temperature that produced good results. These same temperatures were then used for all the runs for all values of θ .

The number of perturbations was chosen subjectively in order to allow each run a “good chance” of reaching a good solution. The values of θ were chosen so as to be sure of including the best value. The number of runs was not predetermined, but was the number completed after a predetermined overall length of time.

The average costs are given in Table 28.1. Underlined costs are those which are lower than the results for $\theta = 0$ at a statistical level of significance greater than 95%. “Best θ ” is the value of θ which gives the lowest average cost. “Best % imp” is the percentage improvement achieved by using Best θ rather than $\theta = 0$. Fuller details of the results are available on request to the author.

Problem	Values of θ										Best θ	Best % impr.
	0	0.5	1	1.5	2	2.5	3	3.5	5	10		
1	305	297	296	<u>292</u>	300	299	305	306	329	554	1.5	4.3
2	242	239	<u>235</u>	237	239	237	243	244	277	440	1.0	3.4
3	727	<u>699</u>	<u>679</u>	<u>671</u>	<u>669</u>	<u>672</u>	<u>644</u>	<u>665</u>	<u>703</u>	1219	3.0	11.3
4	385	381	382	<u>371</u>	377	383	389	383	414	686	1.5	3.5
5	247	242	250	<u>231</u>	<u>233</u>	<u>235</u>	<u>235</u>	240	275	427	1.5	6.3
6	270	272	274	267	264	272	268	282	302	562	2.0	2.2
7	200	<u>187</u>	190	<u>185</u>	<u>185</u>	192	194	197	226	414	2.0	7.3
8	3868	3843	<u>3811</u>	<u>3765</u>	<u>3768</u>	<u>3776</u>	<u>3761</u>	<u>3816</u>	3899	4568	3.0	2.8
9	1565	1537	<u>1451</u>	1422	<u>1396</u>	<u>1390</u>	<u>1355</u>	<u>1368</u>	<u>1420</u>	2355	3.0	13.4
10	4445	<u>4401</u>	<u>4366</u>	4263	4168	4170	4170	<u>4208</u>	4405	5608	2.0	6.2

Table 28.1 Average costs for the school timetabling problems.

For every problem the average cost first decreases and then increases as θ increases from 0 to 10.

For problems 3, 8, 9 and 10 (the four “harder” problems) highly significant improvements are seen for all values of θ between 1 and 3.5, despite the fact that the temperature cooling schedule was selected so as to suit the standard method ($\theta = 0$). The significance levels for the best value of θ are above 99.9999% for each of these four problems. The best value of θ is generally about 3.0.

For the other six (“easier”) problems a smaller improvement is seen. However, five of these are statistically significant at 95% for the best value of θ , with the weakest result (problem 6) being significant at 85%. The best value of θ is lower than for the harder problems at around 1.5.

28.5 SPORTS FIXTURE SCHEDULING EXPERIMENTS

100 runs of 200,000 perturbations each were made for five different values of θ , including zero, and for a number of different cooling schedules.

The reason for using a variety of cooling schedules was that it was felt that the best choice of schedule might depend on the value of θ . The higher the value of θ , the higher the proportion of perturbations accepted at a given temperature, *ceteris paribus*. Thus the previous set of experiments, for which the cooling schedules were chosen to be suitable for $\theta = 0$, was felt to be possibly “unfair” to higher values of θ .

Again only geometric cooling schedules were used, this time with the start temperature always being five times the end temperature. For each value of θ , starting temperatures of 500, 1000 and 1500 were used. Extra starting temperatures were also used for lower values of θ to ensure that the best starting temperature was included within the range used. Thus a wider range of start temperatures was used for lower values of θ than for higher values.

The initial results showed clear improvements for non-zero values of θ compared with $\theta = 0$ but, because of high standard deviations, no results were

statistically significant at this stage. Therefore a further 754 runs were made for the best cooling schedule for $\theta = 0$ (starting temperature 3000) and the best combination for non-zero θ ($\theta = 2$, starting temperature 1500). The number of runs was not explicitly predetermined but was the number completed after a predetermined overall length of time.

The full set of average costs is given in Table 28.2. The two results derived from 854 runs rather than 100 are in bold; the result for $\theta = 2$ is better than the result for $\theta = 0$ at a very high level of statistical significance. The best average cost for each value of θ is underlined (the extra runs revealed that the best starting temperature for $\theta = 2$ may in fact be 1000 rather than 1500 as suggested by the first 100 runs). Fuller details of the results are available on request to the author.

Start temperature	Values of θ				
	0	1	2	3	5
500	34264	24307	20879	19450	18648
1000	19926	18056	<u>15665</u>	<u>15977</u>	<u>18334</u>
1500	16819	<u>15932</u>	15720	16540	21610
2000	16355	15981	15928		
2500	16187	16216	16666		
3000	16184				
3500	16804				
4000	17288				

Table 28.2 Average costs for the sports fixture timetabling problem.

The intuitive hypothesis concerning the effect of the value of θ upon the best cooling schedule appears to have been verified, since the results show that, as θ increases, the best starting temperature decreases. Also, as before, as long as the best cooling schedule is used for each value of θ , the average cost first decreases and then increases as θ increases from zero.

