

Metaheuristic Optimization via Memory and Evolution

Tabu Search and Scatter Search

OPERATIONS RESEARCH/COMPUTER SCIENCE INTERFACES SERIES

Series Editors

Professor Ramesh Sharda
Oklahoma State University

Prof. Dr. Stefan Voß
Universität Hamburg

Other published titles in the series:

- Greenberg / *A Computer-Assisted Analysis System for Mathematical Programming Models and Solutions: A User's Guide for ANALYZE*
- Greenberg / *Modeling by Object-Driven Linear Elemental Relations: A Users Guide for MODLER*
- Brown & Scherer / *Intelligent Scheduling Systems*
- Nash & Sofer / *The Impact of Emerging Technologies on Computer Science & Operations Research*
- Barth / *Logic-Based 0-1 Constraint Programming*
- Jones / *Visualization and Optimization*
- Barr, Helgason & Kennington / *Interfaces in Computer Science & Operations Research: Advances in Metaheuristics, Optimization, & Stochastic Modeling Technologies*
- Ellacott, Mason & Anderson / *Mathematics of Neural Networks: Models, Algorithms & Applications*
- Woodruff / *Advances in Computational & Stochastic Optimization, Logic Programming, and Heuristic Search*
- Klein / *Scheduling of Resource-Constrained Projects*
- Bierwirth / *Adaptive Search and the Management of Logistics Systems*
- Laguna & González-Velarde / *Computing Tools for Modeling, Optimization and Simulation*
- Stilman / *Linguistic Geometry: From Search to Construction*
- Sakawa / *Genetic Algorithms and Fuzzy Multiobjective Optimization*
- Ribeiro & Hansen / *Essays and Surveys in Metaheuristics*
- Holsapple, Jacob & Rao / *Business Modelling: Multidisciplinary Approaches — Economics, Operational and Information Systems Perspectives*
- Sleeker, Wentling & Cude / *Human Resource Development And Information Technology: Making Global Connections*
- Voß & Woodruff / *Optimization Software Class Libraries*
- Upadhyaya et al / *Mobile Computing: Implementing Pervasive Information and Communications Technologies*
- Reeves & Rowe / *Genetic Algorithms—Principles and Perspectives: A Guide to GA Theory*
- Bhargava & Ye / *Computational Modeling And Problem Solving In The Networked World: Interfaces in Computer Science & Operations Research*
- Woodruff / *Network Interdiction And Stochastic Integer Programming*
- Anandalingam & Raghavan / *Telecommunications Network Design And Management*
- Laguna & Martí / *Scatter Search: Methodology And Implementations In C*
- Gosavi / *Simulation-Based Optimization: Parametric Optimization Techniques and Reinforcement Learning*
- Koutsoukis & Mitra / *Decision Modelling And Information Systems: The Information Value Chain*
- Milano / *Constraint And Integer Programming: Toward a Unified Methodology*
- Wilson & Nuzzolo / *Schedule-Based Dynamic Transit Modeling: Theory and Applications*
- Golden, Raghavan & Wasil / *The Next Wave In Computing, Optimization, And Decision Technologies*

Metaheuristic Optimization via Memory and Evolution

Tabu Search and Scatter Search

edited by

César Rego and Bahram Alidaee



Kluwer Academic Publishers
Boston/Dordrecht/London

Distributors for North, Central and South America:

Kluwer Academic Publishers
101 Philip Drive
Assinippi Park
Norwell, Massachusetts 02061 USA
Telephone (781) 871-6600
Fax (781) 871-9045
E-Mail: kluwer@wkap.com

Distributors for all other countries:

Kluwer Academic Publishers Group
Post Office Box 322
3300 AH Dordrecht, THE NETHERLANDS
Telephone 31 786 576 000
Fax 31 786 576 254
E-mail: services@wkap.nl



Electronic Services <<http://www.wkap.nl>>

Library of Congress Cataloging-in-Publication Data

A C.I.P. Catalogue record for this book is available from the Library of Congress.

*Rego & Alidaee/ METAHEURISTIC OPTIMIZATION VIA MEMORY AND EVOLUTION:
Tabu Search and Scatter Search*

ISBN 1-4020-8134-0
ISBN 0-387-23667-8 (e-book)

Copyright © 2005 by Kluwer Academic Publishers.

All rights reserved. No part of this work may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, microfilming, recording, or otherwise, without the written permission from the Publisher, with the exception of any material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the work.

Permission for books published in Europe: permissions@wkap.nl

Permissions for books published in the United States of America: permissions@wkap.com

Printed on acid-free paper.

Printed in the United States of America.

CONTENTS

Foreword

vii

Part I: ADVANCES FOR NEW MODEL AND SOLUTION APPROACHES

- | | | |
|----|---|----|
| 1. | A Scatter Search Tutorial for Graph-Based Permutation Problems
<i>César Rego and Pedro Leão</i> | 1 |
| 2. | A Multistart Scatter Search Heuristic for Smooth NLP and
MINLP Problems
<i>Zsolt Ugray, Leon Lasdon, John C. Plummer, Fred Glover, Jim Kelly
and Rafael Martí</i> | 25 |
| 3. | Scatter Search Methods for the Covering Tour Problem
<i>Roberto Baldacci, Marco A. Boschetti, Vittorio Maniezzo and Marco
Zamboni</i> | 59 |
| 4. | Solution of the Sonet Ring Assignment Problem with
Capacity Constraints
<i>Roberto Aringhieri and Mauro Dell'Amico</i> | 93 |

Part II: ADVANCES FOR SOLVING CLASSICAL PROBLEMS

- | | | |
|----|--|-----|
| 5. | A Very Fast Tabu Search Algorithm for Job Shop Problem
<i>Józef Grabowski and Mieczyslaw Wodecki</i> | 117 |
| 6. | Tabu Search Heuristics for the Vehicle Routing Problem
<i>Jean-François Cordeau and Gilbert Laporte</i> | 145 |
| 7. | Some New Ideas in TS for Job Shop Scheduling
<i>Eugeniusz Nowicki and Czeslaw Smutnicki</i> | 165 |
| 8. | A Tabu Search Heuristic for the Uncapacitated Facility Location
Problem
<i>Minghe Sun</i> | 191 |
| 9. | Adaptive Memory Search Guidance for Satisfiability Problems
<i>Arne Løkketangen and Fred Glover</i> | 213 |

Part III: EXPERIMENTAL EVALUATIONS

- | | | |
|-----|---|-----|
| 10. | Lessons from Applying and Experimenting with Scatter Search
<i>Manuel Laguna and Vinícius A. Armentano</i> | 229 |
|-----|---|-----|

11. Tabu Search for Mixed-Integer Programming <i>João Pedro Pedroso</i>	247
12. Scatter Search vs. Genetic Algorithms: An Experimental Evaluation with Permutation Problems <i>Rafael Martí, Manuel Laguna and Vicente Campos</i>	263

Part IV: REVIEW OF RECENT DEVELOPMENTS

13. Parallel Computation, Co-operation, Tabu Search <i>Teodor Gabriel Crainic</i>	283
14. Using Group Theory to Construct and Characterize Metaheuristic Search Neighborhoods <i>Bruce W. Colletti and J. Wesley Barnes</i>	303
15. Logistics Management: An Opportunity for Metaheuristics <i>Helena R. Lourenço</i>	329

Part V: NEW PROCEDURAL DESIGNS

16. On the Integration of Metaheuristic Strategies in Constraint Programming <i>Mauro Dell'Amico and Andrea Lodi</i>	357
17. General Purpose Metrics for Solution Variety <i>David L. Woodruff</i>	373
18. Controlled Pool Maintenance for Metaheuristics <i>Peter Greistorfer and Stefan Voß</i>	387
19. Adaptive Memory Projection Methods for Integer Programming <i>Fred Glover</i>	425
20. RAMP: A New Metaheuristic Framework for Combinatorial Optimization <i>César Rego</i>	441
<i>Index</i>	461

Foreword

WHAT'S ALL THE COMMOTION ABOUT SCATTER SEARCH AND TABU SEARCH ANYWAY?

Fred Glover

*Leads School of Business, University of Colorado, Boulder, CO 80309-0419, USA,
fred.glover@colorado.edu*

Given an opportunity to dig into a good book, such as the one you presently hold in your hands, little can be more of a nuisance than to read a preface. Let me acknowledge at once that I certainly won't be offended if you don't read this one. (After all, I'll never know whether you did or not!) As my grandmother used to say, a preface to a book is like a speech before a banquet: it only delays enjoying what follows.