28.6 FLOWSHOP EXPERIMENTS

Nine cooling schemes were used for each of four values of θ : 0, 0.5, 1 and 1.5. Each cooling scheme was geometric with starting temperature 100 times the ending temperature. For each combination at least 165 runs were made of 200,000 perturbations each (the variation in number of runs reflects the fact that the runs were made at different times and that for one set of runs the experiment was terminated early by a powercut).

The average costs are given in Table 28.3 (fuller details are available on request to the author). The best average cost for each value of θ is underlined.

The results here are by no means as conclusive as before. While the best result for $\theta > 0$ is just significantly better at 95% than the best result for $\theta = 0$, the overall picture is far from clear-cut, though we can be fairly confident in

concluding that a value of 0.5 or 1 for θ is certainly no worse than a value of zero.

On reflection, this is perhaps not surprising. The rationale behind subcost-guided methods, as outlined earlier, is that it can be beneficial to accept a perturbation which decreases one subcost at the expense of an increase in one or more other subcosts (call these Type A perturbations), because it may then be possible to make subsequent changes which remove the latter without jeopardising the former (call these Type B perturbations).

Start temperature	Values of θ			
	0	0.5	1	1.5
1	10132	10058	10103	<u>10055</u>
2.5	10092	10064	10054	10056
5	<u>10056</u>	10062	<u>10040</u>	10061
10	10074	<u>10035</u>	10053	10069
25	10075	10054	10070	10076
50	10075	10073	10073	10119
100	10084	10092	10095	10246
250	10121	10142	10264	10728
500	10232	10299	10711	11195

Table 28.3 Average costs for the flow shop problem.

For the flowshop example, however, a perturbation which inserts a job earlier in the sequence obviously makes other jobs later than before. Thus a decrease in the lateness cost of one job is almost certain to increase the lateness cost of another. Since for many jobs there was no earliness cost, and for all jobs the lateness cost was higher than the earliness cost, this means that, once a reasonably good solution has been reached, there will be very few perturbations of Type B above. To put it another way, the subcosts are very highly correlated (negatively). Thus the intuitive argument for SGSA is much less strong.

This is a very simplified analysis, but the results do appear to back up the tentative suggestion that it is the very nature of solution space itself which determines how effective, if at all, the subcost-guided approach will be. Of course, the nature of solution space is very difficult to define and measure, though some authors including Maret and Wright [9] have made a start.

The full results also show another interesting feature. While the best solutions of all were obtained with $\theta = 0$, these results had a much higher variance than those for $\theta = 0.5$ or $\theta = 1$, and the latter gave better worst solutions than $\theta = 0$. This suggests that it may be sensible to use a non-zero value of θ if there is time only for a few runs, but $\theta = 0$ if time is plentiful.

28.7 QUALITY AND ROBUSTNESS OF RESULTS

It is not possible to tell for certain just how good the results of any of these experiments are in absolute terms, since the optimal solutions are not known.

Moreover, although the school timetabling and sports fixture timetabling problems were based on real problems, there were fairly substantial simplifications such that comparison with real solutions is not possible. However, the purpose of the research was to compare SGSA with SA rather than to reach any conclusions about the absolute value of SGSA.

For robust results, ideally the parameters which give the best average costs would also give the lowest standard deviations. Examination of the full results shows that, for most of the problems used, this is approximately the case.

28.8 CONCLUSIONS AND FURTHER WORK

Overall it appears that SGSA can be of great value for compound-objective problems. It appears that its value is greatest for “difficult” problems where the subcosts are not too strongly correlated, but that it never appears to have a harmful effect as long as θ is not too high.

Improvements appear to apply over a wide range of values of θ , so it may not be too important to get a precise value. However, it could prove very worthwhile to conduct research into the possibility of relating the best value of θ to some measure of the nature of the solution space.

Further research is needed into the precise form of the acceptance formula. The formulation used is only one possible way of using the subcost information — many others are possible and should be tried.

For example, it may be worth including information about more than just the most improved subcost in the acceptance formula, possibly even including all improved subcosts, each with its own value of θ .

An alternative could be to try $\exp(-C/(T - \theta B/C))$ as the acceptance probability, equivalent to increasing the temperature for a perturbation giving an improvement to an individual subcost — the bigger the subcost improvement, the larger the equivalent increase in temperature, and thus the higher the acceptance probability.

Yet another possibility would be to allow the value of θ to vary according to the progress of the search - perhaps it could be increased when diversification is needed, and decreased when intensification is more appropriate.

References

- [1] K.A. Dowsland. Simulated Annealing. In: *Modern Heuristic Techniques for Combinatorial Problems*, C.R. Reeves, editor, pages 20–69, Blackwell, 1993.
- [2] K.A. Dowsland. Nurse Scheduling with Tabu Search and Strategic Oscillation. *European Journal of Operational Research*, 106:393–407, 1998.
- [3] G. Dueck. New Optimization Heuristics: The Great Deluge Algorithm and the Record-to-Record Travel. Technical Report 89.06.011, IBM Germany, Heidelberg, 1990.