If you've bothered to read this far, at the least I owe you the courtesy of letting you know what to expect by reading farther. First, I'll quickly describe some features of the book that I'm delaying you from enjoying. Then I'll see I can offend as many of our mutual friends as possible (including you, to be sure) by suggesting that some popular conceptions of search strategies may be leading us down the wrong track. If at this point you haven't yet shredded the preface in the firm resolve to read the remainder of the book without further distraction, you will come to a proposal for an experiment to put my contentions to the test, and thereby to expose their underlying sense or nonsense. Finally, I'll wrap up by commenting on where we are headed in the grand scheme of things, as a way to remind you once again of what you've been missing by pausing to scan these preliminary pages.

What the book offers.

Tabu search and scatter search have come a long way from their origins, and the papers in this volume disclose some of the significant recent developments that are fueling advances in these areas. New contributions are documented for solving a varied collection of challenging problems, both those representing classical optimization models that have attracted the

attention of researchers for many years, and those representing quite recent models, that belong to forefront of modern optimization concerns. The volume additionally includes papers devoted to experimentation, exposing lessons learned from evaluating the performance of past and recent proposals. Still another section focuses on identifying currently established and emerging roles for tabu search and scatter search in realms such as parallel optimization and constraint programming, thereby affording a cross fertilization with other disciplines. And not least, the volume contains a collection of papers offering innovative proposals for new research directions.

This work represents the first time that tabu search and scatter search have been examined together in the same volume. Such a side-by-side examination is long overdue, particularly in view of the fact that the two approaches grew out of common beginnings, and share overlapping principles and perspectives. Today tabu search is typically applied from the perspective of adaptive memory concepts and strategies that it has introduced into the metaheuristic literature, while scatter search is primarily pursued within the context of an evolutionary approach, emphasizing processes concerned with generating and managing a population of solutions. Yet tabu search has always likewise embraced strategies for managing populations of solutions, notably within the setting of two of its basic types of intensification processes, the first concerned with saving and re-visiting elite solutions to explore their vicinity more thoroughly and the second concerned with analyzing these solutions to isolate critical features, such as variables that strongly and frequently receive particular values, as a basis for generating other solutions sharing these features.

In a complementary manner, scatter search has made use of adaptive memory, chiefly by virtue of incorporating improvement processes which, curiously enough, have only become recognized as relevant in other evolutionary approaches in relatively recent years. In the setting of these contrasting evolutionary procedures, the slowly dawning appreciation of the value of such improvement procedures has spawned the appearance of methods that currently go by the names of “hybrid” and “memetic” procedures. Characteristically, however, by its association with tabu search, scatter search is more often disposed to make use of improvement processes that embody adaptive memory, whereas other evolutionary approaches often still lag in bridging the gap to exploit such strategies.

Curious as it may seem, an eminent body of traditional wisdom inclines us to see little of interest in designs anchored to adaptive memory and associated strategies for exploiting it. The TS focus on adaptive memory admittedly

poses challenges that many researchers may prefer to sidestep. But, granting some liberty to speculation, these challenges may also help to define the frontiers of research into problem-solving processes that seek to incorporate abilities analogous to our own. Perhaps the human impulse to avoid acknowledging that our methods are still somewhat primitive steers us away from ideas that expose the yawning chasms of our ignorance. In any case, the blueprints drawn up for methods that do not venture far into the realms of memory and learning seem to capture the devotion of many search practitioners. From this standpoint, the papers of this volume represent a more daring collection than those found in many treatments of metaheuristics, by confronting important issues connected with the use of adaptive memory in search and seeking to draw inferences about experiments that may shed light on such issues.

Randomization: For Better or Worse¹

Apart from the issue of adaptive memory, a topic that conspicuously separates tabu search and scatter search from other approaches concerns the application of random choice. It is impossible to look very far in the metaheuristic literature without becoming aware that it is engaged in a love affair with randomization. Our “scientific” reports of experiments with nature reflect our fascination with the role of chance. When apparently chaotic fluctuations are brought under control by random perturbations, we seize upon the random element as the key, while downplaying the importance of attendant restrictions on the setting in which randomization operates. The diligently concealed message is that under appropriate controls, perturbation is effective for creating outcomes of desired forms. If the system and attendant controls are sufficiently constrained, perturbation works even when random, but a significant leap is required to conclude that randomization is preferred to intelligent design. (Instead of accentuating differences between workable and unworkable kinds of perturbation, in our quest to mold the universe to match our mystique we often portray the source of useful outcomes to be randomization in itself.)²

¹ This section and the next draw on portions of the reference F. Glover (2003) “Tabu Search – Uncharted Domains,” University of Colorado, Boulder.

² The attraction of randomization is particularly evident in the literature of evolutionary methods, but the theme that randomization is an essential underpinning of modern search methods is also echoed throughout many other segments of the metaheuristic literature. Perhaps this orientation relates to our natural disposition to take comfort in the notion that all can be left up to chance and everything will turn out for the best in the end. Or perhaps randomization has a seductive charm akin to the appeal of a certain kind of romantic

The orientation behind tabu search and scatter search evidently contrasts with this perspective. Many of the fundamental TS and SS strategies do not require randomization for their execution. While tabu search and scatter search principles do not exclude recourse to randomization, neither do they embrace it as essential. Occasionally such implementations are found where randomization is given a highly significant role, a fact that is consonant with the notion that a probing literature should entertain exceptions. However, as a rule, variants of TS and SS that incorporate randomized elements operate in a context where variation is tightly controlled by strategy.

There is of course a setting where randomization makes good sense. In a game against a shrewd opponent, it can be desirable to make sure one's behavior exhibits no demonstrable pattern, in order to avoid the chance that the pattern will be detected and turned to the adversary's advantage. In this case, recourse to randomization is a prudent policy. On the other hand, if we are entitled to presume that our search spaces are not possessed of a devious intelligence bent on thwarting our moves, random behavior fulfills no purpose comparable to its role in game playing. Instead, within the context of search, randomization seems more likely to be a means of outwitting the player who uses it. A policy of embarking on a series of random steps would seem one of the best ways of confounding the goal of detecting and exploiting structure.

We may grant that systematic search runs the risk of systematic blundering, something we are all susceptible to. But with appropriate monitoring, systematic search also affords an opportunity to isolate and rectify our blunders, and therefore to carry out more effective explorations. (Of course, if the search terrain itself is inherently random, no amount of systematization will do any good. The presence or absence of patterned design makes no difference when all is reduced to blind luck.)³

encounter – which seems somehow more captivating when marked by an element of capriciousness.

³ Even the statement that structured search has little value for a problem whose character is “essentially random” deserves qualification. A sorting problem may involve entirely random data, but may still be handled advantageously by a method that is highly systematic. (The “P vs. NP” distinction is relevant, but does not remove the need for additional qualification.)

An Experiment with Randomization.

I am tempted to pose a research challenge, as a way of testing the relevance of randomization in search. If a policy of random choice produces a useful sequence of responses, then we may presume that a different randomly generated sequence will likewise prove useful. On the other hand, not all sequences are apt to be equally good if the total number of moves is limited – a relevant consideration if it is important to reach good outcomes before the knell of doomsday, or at least before next summer’s vacation. When time matters, we may anticipate that not all randomly generated sequences will provide the same quality of performance.

This observation prompts the following experiment. To determine the relative utility of randomization, we might examine the outcomes of executing some number, say k , of randomly guided searches (each of limited duration) with the goal of finding a value of k such that at least one of the underlying choice sequences will lead to a solution of specified quality. We seek a value for our parameter that is not too large and that is applicable to a large portion of problems from a given class.

Now the evident question comes to mind. If we replace the k randomly generated choice sequences with k systematically generated sequences, can we obtain comparable or better results? Or can the replacement allow us to produce such results for a smaller value of k ? In accordance with the theme of diversification strategies used in TS and SS, the systematically generated sequences may be equipped to incorporate feedback, allowing a new sequence may be influenced by the outcomes of preceding sequences. (There is an issue of time invested in the systematic approach, so that the value of k must be adjusted to account for this.)

The outcome of such an experiment of course depends on our skills in designing systematic choice sequences, and also depends on the types of problems we confront. It must be acknowledged that the experiment I am advocating has been implicit in a variety of TS and SS implementations of the past. But there may be advantages to bringing the relevant factors into more explicit focus. Perhaps this will help us to better understand what is essential and what is superfluous when we speak of “intelligent strategy,” and to identify better instances of such strategy over time.

Where Are We Headed?

The papers of this book provide a series of landmarks along the way as we investigate and seek to better understand the elements of tabu search and scatter search that account for their successes in an astonishingly varied range of applications. The contributions of the chapters are diverse in scope, and are not uniform in the degree that they plumb or take advantage of fundamental principles underlying TS and SS. Collectively, however, they offer a useful glimpse of issues that deserve to be set in sharper perspective, and that move us farther along the way toward dealing with problems whose size and complexity pose key challenges to the optimization methods of tomorrow.

Acknowledgment and Disclaimer

This material is based upon work supported by the National Science Foundation under Grant No. 0118305. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Chapter 1

A SCATTER SEARCH TUTORIAL FOR GRAPH-BASED PERMUTATION PROBLEMS

César Rego¹ and Pedro Leão²

¹*Hearin Center for Enterprise Science, School of Business Administration, University of Mississippi, University, MS 38677, USA, crego@bus.olemiss.edu;*

²*Universidade Portucalense, Departamento de Informática, Rua Dr. António Bernardino de Almeida 541-619, 4200, Porto, Portugal, pedro@upt.pt.*

Abstract: Scatter search is an evolutionary method that has proved highly effective in solving several classes of non-linear and combinatorial optimization problems. Proposed early 1970s as a primal counterpart to the dual surrogate constraint relaxation methods, scatter search has recently found a variety of applications in a metaheuristic context. Because both surrogate constraint methods and scatter search incorporate strategic principles that are shared with certain components of tabu search methods, scatter search provides a natural evolutionary framework for adaptive memory programming. The aim of this paper is to illustrate how scatter search can be effectively used for the solution of general permutation problems that involve the determination of optimal cycles (or circuits) in graph theory and combinatorial optimization. In evidence of the value of this method in solving constrained optimization problems, we identify a general design for solving vehicle routing problems that sets our approach apart from other evolutionary algorithms that have been proposed for various classes of this problem.

Keywords: Metaheuristics, Scatter Search, Combinatorial Optimization, Permutation Problems, Vehicle Routing Problems

1. Introduction

Scatter search is an evolutionary method proposed by Glover (1977) as a primal counterpart to the dual approaches called surrogate constraint methods, which were introduced as mathematical relaxation techniques for discrete optimization problems (Glover 1965). As opposed to eliminating constraints by plugging them into the objective function as in Lagrangean

relaxations, for example, surrogate relaxations have the goal of generating new constraints that may stand in for the original constraints. The method is based on the principle of capturing relevant information contained in individual constraints and integrating it into new surrogate constraints as a way to generate composite decision rules and new trial solutions. Scatter search combines vectors of solutions in place of the surrogate constraint approach of combining vectors of constraints, and likewise is organized to capture information not contained separately in the original vectors. Also, in common with surrogate constraint methods it is organized to take advantage of auxiliary heuristic solution methods to evaluate the combinations produced and generate new vectors. As any evolutionary procedure, the method maintains a population of solutions that evolves in successive generations.

A number of algorithms based on the scatter search approach have recently been proposed for various combinatorial problems (see Kelly, Rangaswamy and Xu 1996, Fleurent et al. 1996, Cung et al. 1999, Laguna, Martí and Campos 1999, Campos et al. 1999, Glover, Løkketangen and Woodruff 1999, Atan and Secomandi 1999, Laguna, Lourenço and Martí 2000, Xu, Chiu and Glover 2000).

In this study we provide a general scatter search design for problems dealing with the optimization of cycles (or circuits) on graphs. The traveling salesman problem (TSP) which consists of finding the shortest Hamiltonian *cycle* (or *circuit* in the asymmetric case) is a typical example of this class of problems. It is well-known that the TSP is NP-Complete so that efficient heuristic methods are required to provide high quality solutions for large problem instances (see Johnson and McGeoch 1997, Rego 1998). A direct extension of the TSP is the vehicle routing problem (VRP) where side constraints force the creation of multiple Hamiltonian cycles (*routes*) starting and ending at a central node (representing a hub or depot). Both TSP and VRP problems provide a general model for a wide range of practical applications and are central in the fields of transportation, distribution and logistics (see Reinelt 1994, Laporte and Osman 1995). Empirical studies have shown that the VRP is significantly harder to solve than TSPs of similar sizes (see Gendreau, Hertz and Laporte 1994, Rochat and Taillard 1995, Rego 1998, 2000, and Toth and Vigo 2003, for some of the current best heuristic algorithms for the VRP). Because of its theoretical and practical relevance we use the vehicle routing problem to illustrate the various procedures involved in the scatter search template.

Also, while other tutorials on scatter search exist (see, e.g. the articles by Laguna 2001, and Glover, Laguna and Martí 2001), there have been no exposition of the approach that disclose its operation within the setting of routing.

The remainder of this paper is organized as follows. An overview of the scatter search method is presented in Section 2. Section 3 defines the vehicle routing problem and presents formulations that are relevant for the context of the paper. Section 4 illustrates the design of the scatter search procedure for an instance of the problem. Section 5 summarizes and discusses possible generalizations of the proposed template.

2. Scatter Search Overview

Scatter search operates on a set of reference solutions to generate new solutions by weighted linear combinations of structured subsets of solutions. The reference set is required to be made up of high-quality and diverse solutions and the goal is to produce weighted centers of selected subregions that project these centers into regions of the solution space to be explored by auxiliary heuristic procedures. Depending on whether convex or nonconvex combinations are used, the projected regions can be respectively internal or external to the selected subregions. For problems where vector components are required to be integer a rounding process is used to yield integer values for such components. Rounding can be achieved either by a generalized rounding method or iteratively, using updating to account for conditional dependencies that can modify the rounding options. Regardless the type of combinations employed, the projected regions are not required to be feasible so that the auxiliary heuristic procedures are usually designed to incorporate a double function of mapping an infeasible point to a feasible trial solution and then to transform this solution into one or more trial solutions. (Although, the auxiliary heuristic commonly includes the function of restoring feasibility, this is not an absolute requirement since scatter search can be allowed to operate in the infeasible solution space.) From the implementation standpoint the scatter search method can be structured to consist of the following subroutines:

Diversification Generation Method - Used to generate diverse trial solutions from one or more arbitrary *seed solutions* used to initiate the method.

Improvement Method - Transform a trial solution into one or more enhanced trial solutions. (If no improvement occurs for a given trial solution, the enhanced solution is considered to be the same as the one submitted for improvement.)

Reference Set Update Method - Creates and maintains a set of *reference solutions* consisting of the best according to the criteria under consideration. The goal is to ensure diversity while keeping high-quality solutions as measured by the objective function.

Subset Generation Method - Generates subsets of the reference set as a basis for creating combined solutions.

Solution Combination Method - Uses weighted structured combinations to transform each subset of solutions produced by the subset generation method into one or more combined solutions.

A general template for a scatter search algorithm can be organized in two phases outlined as follows.

- **Initial Phase**

 Diversification Generation Method

 Improvement Method

 Reference Set Update Method

 Repeat this initial phase until producing a desirable level of high-quality and diverse solutions.

- **Scatter Search Phase**

 Subset Generation Method

 Solution Combination Method

 Improvement Method

 Reference Set Update Method

 Repeat this scatter search phase while the reference set converges or until a specified cutoff limit on the total number of iterations.

3. The Vehicle Routing Problem

The vehicle routing problem is a classic in Combinatorial Optimization. To establish some basic notation, and to set the stage for subsequent illustrations, we provide a formal definition of the vehicle routing problem in this section. We also present two mathematical formulations which are relevant for the scatter search design introduced in the next section.

3.1 Problem Definition

In graph theory terms the classical VRP can be defined as follows. Let $G = (V, A)$ be a graph where $V = \{v_0, v_1, \dots, v_n\}$ is a vertex set, and $A = \{(v_i, v_j) \mid v_i, v_j \in V, i \neq j\}$ is an arc set. Vertex v_0 denotes a *depot*, where a fleet of m identical vehicles of capacity Q are based, and the remaining vertices $V' = V \setminus \{v_0\}$ represent n *cities* (or client locations). A nonnegative *cost* or distance matrix $C = (c_{ij})$ which satisfies the triangle inequality ($c_{ij} \leq c_{ik} + c_{kj}$) is defined on A . When $c_{ij} = c_{ji}$ for all $(v_i, v_j) \in A$ the problem is said to be symmetric and it is then common to replace A with the edge set $E = \{(v_i, v_j) \mid v_i, v_j \in V, i \neq j\}$. It is assumed that $m \in [m, \bar{m}]$ with

$\underline{m} = 1$ and $\bar{m} = n - 1$. The value of m can be a decision variable or can be fixed depending on the application.