- [4] F. Glover. Tabu Thresholding: Improved Search Trajectories by Non-Monotonic Search Trajectories. *ORSA Journal on Computing*, 7:426–442, 1995.
- [5] F. Glover and M. Laguna. *Tabu Search*. Kluwer, 1997.
- [6] R.C. Maret and M.B. Wright. *A Preliminary Investigation into the Performance of Heuristic Search Methods Applied to Compound Combinatorial Problems*. In: *Metaheuristics Theory and Applications*, I.H. Osman and J.P. Kelly, editors, pages 299–318, Kluwer, 1996.
- [7] N. Mladenovic and P. Hansen. Variable Neighborhood Search. *Computers and Operations Research*, 24:1097–1100, 1997.
- [8] M.B. Wright. Timetabling County Cricket Fixtures using a Form of Tabu Search. *Journal of the Operational Research Society*, 45:758–770, 1994.
- [9] M.B. Wright. School Timetabling using Heuristic Search. *Journal of the Operational Research Society*, 47:347–357, 1996.
- [10] M.B. Wright. Subcost-Guided Search — Experiments with Timetabling Problems. To appear in: *Journal of Heuristics*.

29 A PRUNING PATTERN LIST APPROACH TO THE PERMUTATION FLOWSHOP SCHEDULING PROBLEM

Takeshi Yamada

NTT Communication Science Laboratories
2-4 Hikaridai, Seika-cho, Soraku-gun, Kyoto 619-0237, Japan
yamada@cslab.kecl.ntt.co.jp

Abstract: This paper investigates an approach to the permutation flowshop scheduling problem based on Tabu Search with an additional memory structure called a ‘pruning pattern list’. The pruning pattern list approach allows a better use of the critical block information. A solution of the flowshop scheduling problem is represented by a permutation of job numbers. A pruning pattern is generated from a solution by replacing job numbers inside a critical block with ‘wild cards’ so that solutions that ‘match’ the pattern would be excluded from the search. A set of pruning patterns, which is called a ‘pruning pattern list’, is used to navigate the search by avoiding solutions that would match any pattern on the list. Computational experiments using benchmark problems demonstrate the effectiveness of the proposed approach.

29.1 INTRODUCTION

The scheduling problem investigated in this paper is called the permutation flowshop scheduling problem (PFSP) and is conventionally designated as $n/m/P/C_{max}$ [1], where n jobs have to be processed on m machines in the same order. The processing of each job on each machine is an operation, which requires the exclusive use of the machine for an uninterrupted duration called the processing time. P indicates that only the permutation schedules are considered, where the order in which each machine processes the jobs is identical for all machines. Hence, a schedule is uniquely represented by a permutation of jobs. The objective here is to find a schedule that minimizes the makespan C_{max} , the time at which the last job is completed on the last machine. The problem is strongly \mathcal{NP} -hard [1], therefore complete enumeration methods are not computationally

practical as the problem size increases. Approximation methods to solve the problem are categorized to two types: one is constructive methods and another is iterative improvement methods. Constructive methods have been proposed by Campbell et al. [2], Dannenbring [3] and Nawaz et al. [4]. As for iterative improvement methods, Osman and Potts [8] and Ogbu and Smith [6] proposed simulated annealing methods, and Widmer and Herz [7] and Taillard [10] proposed tabu search methods. More recently, Nowicki and Smutnicki proposed a very powerful approximation method based on tabu search [5].

In this paper, an approximation method based on Tabu Search with an additional memory structure called “pruning pattern list” is proposed. In short, the pruning pattern list approach allows a better use of the critical block information. A pruning pattern is constructed from a solution represented by a permutation of job numbers by replacing some of job numbers inside a critical block by wild cards (\star -s) so that solutions that ‘match’ the pattern will be excluded from the search hereafter. This means that any modification of a solution, regardless of inside or outside the critical block, is discouraged if the resulting sequence would match the pattern. A list of pruning patterns generated from good schedules collected in the course of a search process is used to prevent the search from visiting already searched and no longer interesting region again and again.

29.2 BASIC CONCEPTS

A critical path is a sequence of operations starting from the first operation on the first machine M_1 and ending with the last operation on the last machine M_m . The starting time of each operation on the path, except for the first one, is equal to the completion time of its preceding operation—that is, there is no idle time along the path. Thus, the length of the critical path is the sum of the processing times of all the operations on the path and equals to C_{max} . There can be more than one critical path in a schedule.