Vehicles make *collections* or deliveries but not both. With each vertex v_i is associated a quantity q_i ($q_0 = 0$) of some goods to be delivered by a vehicle. The VRP consists of determining a set of m vehicle routes of minimal total cost, starting and ending at a depot v_0 , such that every vertex in $v_i \in V'$ is visited exactly once by one vehicle and the total quantity assigned to each route does not exceed the capacity Q of the vehicle which services the route.

We define a solution for the VRP as a set of m routes, $S = \{R_1, \dots, R_m\}$, $R_k = (v_0, v_{k_1}, v_{k_2}, \dots, v_0)$ where R_k is an ordered set representing consecutive vertices in the route k . Consequently, we denote $v_i \in R_k$ if v_i is component of R_k and similarly $(v_i, v_j) \in R_k$ if v_i, v_j are two consecutive vertices in R_k . Finally, the cost of a solution S is defined as $C(S) = \sum_{1 \leq k \leq m} \sum_{(i,j) \in R_k} c_{ij}$.

3.2 Vehicle Flow Formulation

$$\min \sum_{i=0}^n \sum_{j=0}^n c_{ij} \sum_{k=1}^m x_{ij}^k$$

s.t.

$$\sum_{i=1}^n q_i y_i^k \leq Q_k, \quad k = 1, \dots, m \quad (1)$$

$$\sum_{k=1}^m y_i^k = \begin{cases} m, & i = 0 \\ 1, & i = 1, \dots, n \end{cases} \quad (2)$$

$$\sum_{i=0}^n x_{ij}^k = \sum_{i=0}^n x_{ji}^k = y_i^k, \quad \begin{matrix} j = 0, \dots, n \\ k = 1, \dots, m \end{matrix} \quad (3)$$

$$\sum_{i,j \in U} x_{ij}^k \leq |U| - 1 \quad \begin{matrix} 1 \leq |U| \leq n-1 \\ i = 0, \dots, n \\ k = 1, \dots, m \end{matrix} \quad (4)$$

$$y_i^k, x_{ij}^k \in \{0, 1\} \quad \begin{matrix} i, j = 1, \dots, n \\ k = 1, \dots, m \end{matrix} \quad (5)$$

In this formulation (Fisher and Jaikumar 1981), variables x_{ij}^k indicates whether or not (v_i, v_j) is an edge of route k in the optimal solution, and

similarly y_i^k specifies whether route k contains vertex v_i . Constraints (1) guarantee that vehicle capacity is not exceeded; constraints (2) ensure that each city is visited by exactly one vehicle and m vehicles visit the depot; constraints (3) ensure that every city is entered and left by the same vehicle; subtour elimination are specified by constraints (4); and equations (5) are the integrality constraints.

This formulation contains two well-known problems: the generalized assignment problem (GAP) specified by constraints (1), (2) and (5); and the traveling salesman problem (TSP). The TSP results when the y_i^k 's are fixed to satisfy the GAP constraints for a given k , whereupon constraints (3) and (5) define a TSP for vehicle k . Specifically, the GAP is the subproblem responsible for assigning clients to vehicle routes, and the sequence in which clients are visited by a vehicle is determined by the TSP solution over each individual route.

Laporte, Nobert, and Desrochers (1985) show that a two-index formulation can be derived from a three-index formulation by aggregating all x_{ij}^k variable into a single variable x_{ij} which does not refer specifically to a particular route, but simply indicates whether or not an arc (v_i, v_j) is in the optimal solution. For the symmetrical case the value of the x_{ij} variable indicates the number of times (0,1, or 2) that the corresponding edge is used in the optimal solution. We will see later that this vehicle-flow formulation and the index-reduction concept proves useful in several parts of our scatter search model.

3.3 A Permutation-Based Formulation

In combinatorial optimization it is usually useful to distinguish two classes of problems according to whether the objective is to find (1) an optimal *combination* of discrete objects, or (2) an optimal *permutation* of objects. In the first case, a solution is a non-ordered set while in the second a solution is a ordered set (or a sequence). For instance, the 0-1 knapsack problem and the set covering problem are typical examples of combination problems where a solution can be interpreted to consist simply of a set of elements. Permutation problems on the other hand, are exemplified by job scheduling, traveling salesman, and vehicle routing problems where different arrangements of the same set represent different solutions (allowing for a sequence to be considered equivalent to its reverse ordering in the case of symmetric problems). The differentiation of these two types of combinatorial problems is particularly relevant because the specialized methods that apply to these problems, especially those methods based on graph theoretic representations, differ dramatically in the two cases. We will address to these features later on when describing the proposed scatter search template.

The VRP, which is the primary focus of this tutorial, seeks to determine an optimal permutation according to the following formulation:

$$\min \sum_{k=1}^m \sum_{i=0}^{n_k} c_{\pi_i^k, \pi_{i+1}^k}$$

s.t.

$$\sum_{i=1}^{n_k} q_{\pi_i^k} \leq Q_k, \quad k = 1, \dots, m \quad (1)$$

$$\sum_{k=1}^m n_k \leq n \quad (2)$$

Here, a solution is defined by the permutation $\{\pi_1^1, \dots, \pi_{n_1}^1, \dots, \pi_1^m, \dots, \pi_{n_m}^m\}$ of vertices $\{v_1, \dots, v_n\}$. Vertex v_0 is implicitly represented by inserting it at the beginning and end of each subsequence k of the permutation (and may be expressed by “dummy elements” π_0^k and $\pi_{n_k+1}^k$). The permutation is partitioned into subsequences by indexes n_k and may be interpreted to consist of m separate routes, where the sequence of nodes on a given route identifies the order in which clients (vertices) are visited by a vehicle assigned to that route.

4. Scatter Search Template

Consider the following VRP instance with $n=14$, $Q=30$, $q = (q_i) (i = 0, \dots, 14) = (0, 6, 20, 8, 9, 10, 8, 7, 5, 5, 4, 3, 4, 7)$, $m \in [1, 14]$, and a cost matrix $C = (c_{ij}) (i, j = 1, \dots, 14, i < j)$ defined as follows:

$$C = \begin{bmatrix} 0 & 7.85 & 10.82 & 8.18 & 9.71 & 4.53 & 3.13 & 5.11 & 6.40 & 7.54 & 6.40 & 9.30 & 8.51 & 4.61 & 7.96 \\ 0 & 14.40 & 4.36 & 4.19 & 9.41 & 9.39 & 2.77 & 1.57 & 1.12 & 2.82 & 1.73 & 8.98 & 12.04 & 15.74 \\ 0 & 10.89 & 12.52 & 15.24 & 13.62 & 12.48 & 13.04 & 14.96 & 15.02 & 16.10 & 19.19 & 13.20 & 13.54 \\ 0 & 1.84 & 11.47 & 10.80 & 4.40 & 3.78 & 5.34 & 6.48 & 5.76 & 12.59 & 12.79 & 15.88 \\ 0 & 12.60 & 12.12 & 5.30 & 4.30 & 5.30 & 6.82 & 5.12 & 13.05 & 14.30 & 17.54 \\ 0 & 1.71 & 7.31 & 8.46 & 8.60 & 6.85 & 10.26 & 5.02 & 4.15 & 8.01 \\ 0 & 6.96 & 8.22 & 8.75 & 7.14 & 10.48 & 6.60 & 2.94 & 6.91 \\ 0 & 1.30 & 2.72 & 2.60 & 4.34 & 8.27 & 9.44 & 13.04 \\ 0 & 1.92 & 2.70 & 3.26 & 8.85 & 10.74 & 14.35 \\ 0 & 1.81 & 1.77 & 7.88 & 11.50 & 15.30 \\ 0 & 3.40 & 6.24 & 9.98 & 13.86 \\ 0 & 8.95 & 13.26 & 17.07 \\ 0 & 9.14 & 12.87 \\ 0 & 3.97 \\ 0 & & \end{bmatrix}$$

A viewgraph of the clients' spatial distribution and the corresponding vertex coordinates that gave rise to the matrix C is provided in the appendix. We will use this special representation of the problem to illustrate different procedures used in the scatter search design.

Now, we can proceed to the illustration of the scatter search design using the defined VRP instance as an example of a permutation problem. We first describe in the context of the VRP each of the methods considered in the general scatter search template. Then, we present the various procedures integrated in the overall algorithm.

- **Initial Phase**

Diversification Generation Method

Scatter search starts with an initial set of trial solutions which are required to be different, thus the use of a systematic procedure to generate these solutions is appropriate. Regarding the VRP as permutation problem we consider the generator of combinatorial objects described in Glover (1997), which operates as follows. A trial permutation P is used as a seed to generate subsequent permutations. Define the subsequence $P(h:s)$ where s is a positive integer between 1 and h , so that $P(h:s) = (s, s+h, s+2h, \dots, s+rh)$, where r is the largest nonnegative integer such that $s+rh \leq n$. Finally, define the permutation $P(h)$ for $h < n$, to be $P(h) = (P(h:h), P(h:h-1), \dots, P(h:1))$. In the VRP context we consider permutations in n -vectors where components are all vertices $v_i \in V \setminus \{v_0\}$. Consider for illustration $h = 4$, and a seed permutation $P = \{1,2,3,4,5,6,7,8,9,10,11,12,13,14\}$ given by the sequence of clients ordered by their indices. The recursive application of $P(4:s)$ for $s = 4, \dots, 1$ results the subsequences:

$P = \{4,8,12\}$, $P = \{3,7,11\}$, $P = \{2,6,10,14\}$ and $P = \{1,5,9,13\}$, hence $P(4) = \{4,8,12,3,7,11,2,6,10,14,1,5,9,13\}$. For illustration purposes we consider the set of initial vertices assignments A_h to vehicle routes derived from permutations $P(h)$ ($h = 1, \dots, 10$) where each cluster of vertices in a route is obtained by successively assigning a vertex $v_i (i \in P(h))$ to a route R_{h_k} (initially $k=1$) until the cumulative quantity $Q_k = \sum_{v_i \in R_{h_k}} q_i$ does not exceed Q with the insertion of a new vertex v_{k_j} . As soon as such a cutoff limit is attained a new assignment is created by incrementing k by one unit, and the process goes on until all vertices have been assigned. Table 1.1 shows the assignment A_4 derived from permutation $P(4)$, where shadowed cells indicate the Q_k value associated with the insertion of vertex v_i in a route R_k .

In fact, the result obtained can be viewed as a generalized assignment process which does not rely on the order in which clients are visited, though it ensures that all the initial solutions that can be created are feasible and different (since they derive from distinct permutations). Vehicle routes can

thus be determined by using any traveling salesman algorithm on the subgraph defined by the previous assignments.

Table 1.1. Sequential assignment of clients to vehicle routes

P(4)	i	4	8	12	3	7	11	2	6	10	14	1	5	9	13
R _k	q _t	8	7	3	9	8	4	20	10	5	7	6	9	5	4
R ₁	8	15	18	27											
R ₂					8	12									
R ₃							20	30							
R ₄									5	12	18	27			
R ₅													5	9	

For illustration, we consider the standard *2-exchange* procedure (Lin 1965) to find a *2-optimal* TSP tour for each individual route. The 2-optimal procedure is a local search improvement method, hence an initial feasible TSP solution for each route R_k is required to start the method. A straightforward method to create this solution can be obtained by successively linking vertices in the order they appear in the permutation and attaching the initial and ending vertices to the depot. Then, the method proceeds by replacing two edges (v_i, \bar{v}_i) and (v_j, \bar{v}_j) by two others (v_i, v_j) and (\bar{v}_i, \bar{v}_j) where \bar{v} denotes the successor of v in a given orientation of the route. Hence, in order to maintain the feasible orientation of the route, the subpath (\bar{v}_i, \dots, v_j) needs to be reversed, so that the subpath $(v_i, \bar{v}_i, \dots, v_j, \bar{v}_j)$ becomes $(v_i, v_j, \dots, \bar{v}_i, \bar{v}_j)$. For convenience, denote the c_{ij} values by $c(v_i, v_j)$ so that the solution cost change produced by a 2-exchange move can be expressed as $\Delta_{ij} = c(v_i, v_j) + c(\bar{v}_i, \bar{v}_j) - c(v_i, \bar{v}_i) - c(v_j, \bar{v}_j)$. A *2-optimal* (or *2-opt*) solution is obtained by iteratively applying 2-exchange moves until no possible move yield a negative Δ value.

As the purpose of the diversification generation method is to generate diverse solutions rather than high quality solutions, it is convenient to make the method fast, therefore we have adopted a *first-improvement* (as opposed to a *best-improvement*) strategy for the TSP heuristic.

Table 1.2 shows the four iterations carried out to create the solution S_4 . Specifically, the first column shows the initial trial solution T_4 created from the assignment A_4 (defined in Table 1.1) and the final solution S_4 . Figures within parenthesis indicate the iteration number. The second column shows vertices v_i, v_j that define the move from one iteration to another. (Note that a 2-exchange move is completely identified by vertices v_i, v_j since the two other vertices are necessarily their successors, respectively \bar{v}_i and \bar{v}_j .) The remaining columns show the routes in each solution, the solution cost, and the value associated with the move that has led to the solution shown in the next row. Boldfaced numbers indicate the changes carried out by the

corresponding move. The result of these changes are illustrated in Figure 1.1, which depicts the initial trial solution T_4 and the final solution S_4 .

Table 1.2. Application of 2-opt procedure to T_4 trial solution

Solution	Move	R_{1_i}	R_{2_i}	R_{3_i}	R_{4_i}	R_{5_i}	Cost	Δ_{ij}
$T_4^{(1)}$	v_{0_i}, v_{12_i}	4 8 12 3	7 11	2 6	10 14 15	9 13	163.54	-11.94
$T_4^{(2)}$	v_{10_i}, v_{1_i}	12 8 4 3	7 11	2 6	10 14 15	9 13	151.60	-12.45
$T_4^{(3)}$	v_{j_i}, v_{5_i}	12 8 4 3	7 11	2 6	10 1 14 5	9 13	139.15	-2.90
$T_4^{(4)}$	v_{0_i}, v_{1_i}	12 8 4 3	7 11	2 6	10 1 5 14	9 13	136.25	-1.10
S_4		12 8 4 3	7 11	2 6	1 10 5 14	9 13	135.15	

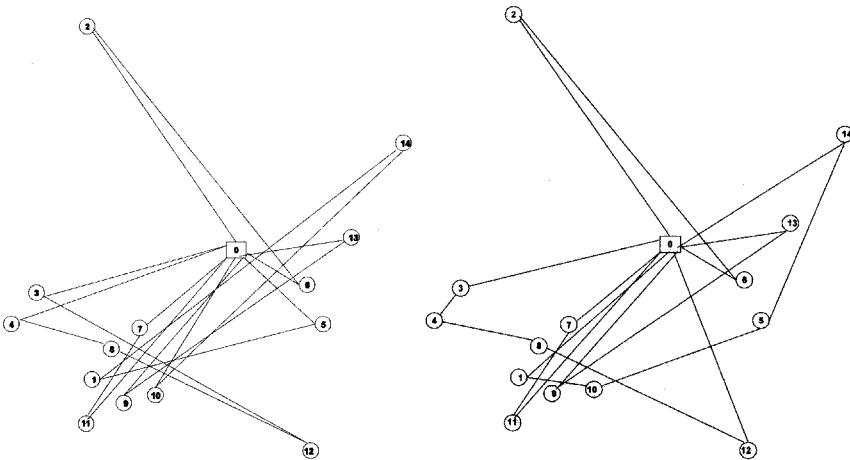


Figure 1.1. Solutions T_4 (left-hand side) and S_4 (right-hand side)

The shape of the solutions in the figure clearly show that routes R_{1_i} and R_{4_i} were significantly improved. Repeating the process for the rest of the permutations we obtain the ten solutions produced by the diversification generation method as illustrated in Table 1.3.

Improvement Method

The improvement method used in the initial phase may or may not be the same method used in the scatter search phase. This decision usually depends on the context and on the search strategy one might want to implement. For the purpose of this tutorial we use the same procedure in both phases, though in the scatter search phase the method is required to deal with some additional features.