The operations on a critical path can be partitioned into subsequences, called *critical blocks*, according to their associated machines. A critical block consists of maximal consecutive operations on the same machine, or to put it more simply, a subsequence of associated jobs. Most of the following notations have previously been used by Nowicki and Smutnicki [5]. Consider a schedule represented by a permutation π . Let B_1, \dots, B_k be a set of all critical blocks that contains more than one job and let m_l be the index of the machine associated with B_l . Let $\pi(u_l)$ be the first job of B_l (and the last job of B_{l-1}). Then the ‘inside’ of B_l , denoted by \hat{B}_l , is defined as follows:

$$\hat{B}_l = \begin{cases} B_l \setminus \{\pi(u_{l+1})\} & \text{if } l = 1 \text{ and } m_l = 1 \\ B_l \setminus \{\pi(u_l)\} & \text{if } l = k \text{ and } m_l = n \\ B_l \setminus \{\pi(u_l), \pi(u_{l+1})\} & \text{otherwise.} \end{cases}$$

Figure 29.1 shows an example of schedule $\pi = 4, 5, 6, 1, 2, 3, 8, 7$ for a problem with $n = 8$ jobs and $m = 6$ machines represented by a grid graph, that is taken from [5]. In the figure, the vertical axis corresponds to machines and

the horizontal axis to jobs. Each circle represents an operation and arrow precedence relation between operations. A critical path is marked by thick lines. In this example, there are four critical blocks B_1, B_2, B_3, B_4 that contain more than one job. B_2 on machine 3, for example, consists of four jobs 5, 6, 1 and 2, and \hat{B}_2 consists of jobs 6 and 1. Likewise \hat{B}_4 on machine 6 consists of jobs 8 and 7.

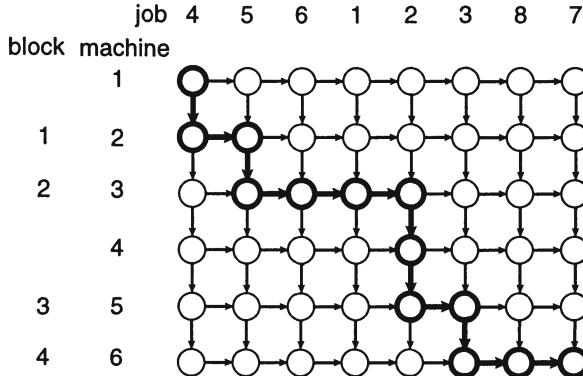


Figure 29.1 A grid graph representation of a solution to a problem of 8 jobs and 6 machines.

A neighborhood $N(x)$ of a point x in a search space can be defined as a set of new points that can be reached from x by exactly one transition or move (a single perturbation of x). One of the well-known transition operators for PFSP is the *shift move* which takes a job from its current position and re-inserts it in another position. Let $v = (a, b)$ be a pair of positions in π . Here, v defines a move that removes the job $\pi(a)$ from a position a and re-inserts it in a position b . If $a < b$, the resulting schedule is represented by $\pi_v = \pi(1), \dots, \pi(a-1)\pi(a+1), \dots, \pi(b), \pi(a), \pi(b+1), \dots, \pi(n)$, and if $a > b$, $\pi_v = \pi(1), \dots, \pi(b), \pi(a), \pi(b+1), \dots, \pi(a-1)\pi(a+1), \dots, \pi(n)$. A neighborhood $N(V, \pi)$ is defined as the set of all schedules obtained by shift moves in $V = \{(a, b) : b \notin \{a-1, a\}, a, b \in \{1, \dots, n\}\}$.

Let $W_l(\pi)$ be a set of moves restricted to the inside of B_l , namely $W_l(\pi) = \{(a, b) \in V | a, b \in \hat{B}_l\}$ and $W(\pi) = \bigcup_{l=1}^k W_l(\pi)$, then the so-called ‘block property’ is formulated as follows:

Property A (block property): For any schedule β , $\beta \in N(W(\pi), \pi) \implies C_{max}(\beta) \geq C_{max}(\pi)$.

According to property A above, no move in $W(\pi)$ can directly improve schedule π . Therefore, it is reasonable, for computational efficiency, to reduce the size of the neighborhood $N(V, \pi)$ by eliminating moves in $W(\pi)$, and to use a new neighborhood $N(V \setminus W(\pi), \pi)$, which we call here a “critical block neighborhood”.

29.3 TABU SEARCH

Nowicki and Smutnicki [5] proposed a Tabu Search (TS) method for the PFSP. The tabu list used in their approach is described as follows: when $v = (a, b)$ is performed on π , a pair of jobs $(\pi(a), \pi(a+1))$ (if $a < b$) or $(\pi(a-1), \pi(a))$ (if $a > b$) is stored as ‘tabu’ in the tabu list T of length tl . Then, a move $v = (a, b)$ cannot be performed on β if $a < b$ and $\{(\beta(j), \beta(a)) \mid j = a+1, \dots, b\} \cap T \neq \emptyset$ or if $a > b$ and $\{(\beta(a), \beta(j)) \mid j = b, \dots, a-1\} \cap T \neq \emptyset$. The neighborhood $N(V \setminus W(\pi), \pi)$, which is a subset of $N(V, \pi)$, is still too big for problems of even moderate size. Thus, it is impracticable to evaluate all the neighbors in $N(V \setminus W(\pi), \pi)$, so Nowicki and Smutnicki use a subset of $N(V \setminus W(\pi), \pi)$ as a neighborhood structure. Alternatively, one can make the evaluation probabilistic: a member α from $N(V \setminus W(\pi), \pi)$ is selected at random and accepted if $C_{max}(\alpha) < C_{max}(\pi)$, otherwise probabilistically accepted according to the Metropolis probability $P_c(\alpha) = \exp(-\Delta C_{max}/c)$, where $\Delta C_{max} = C_{max}(\alpha) - C_{max}(\pi)$.