Table 1.3. Trial solutions (T_k) and final solutions (C_k) produced by the diversification generation method

Solution	Routes	Cost
T1	0 1 2 0 3 4 5 0 6 7 8 9 0 10 11 12 13 14 0	120.90
S1	0 1 2 0 3 4 5 0 6 9 8 7 0 11 10 12 14 13 0	120.83
T2	0 2 4 0 6 8 10 12 0 14 1 3 0 5 7 9 11 13 0	132.28
S2	0 2 4 0 6 12 10 8 0 14 3 1 0 7 11 9 5 13 0	122.74
T3	0 3 6 9 12 0 2 5 0 8 11 14 1 0 4 7 10 13 0	157.24
S3	0 6 12 9 3 0 2 5 0 8 1 11 14 0 7 4 10 13 0	128.27
T4	0 4 8 12 3 0 7 11 0 2 6 0 10 14 1 5 0 9 13 0	163.54
S4	0 12 8 4 3 0 7 11 0 2 6 0 1 10 5 14 0 9 13 0	135.15
T5	0 5 10 4 9 0 14 3 8 13 0 2 7 0 12 1 6 11 0	149.08
S5	0 5 10 9 4 0 8 3 14 13 0 2 7 0 1 11 12 6 0	119.50
T6	0 6 12 5 11 0 4 10 3 9 0 2 8 0 14 1 7 13 0	140.97
S6	0 6 5 12 11 0 3 4 9 10 0 2 8 0 14 13 1 7 0	113.74
T7	0 7 14 6 13 0 5 12 4 11 0 3 10 0 2 9 0 1 8 0	139.83
S7	0 7 6 13 14 0 5 12 11 4 0 3 10 0 2 9 0 1 8 0	130.47
T8	0 8 7 6 0 14 5 13 4 0 12 3 11 0 2 10 0 1 9 0	146.83
S8	0 7 8 6 0 14 13 5 4 0 12 11 3 0 2 10 0 1 9 0	136.29
T9	0 9 8 7 6 0 5 14 4 13 0 3 12 0 2 11 1 0 10 0	148.42
S9	0 7 8 9 6 0 14 13 5 4 0 3 12 0 2 11 1 0 10 0	137.16
T10	0 10 9 8 7 0 6 5 4 0 14 3 13 0 2 12 1 0 11 0	148.65
S10	0 10 9 8 7 0 6 5 4 0 3 14 13 0 2 1 12 0 11 0	135.92

We consider an iterative improvement method based on local search. The method is designed to directly operate on the problem graph using a very straightforward *neighborhood structure* consisting of removing a vertex from its current position and inserting it between two other vertices. Specifically, denoting respectively by v and \bar{v} the predecessor and successor of a vertex v , the insertion of a vertex v_i between vertices v_p and v_q operates by inserting edges (v_i, \bar{v}_i) , (v_p, v_i) , (v_i, v_q) , and by removing edges (v_i, v_i) , (v_i, \bar{v}_i) and (v_p, \bar{v}_p) , where $v_q = \bar{v}_p$. Hence, the solution cost change is given by $\Delta_{ip} = c(v_i, \bar{v}_i) + c(v_p, v_i) + c(v_i, v_q) - c(v_i, v_i) - c(v_i, \bar{v}_i) - c(v_p, \bar{v}_p)$. Applying the procedure on each S_k solution (in Table 1.3) we obtain the corresponding improved solutions reported in Table 1.4. (A graphic illustration of the method is shown in the scatter search phase where some additional features of the method are included.)

Reference Set Update Method

This method is used to create and maintain a set of reference solutions. As in any evolutionary (population-based) method, a set of solutions (population of individuals) containing high evaluation combinations of attributes replaces less promising solutions at each iteration (generation) of the method in order to enhance the quality of the population.

Table 1.4. Improved Solutions

Solution	Routes	Cost
S1	0 2 0 3 4 5 0 6 9 8 7 0 1 11 10 12 14 13 0	109.67
S2	0 2 0 3 4 1 8 0 6 5 13 14 0 7 11 9 10 12 0	92.51
S3	0 6 5 12 9 0 2 0 8 1 11 13 14 0 7 3 4 10 0	106.37
S4	0 8 4 3 0 7 1 11 9 10 0 2 0 12 5 6 0 13 14 0	96.84
S5	0 5 12 10 9 4 0 7 3 14 13 0 2 0 8 1 11 6 0	111.52
S6	0 6 5 0 3 4 9 10 12 0 2 8 0 14 13 11 1 7 0	106.30
S7	0 6 5 13 14 0 3 4 11 10 12 0 2 0 9 1 8 7 0	92.48
S8	0 7 8 1 9 0 14 13 5 6 0 12 10 11 4 3 0 2 0	92.48
S9	0 7 8 0 14 13 5 6 0 3 4 9 10 12 0 2 1 1 1 0	102.11
S10	0 10 9 11 8 7 0 6 5 0 4 3 14 13 0 2 1 12 0	107.74

In genetic algorithms, for example, the updating process relies on randomized selection rules which select individuals according to their relative fitness value. In scatter search the updating process relies on the use of memory and is confined to maintain a good balance between *intensification* and *diversification* of the solution process. In advanced forms of scatter search reference solutions are selected based on the use of memory which operates by reference to different dimensions as defined in tabu search. Depending on the context and the search strategy, different types of memory can be used. The way they are integrated to achieve both intensification and diversification is generically called *adaptive memory programming*. (See Glover and Laguna 1997 for a detailed explanation of various forms and uses of memory within search processes.) For the purpose of this tutorial we use a simple rule to update the set of reference solutions, where intensification is achieved by the selection of high-quality solutions (in terms of the objective function value) and diversification is basically induced by including diverse solutions from the current candidate set CS . Thus the reference set RS can be defined by two distinct subsets B and D , representing respectively the subsets of high-quality and diverse solutions, hence $RS = B \cup D$. Also, in the context of our example the terms “highest evaluated solution” and “best solution” are interchangeable and refer to the solution that best fits the evaluation criterion under consideration.

Consider the candidate set $CS = \{S_1, \dots, S_{10}\}$ defined by the improved solutions, and a reference set of size $|RS| = 6$ where $|B| = |D| = 3$.¹ Before

¹ The cardinality of B and D does not need to be identical and can vary during the search. For instance, relatively higher values of B (D) can be appropriate during a phase that is more strongly characterized by an intensification (diversification) emphasis. Also different schemes can be chosen to implement these variations. A dynamic variation of these values can be implemented by a perturbation scheme conducted strategically rather than randomly. For example, a strategic oscillation can be

proceeding to the creation of the reference set we first need to eliminate possible repeated solutions in the current set of trial solutions of Table 1.4. We can see that S_7 and S_8 represent the same solution, so we drop S_8 from the solution candidate set, making $CS = CS \setminus \{S_8\}$. Now, to create RS we start by selecting the three best solutions S_7 , S_2 and S_4 in CS to generate B , then the set D of diverse solutions is generated by successively selecting the solution which mostly differs from the ones currently belonging to RS . As a diversity measure we define $d_{ij} = |(S_i \cup S_j) \setminus (S_i \cap S_j)|$ as the distance between solutions S_i and S_j , which gives the number of edges by which the two solutions differ from each other. For example, solution S_3 contains 6 edges $(1,8), (3,7), (4,10), (9,0), (9,12), (11,13)$, which are not in solution S_4 , and solution S_4 has 7 edges $(1,7), (3,0), (4,8), (9,10), (9,11), (12,0), (13,0)$, which are not in the solution S_3 . Hence the distance between the two solutions is 13. Table 1.5 shows d_{ij} values for each pair of solutions $S_i \in RS$ and $S_j \in CS$.

Candidate solutions are included in RS according to the *Maxmin* criterion which maximizes the minimum distance of each candidate solution to all the solutions currently in the reference set. The method starts with $RS = B$ and at each step RS is extended with a solution $S_j \in CS$, to be $RS = RS \cup \{S_j\}$, and consequently CS is reduced to $CS = CS \setminus \{S_j\}$. Then the distance of solution S_j to every solution currently in the reference set is computed to make possible the selection of a new candidate solution according to the *Maxmin* criterion. More formally, the selection of a candidate solution is given by $S_j = \arg \max_{i=1, \dots, |RS|} \min_{j=1, \dots, |CS|} \{d_{ij}\}$.

Table 1.5. Distances between solutions

Reference set (RS)	Candidate solutions (CS)					
	S_1	S_3	S_5	S_6	S_9	S_{10}
S_7	16	16	22	16	12	20
S_2	20	16	18	12	10	18
S_4	19	13	17	11	13	15
Minimum distance	16	13	17	11	10	15
S_5	18	16		16	18	18
Minimum distance	16	13		11	10	15
S_1		22		18	16	18
Minimum distance		13		11	10	15

The process is repeated until $|RS|$ is achieved. In the table, boldfaced figures represent the maxmin values obtained at each step of the method.

implemented by using critical event memory as an indicator of the order of magnitude of the relative variations. Note that since the cardinality of subsets B and D are complementary in relation to the RS size and the decision can be uniquely based on the variation of either one.

This results in an initial reference set formed by solutions S_7 , S_2 , S_4 , S_5 , S_1 , S_{10} . For convenience, we reorder the solution indexes by setting $RS = \{S_1, S_2, S_3, S_4, S_5, S_6\}$ as illustrated in Table 1.6.

Table 1.6. Initial reference set.

Solution	Routes	Cost
S1	0 6 5 13 14 0 3 4 11 10 12 0 2 0 9 1 8 7 0	92.48
S2	0 2 0 3 4 1 8 0 6 5 13 14 0 7 11 9 10 12 0	92.51
S3	0 8 4 3 0 7 1 11 9 10 0 2 0 12 5 6 0 13 14 0	96.84
S4	0 5 12 10 9 4 0 7 3 14 13 0 2 0 8 1 11 6 0	111.52
S5	0 2 0 3 4 5 0 6 9 8 7 0 1 11 10 12 14 13 0	109.67
S6	0 10 9 11 8 7 0 6 5 0 4 3 14 13 0 2 1 12 0	107.74

It is important to note that a balance between intensification and diversification is achieved by an evaluation criterion that causes the highest-evaluated solutions considered for the reference to include qualities other than a “good” (small) objective function value. Solution S_9 with a cost of 102.11 is by-passed in favor of other solutions that will add more diversity to the set. As indicated in Table 1.5, the distance between S_9 to solutions in the reference set are relatively lower, making this solution unattractive from the standpoint of diversification.

- **Scatter Search Phase**

Subset Generation Method

This method consists of generating subsets of reference solutions to create structured combinations in the next step. The method is typically designed to organize subsets of solutions to cover different promising regions of the solution space. In a spatial representation, the convex-hull of each subset delimits the solution space in subregions containing all possible convex combinations of solutions in the subset. In order to achieve a suitable intensification and diversification of the solution space, three types of subsets are required to be organized:

1. subsets containing only solutions in B ,
2. subsets with only solutions in D , and
3. subsets mixing in solutions in B and D in different proportions.

Subsets defined by solutions of type 1 are conceived to intensify the search in regions of high-quality solutions while subsets of type 2 are created to diversify the search to unexplored regions. Finally, subsets of type 3 integrate both high-quality and diverse solutions with the aim of exploiting solutions across these two types of subregions.

Again, adaptive memory can be used to appropriately define combined rules for clustering elements in the various types of subsets. This has the

advantage of incorporating additional information about the search space and problem context.

Since the use of sophisticated memory features is beyond the scope of this study, we consider a simpler yet systematic procedure to generate the following types of subsets:

1. All 2-element subsets.
2. 3-element subsets derived from two element subsets by augmenting each 2-element subset to include the best solution (as measured by the objective function value) not in this subset.
3. 4-element subsets derived from the 3-element subsets by augmenting each 3-element subset to include the best solution (as measured by the objective function value) not in this subset.
4. The subsets consisting of the best b elements (as measured by the objective function value), for $b = 5, \dots, |B|$

Table 1.7 shows the subset generated using the current reference set.

Table 1.7. Subset generation

Type	Subsets
1	(S1, S2), (S1, S3), (S1, S4), (S1, S5), (S1, S6), (S2, S3), (S2, S4), (S2, S5), (S2, S6), (S3, S4), (S3, S5), (S3, S6), (S4, S5), (S4, S6), (S5, S6)
2	(S1, S2, S3), (S1, S2, S4), (S1, S2, S5), (S1, S2, S6), (S1, S3, S4), (S1, S3, S5), (S1, S3, S6), (S1, S4, S5), (S1, S4, S6), (S1, S5, S6)
3	(S1, S2, S3, S4), (S1, S2, S3, S5), (S1, S2, S3, S6), (S1, S2, S4, S5), (S1, S2, S4, S6), (S1, S2, S5, S6);
4	(S1, S2, S3, S5, S6), (S1, S2, S3, S4, S5, S6)

Subsets of type 1 are the fifteen possible combinations of two solutions. For type 2 subsets, we start by adding solution S_3 to the subset (S_1, S_2) . Then since adding solution S_2 to the subset (S_2, S_3) leads to a repeated subset, the procedure jumps to the next subset, adding S_2 to the subset (S_1, S_4) , and so forth. Type 3 subsets are created in a similar way, now starting by adding solution S_4 to the subset (S_1, S_2, S_3) . Finally, subsets of type 3 consist of successively adding solutions S_5 and S_6 to subsets (S_1, S_2, S_3, S_6) and $(S_1, S_2, S_3, S_5, S_6)$.

Solution Combination Method

This method is designed to explore subregions within the convex-hull of the reference set. We consider solutions encoded as vectors of variables x_{ij} representing edges (v_i, v_j) . New solutions are generated by weighted linear combinations which are structured by the subsets defined in the last step. In order to restrict the number of solutions only one solution is generated in each subset by a convex linear combination defined as follows.

Let E be a subset defined in RS , $|E| = r$, and let $H(E)$ denote the convex-hull of E . We generate solutions $S \in H(E)$ represented as

$$\begin{aligned} S &= \sum_{t=1}^r \lambda_t S_t \\ \sum_{t=1}^r \lambda_t &= 1 \\ \lambda_t &\geq 0 \quad j = 1, \dots, r \end{aligned}$$

where the multiplier λ_t represents the weight assigned to solution S_t . We compute these multipliers by

$$\lambda_t = \frac{\frac{1}{C(S_t)}}{\sum_{t=1}^r \frac{1}{C(S_t)}}$$

so that the better (lower cost) solutions receive higher weight than less attractive (higher cost) solutions. Then, we calculate the score of each variable x_{ij} relative to the solutions in E by computing

$$score(x_{ij}) = \sum_{t=1}^r (\lambda_t x_{ij}^t)$$

where x_{ij}^t means that x_{ij} is an edge in the solution S_t . Finally as variables are required to be binary, the value is obtained by rounding its score to give $x_{ij} = \lfloor score(x_{ij}) + .5 \rfloor$. The computation of the value for each variable in E results in the linear combination of the solutions in E . Table 1.8 shows the computation of the linear combination of solutions S_1 , S_2 , S_3 , and S_4 in the initial reference set illustrated in Table 1.7. For the sake of simplicity only edges that appear in at least one solution are represented in the table, since the remaining variables correspond to $x_{ij} = 0$.

Table 1.8. Linear combinations of solution vectors

Solution	$S_{1.}$	$S_{2.}$	$S_{3.}$	$S_{4.}$	$score(x_{ij})$	x_{ij}
$\lambda_{i,j}$	0.2643	0.2642	0.2524	0.2191		
Edges						
(1,4)		0.2642			0.2642	0
(1,7)			0.2524		0.2524	0
(1,8)	0.2643	0.2642		0.2191	0.7476	1
(1,9)	0.2643				0.2643	0
(1,11)			0.2524	0.2191	0.4715	0
(2,0)	0.2643	0.2642	0.2524	0.2191	1.0000	1
(3,0)	0.2643	0.2642	0.2524		0.7809	1
(3,4)	0.2643	0.2642	0.2524		0.7809	1
(3,7)				0.2191	0.2191	0
(3,14)				0.2191	0.2191	0
(4,0)				0.2191	0.2191	0
(4,8)			0.2524		0.2524	0
(4,9)				0.2191	0.2191	0
(4,11)	0.2643				0.2643	0
(5,0)				0.2191	0.2191	0
(5,6)	0.2643	0.2642	0.2524		0.7809	1
(5,12)			0.2524	0.2191	0.4715	0
(5,13)	0.2643	0.2642			0.5285	1
(6,0)	0.2643	0.2642	0.2524	0.2191	1.0000	1
(6,11)				0.2191	0.2191	0
(7,0)	0.2643	0.2642	0.2524	0.2191	1.0000	1
(7,8)	0.2643				0.2643	0
(7,11)		0.2642			0.2642	0
(8,0)		0.2642	0.2524	0.2191	0.7357	1
(9,0)	0.2643				0.2643	0
(9,10)		0.2642	0.2524	0.2191	0.7357	1
(9,11)		0.2642	0.2524		0.5166	1
(10,0)			0.2524		0.2524	0
(10,11)	0.2643				0.2643	0
(10,12)	0.2643	0.2642	0.2524	0.2191	1.0000	1
(12,0)	0.2643	0.2642			0.5285	1
(13,0)			0.2524	0.2191	0.4715	0
(13,14)	0.2643	0.2642	0.2524	0.2191	1.0000	1
(14,0)	0.2643	0.2642			0.5285	1

It is important to observe the effect of this weighting on the resulting solutions. As previously intimated, the least cost solution $S_{1.}$ has the largest weight in the combination. Also, note that both edges (5,12) and (5,13) appear in two solutions, though only the variable x_{513} is considered for the resulting solution.