29.4 PRUNING PATTERN

A pruning pattern $[\pi]_l$ associated with π and its critical block B_l is derived from π by replacing the jobs that belong to \hat{B}_l by \star as follows:

$$[\pi]_l(j) = \begin{cases} \star & \text{if } j \in \hat{B}_l, \\ \pi(j) & \text{otherwise.} \end{cases}$$

For example, the pruning pattern corresponding to $\pi = 4, 5, 6, 1, 2, 3, 8, 7$ and B_2 in Figure 29.1 is $[\pi]_2 = 4, 5, \star, \star, 2, 3, 8, 7$. The makespan of $[\pi]_l$ is defined as the makespan of π , namely, $C_{max}([\pi]_l) := C_{max}(\pi)$. The \star means a wild card, and a permutation β ‘matches’ $[\pi]_l$ if $\beta(j) = [\pi]_l(j)$ at all but \star positions. For example $4, 5, 6, 1, 2, 3, 8, 7$ and $4, 5, 1, 6, 2, 3, 8, 7$ both match $[\pi]_2$. $[\pi]_l$ and a set of all permutations that match $[\pi]_l$ are identified. It is clear that $\mathcal{N}(W_l(\pi), \pi) \subset [\pi]_l$. The block property can be reformulated using $[\pi]_l$ as follows:

Property B: For any schedule β and any pruning pattern $[\pi]_l$, $\beta \in [\pi]_l \implies C_{max}(\beta) \geq C_{max}([\pi]_l)$.

Property C below is a corollary of Property B:

Property C:] For any two pruning patterns $[\pi]_i$ and $[\phi]_j$, $[\pi]_i \subset [\phi]_j \implies C_{max}([\pi]_i) \geq C_{max}([\phi]_j)$.

Property B suggests that when a new, possibly good, solution π is found during the search, $[\pi]_l$ identifies a region where no solutions are better than π , and that excluding $[\pi]_l$ from the search space can reduce the size of the search space without eliminating the global optima.

29.5 PRUNING PATTERN LIST APPROACH

The basic idea of the pruning pattern list approach is to reduce the size of the search space effectively through storing the ‘important’ pruning patterns that

- [1] At the beginning of the search, PL is initialized as a list of empty pl elements.
- [2] After $N(\pi)$ is calculated from π , $N^P(\pi)$ is initialized as $N(\pi)$. For each $\alpha \in N^P(\pi)$, if α matches any pattern $[\beta] \in PL$, then α is removed from $N^P(\pi)$, resulting $N^P(\pi) := N^P(\pi) \setminus \alpha$, and the ‘access count’ of $[\beta]$ is incremented. $N^P(\pi)$ is used as the new neighborhood of π .
- [3] When a cost-decreasing solution α is selected from the neighborhood $N^P(\pi)$ of the current solution π ($C_{max}(\alpha) < C_{max}(\pi)$), then its longest pruning pattern $[\alpha]$ is stored in the list.
- [4] All the existing patterns $[\beta]$ in the list that are subsets of a newly stored pattern $[\alpha]$ ($[\beta] \subset [\alpha]$) are removed and substituted by empty patterns.
- [5] If the number of non-empty patterns on PL exceeds pl , then the least accessed pattern $[\gamma]$ is removed from the list.

Figure 29.2 The pruning pattern list approach to the PFSP.

correspond to a long critical block of a good solution. Let $[\pi]$ (without suffix) be the pruning pattern that corresponds to the longest critical block of schedule π . The pruning pattern list PL with length pl is maintained and updated as shown in Figure 29.2. In the figure, $N(\pi)$ represents the neighborhood of π . $N(V \setminus W(\pi), \pi)$ or its subset is normally used as $N(\pi)$. There is no good reason to keep patterns that are rarely accessed in the list. Therefore such patterns are replaced by new patterns. According to property C, it is also not necessary to keep pruning patterns in the list that are subsets of other patterns. Thus they are removed.

Starting from an initial solution, the local search iteratively replaces the current solution with one of its neighbors. In advanced local search strategies, such as Simulated Annealing (SA) and TS, cost-increasing neighbors can be accepted as well as cost-decreasing ones. Accepting cost-increasing moves enables the search to escape from the local optima, but this may also cause revisiting of previously evaluated points in the search space – something that is wasteful of computing resources in itself, and which also means the search is not adequately diversified. One of the main aims of tabu search is to discourage such revisiting. This can be accomplished by means of an explicit ‘tabu list’ of points previously visited, but normally it is easier, and more efficient, to record specific attributes of such points, or of moves that would lead towards them. Nevertheless, recording attributes of the points, and not the points themselves, can risk treating even moves that are better than any solution obtained so far as tabu. This is one of the main reasons why aspiration criteria should be introduced into TS.

In the case of the pruning pattern list approach, property B guarantees that a move that matches a pruning pattern is never better than the best solution obtained so far. This means that an aspiration criterion is not necessary, and it also means that the pruning pattern list can serve as a longer-term memory to prevent the search getting stuck in an already searched and no longer interesting

region. Unlike the tabu list, an old pattern can remain in the pruning list as long as it is accessed frequently. If $N(V \setminus W(\pi), \pi)$ or its subset is used as the neighborhood of the current solution π , it does not contain any solution that matches $[\pi]_l$, even without PL . However, it is still possible that the region $[\pi]_l$ would be revisited in some later stages without any better solutions than π being found. The size of PL is fixed to pl mainly for computational efficiency. Theoretically, $N^P(\pi)$ may not satisfy the connectivity property that $N(V \setminus W(\pi), \pi)$ does. It should be noted that the pruning pattern list is treated in a similar way that the population is treated in Genetic Algorithms (GAs), especially in the steady state model [12]. The access count of the pruning pattern corresponds to the fitness function of the individual in GAs.

29.6 EXPERIMENTAL RESULTS

The pruning pattern list approach described in the previous section can be embedded into any local search method, including TS, SA, and even GAs [13, 9]. Here, a simple probabilistic version of the TS described in Section 29.3 is used as a test case. The programs are coded in *C*.

The graph on the left side of Figure 29.3 shows the time evolution of the makespan averaged over 30 runs of the TS with and without the pruning pattern list. This uses the ta041 problem, which is one of Taillard's benchmarks for size

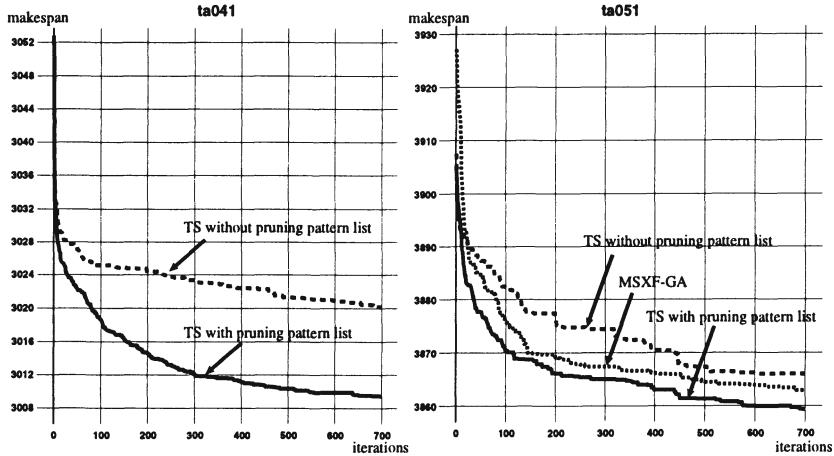


Figure 29.3 The time evolutions of makespans for the ta041 (50 jobs and 10 machines) and ta051 (50 jobs and 20 machines) problems.

50×20 (50 jobs and 10 machines) [11]. The parameters used are $c = 3.0$, $tl = 7$ and $pl = 10$. The pruning pattern list is updated after every 10000 TS evaluations for 700 times (thus total 10000×700 TS evaluations). One run takes about 10 minutes on an HP 700 workstation. The best and worst makespans obtained among 30 runs are 3000 and 3016 respectively when TS with the pruning pattern list is used, and 3010 and 3025 when TS without the pruning

pattern list is used. It can be seen that the pruning pattern list approach improves both the solution quality and the speed. Without the pruning pattern list 12 runs out of 30 were trapped at a local minimum of makespan = 3025, which Nowicki and Smutnicki reported as the new reference makespan for this problem [5]. This suggests that their TS method can also be improved by incorporating the pruning pattern list approach.

The pruning pattern list approach is also applied to 50×20 (50 jobs and 20 machines) problems of Taillard's benchmarks (ta051 - ta060). The parameters used are the same as in the ta041 case and the results are averaged over 10 replications instead of 30. The results are summarized in Table 29.1, where the columns labeled TS+PL and TS show the average performance with and without the pruning pattern list respectively. The column labeled NOW shows the best makespan reported in [5]. The results of TS with the pruning pattern list generally outperform the results of TS without the pruning pattern list, except for the ta057 problem. As typical examples, the graph on the right side of Figure 29.3 shows the time evolution of the average makespan for ta051 problem. It also includes the computationally equivalent MSXF-GA results reported in [9] for comparison. Figure 29.4 shows the time evolution for other 50×20 problems.

Problem	TS+PL	TS	NOW (best)
051	3859.4	3866.0	3875
052	3708.0	3713.4	3715
053	3653.9	3657.4	3668
054	3734.0	3737.2	3752
055	3617.1	3625.7	3635
056	3689.3	3691.7	3698
057	3712.8	3712.4	3716
058	3701.4	3706.8	3709
059	3756.9	3762.3	3765
060	3765.7	3767.3	3777

Table 29.1 Comparison with and without the pruning pattern list for problems size 50×20 .

To see how the proposed approach behaves with growing problem sizes, the experiments are extended to Taillard's 100×20 (ta081 - ta090), 200×20 (ta101 - ta110) and 500×20 (ta111 - ta120) problems. For 100×20 (ta081 - ta090), 200×20 (ta101 - ta110) problems, the results are summarized in Table 29.2. Here, the pruning pattern list is updated after every 10000 TS evaluations for 160 times, which resulted in comparable and slightly better performances than those reported in [5]. The parameters used here are $c = 3.0$, $tl = 7$, and $pl = 5$, and the results are averaged over 10 replications. One run for each 100×20 and 200×20 problems takes about 6 and 12 minutes on an HP 700 workstation respectively. Preliminary experiments show that the computational overhead of the pruning pattern calculation is about 8 % when $pl = 5$ and increases linearly

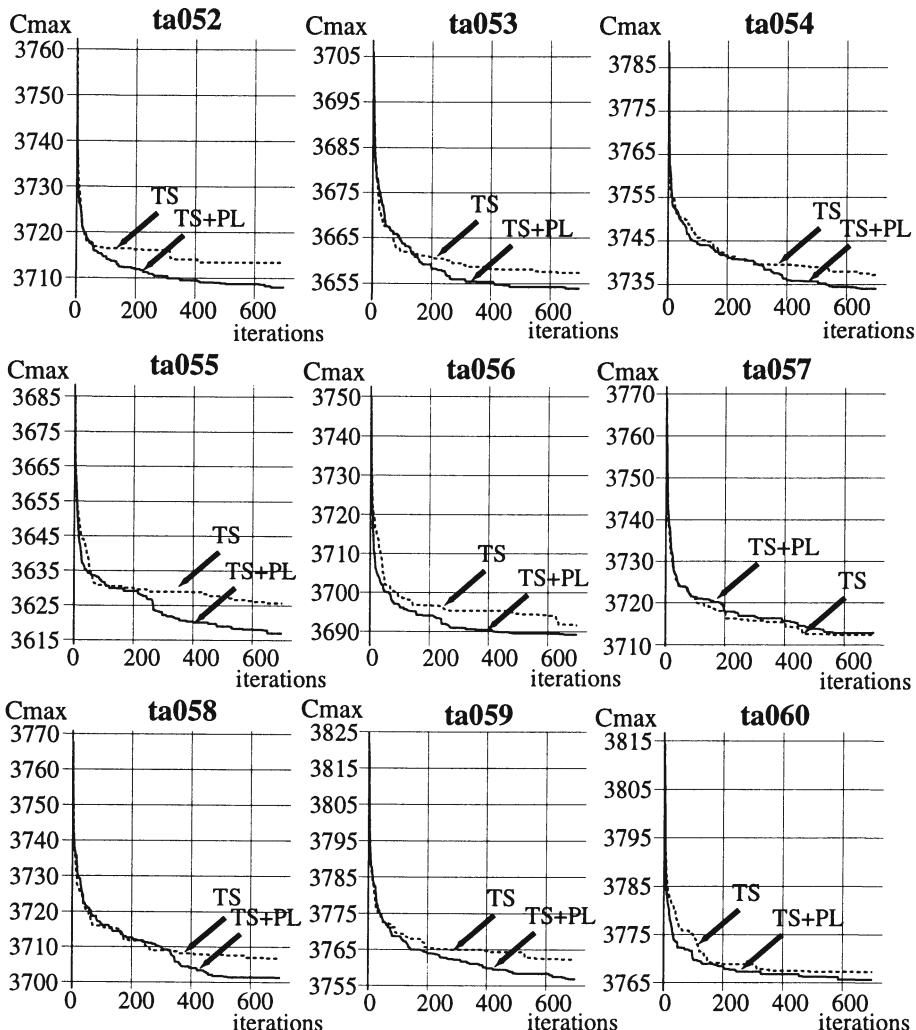


Figure 29.4 Time evolution for the other nine Taillard problems with 50 jobs and 20 machines (ta052 – ta060).

as the size of PL but with no significant performance improvement, therefore $pl = 5$ seems a reasonable value unless a more efficient way to scan the pruning pattern list is invented. It can be seen that the results of TS with the pruning pattern list generally outperform the results of TS without the pruning pattern list with some exceptions, whereas the gaps between TS+PL and TS become decreasing and less significant (less than 0.1 %).

Table 29.2 shows the results for Taillard's 500×20 (ta111 - ta120) problems. In the columns TS+PL and TS, the average, best and worst results over 10

Problem	TS+PL	TS	NOW		Problem	TS+PL	TS	NOW	
			(best)	worst				(best)	worst
081	6270.6	6270.2	6286	101	11285.5	11305.0	11294		
082	6228.9	6233.0	6241	102	11358.5	11349.6	11420		
083	6309.8	6314.8	6329	103	11428.3	11430.3	11446		
084	6302.5	6306.6	6306	104	11357.7	11366.9	11347		
085	6358.1	6362.1	6377	105	11301.5	11310.3	11311		
086	6420.8	6431.1	6437	106	11269.7	11275.0	11282		
087	6328.2	6333.6	6346	107	11463.8	11459.9	11456		
088	6470.1	6471.5	6481	108	11413.5	11420.9	11415		
089	6327.0	6332.2	6358	109	11302.8	11313.2	11343		
090	6469.3	6476.9	6465	110	11404.6	11418.3	11422		

Table 29.2 Comparison with and without the pruning pattern list for problems size 100×20 and 200×20 .

replications are reported respectively. To keep the results comparable with those reported in [5], the number of iterations is doubled and the pruning pattern list is updated after every 10000 TS evaluations for 320 times with the other parameters same as 100×20 and 200×20 problems. Due to the increase both in the problem size and the number of iterations, one run now takes about 60 minutes. The same tendencies can be observed here again and TS+PL runs are generally better than TS runs with some exceptions, in terms of both the average and the best results, whereas the gaps between TS+PL and TS become decreasing.

To see the maximal performance of the TS+PL method, the number of iterations are doubled again for TS+PL runs only, resulting in best, worst and average makespans shown in the columns under the label “TS+PL (long)” in Table 29.2.

Problem	TS+PL			TS			TS+PL (long)			NOW (best)
	average	[best	worst]	average	[best	worst]	average	[best	worst]	
111	26324.0	[26198	26398]	26321.0	[26263	26403]	26159.6	[26139	26182]	26189
112	26687.6	[26633	26735]	26717.0	[26629	26805]	26625.5	[26600	26642]	26629
113	26465.6	[26435	26522]	26483.6	[26466	26512]	26431.4	[26417	26435]	26458
114	26547.2	[26522	26590]	26555.6	[26533	26584]	26525.0	[26520	26541]	26549
115	26371.4	[26354	26412]	26399.6	[26383	26413]	26356.2	[26354	26366]	26404
116	26589.2	[26535	26621]	26584.0	[26564	26621]	26516.3	[26498	26543]	26581
117	26432.4	[26412	26481]	26438.4	[26412	26463]	26413.0	[26412	26422]	26461
118	26678.8	[26619	26748]	26659.4	[26619	26736]	26615.9	[26615	26619]	26615
119	26149.8	[26097	26196]	26151.0	[26101	26235]	26078.8	[26050	26103]	26083
120	26600.0	[26546	26669]	26608.6	[26572	26684]	26526.4	[26497	26552]	26527

Table 29.3 Computational results for problems size 500×20 .

29.7 CONCLUSION

The pruning pattern list approach to the makespan-minimizing permutation flowshop scheduling problem is proposed and embedded into a Tabu Search method. The experimental results demonstrate the effectiveness of the proposed approach both for medium- and large-size problems, while the gap between with and without the pruning pattern list seems to be decreasing as the problem size increases. Future research will aim to implement more efficient ways to scan the pruning pattern list and find a match more quickly. The proposed approach can also be embedded into GAs. However, the implementation details including whether each individual should have its own pruning pattern list and should occasionally exchange it with others, or a single list should be shared by all the members in the population are yet to be investigated.

References

- [1] A.H.G. Rinnooy Kan. *Machine Scheduling Problems: Classification, Complexity and Computations*. Freeman, 1979.
- [2] H.G. Campbell, R.A. Dudek, and M.L. Smith. A Heuristic Algorithm for the n Job m Machine Sequencing Problem. *Management Science*, 16:630–637, 1970.
- [3] D.G. Dannenbring. An Evaluation of Flow Shop Sequencing Heuristics. *Management Science*, 23:1174–1182, 1977.
- [4] M. Nawaz, E.E. Enscore Jr., and I. Ham. A Heuristic Algorithm for the m -Machine, n -Job Flow-Shop Sequencing Problem. *OMEGA*, 11:91–95, 1983.
- [5] E. Nowicki and C. Smutnicki. A Fast Tabu Search Algorithm for the Permutation Flow-Shop Problem. *European Journal of Operational Research*, 91:160–175, 1996.
- [6] F.A. Ogbu and D.K. Smith. Simulated Annealing for the Permutation Flowshop Problem. *OMEGA*, 19:64–67, 1990.
- [7] M. Widmer and A. Hertz. A New Heuristic Method for the Flow Shop Sequencing Problem. *European Journal of Operational Research*, 41:186–193, 1989.
- [8] I.H. Osman and C.N. Potts. Simulated Annealing for Permutation Flow-Shop Scheduling. *OMEGA*, 17:551–557, 1989.
- [9] C.R. Reeves and T. Yamada. Genetic Algorithms, Path Relinking and the Flowshop Sequencing Problem. *Evolutionary Computation Journal*, 6:45–60, 1998.
- [10] E. Taillard. Some Efficient Heuristic Methods for Flow Shop Sequencing. *European Journal of Operational Research*, 47:65–74, 1990.

- [11] E. Taillard. Benchmarks for Basic Scheduling Problems. *European Journal of Operational Research*, 64:278–285, 1993.
- [12] D. Whitley, T. Starkweather, and D. Fuquay. Scheduling Problems and Traveling Salesman: The Genetic Edge Recombination Operator. In: *Proceedings of the Third International Conference on Genetic Algorithms*, pages 133–140, Los Altos, Morgan Kaufmann, 1989.
- [13] T. Yamada and C.R. Reeves. Permutation Flowshop Scheduling by Genetic Local Search. In: *Proceedings of the 2nd IEE/IEEE International Conference on Genetic Algorithms in Engineering Systems*, pages 232–238, Glasgow, 1997.