A new solution can now be created using the edges associated with variables $x_{ij} = 1$. Nevertheless, the set of these edges does not necessarily (and usually doesn't) represent a feasible graph structure for a VRP solution. Instead, it produces a subgraph containing vertices with a degree different than two. Such subgraphs can be viewed as fragments of solutions (or partial routes). For example, the previous linear combination produced the following solution fragments: $(1,8,0)$, $(2,0)$, $(0,3,4)$, $(0,6,5,13,14,0)$, $(0,11,9,10,12,0)$, and $(7,0)$. To create a feasible solution subgraph we can simply link vertices 1, 2, 4, and 7 (which have a degree equal to 1) directly to the depot. This has the advantage that the algorithm always deals with feasible structures. However, it is also possible that the subgraph resulting from a linear combination contains vertices of degree greater than two. In these cases, a very straightforward technique consists of successively dropping edges with the smallest scores, from among those incident at these vertices, until the degree of each vertex becomes equal to two. By doing so, the subgraph obtained will be either feasible or fall into the case where some of the vertices have degree 1, which can be handled as already indicated.

Improvement Method

Even though the solution combination method creates feasible subgraphs for the VRP, the solution may still not be feasible in relation to the other problem constraints. Therefore, the improvement method must be able to deal with infeasible solutions. As already noted, the solution combination method has generated 33 new solutions (one per each subset). Let C_1, \dots, C_{33} denote these solutions indexed by the order they appear in Table 1.7. In the whole set, combinations C_{26}, \dots, C_{31} and C_{33} repeat previously generated solutions, so we first drop these solutions before applying the improvement method. The relatively high rate of repeated solutions results from the fact that the example deals with a very small instance. However, some number of repeated solutions may still be expected to appear when solving more realistic problems. Thus, it is valuable to have a fast procedure to identify repeated solutions which can be achieved in this context by using an appropriate hash function.

Table 1.9 shows an example of the improvement method applied to solution C_{12} where the local optimum C_{12}^* is found after 3 iterations. This is the same improvement method used in the initial phase of the scatter search algorithm, though the illustration shows how the method deals with infeasible solutions.

Table 1.9. Improving Method

Solution	k	R_k	Q_k	$C(R_k)$	$C(C_{12})$
Initial	1	0 7 1 11 9 10 0	28	19.58	
Solution	2	0 2 0	20	21.63	
C_{12}	3	0 13 14 3 4 8 0	35	37.00	
	4	0 5 6 0	19	9.36	
	5	0 12 0	3	17.02	104.60†
Iteration 1	Drop vertex 14 from R_3 and insert it into R_2 .				
	1	0 7 1 11 9 10 0	28	19.58	
	2	0 2 14 0	27	32.32	
	3	0 13 3 4 8 0	28	29.94	
	4	0 5 6 0	19	9.36	
	5	0 12 0	3	17.02	108.23
Iteration 2	Drop vertex 12 from R_5 and insert it into R_4 .				
	1	0 7 1 11 9 10 0	28	19.58	
	2	0 2 14 0	27	32.32	
	3	0 13 3 4 8 0	28	29.94	
	4	0 12 5 6 0	22	18.37	100.21
Iteration 3	Drop vertex 13 from R_3 and insert it into R_4 .				
Final	1	0 7 1 11 9 10 0	28	19.58	
Solution	2	0 2 14 0	27	32.32	
C_{12}^*	3	0 3 4 8 0	24	20.72	
	4	0 12 5 6 13 0	26	22.79	95.41

† infeasible solution, $Q_3 = 35 \geq Q = 30$

The method works in two stages. The first stage is concerned with making the solution feasible while choosing the most favorable move (relative to the objective function cost), and the second stage is the improvement process that operates only on feasible solutions. In general, several variants can be used to deal with infeasible solutions. These techniques are usually based on varying certain penalty factors associated with the problem constraints. Some constraints are potentially “less damaging” when violated than others, and usually the choice of penalty values should take this into consideration. Also, the way these penalties are modified can make the search more or less aggressive for moving into the feasible region. High penalty values are usually employed for an intensification strategy, and lower values for a diversification approach that allows the search keep going longer in the infeasible region. An interplay between intensification and diversification can be carried out by changing penalties according to certain patterns. For example, such patterns can be based on critical event memory within a strategic oscillation approach as suggested in tabu search. As previously noted, different balances between intensification and diversification of the search can also be controlled in the construction of the reference set, and in the solution combination method (as by considering non-convex linear

combinations to extrapolate to solutions outside the convex hull of a subset). Our illustrative method, however, doesn't use a penalty factor but rather identifies the most violated route and makes the best (cost reducing) move that consists of removing a vertex from this route and feasibly inserting it in another route. It is possible that an additional route will be created if such a move is not possible or if all routes are over their capacity.

A graphic representation of the procedure is illustrated in Figure 1.2.

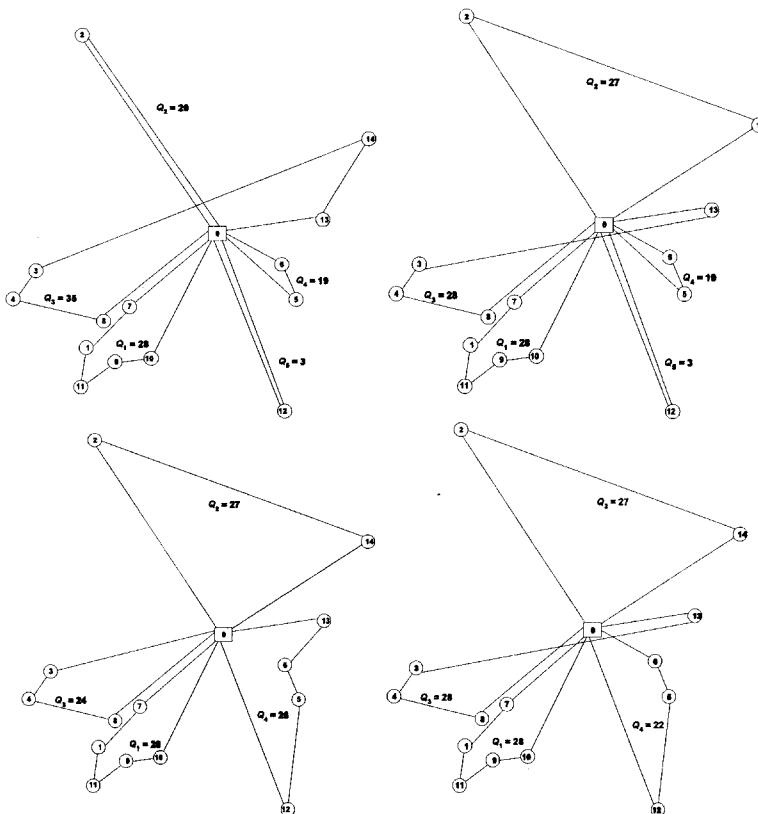


Figure 1.2. Graphical illustration of the improvement method

In our example, we apply the improving method to all the remaining (non-repeated) solutions to obtain a set of improved solutions. The best solution results from the improvement of C_{16} associated with a combination involving subset (S_1, S_2, S_3) . C_{16} is the solution $(0, 1, 8, 0, 2, 0, 3, 4, 0, 6, 5, 13, 14, 0, 7, 0, 12, 10, 9, 11, 0)$ with cost $C(C_{16}) = 115.94$, which is transformed by the

improvement method into the solution S_{16} with cost $C(S_{16}) = 91.01$. We have also solved the problem using an exact method (which requires substantially greater computation time, even for this simple example, and verify that S_{16} is in fact the optimal solution. This solution is illustrated in Figure 1.3.

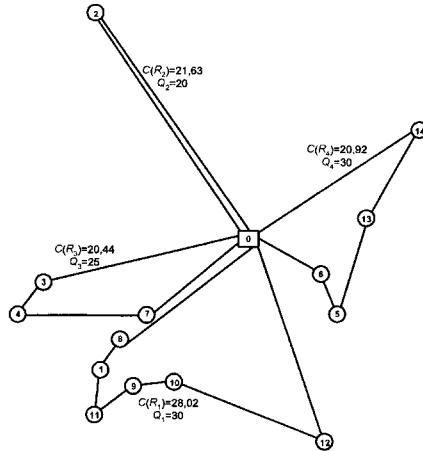


Figure 1.3. Final solution

In the absence of external confirmation that an optimal solution has been found, the method could continue to iterate in the scatter search phase by updating the reference set in the same way as in the initial phase. In this succeeding phase, however, the current reference set and the solutions produced by the improving method are all considered together before determining the new B and D sets that will form the updated reference set for the next iteration. The method stops when no elements in the current reference are replaced (i.e. when the reference set has not changed in two successive iterations).

5. Summary and Conclusion

We have developed a scatter search template for circuit-based graph problems and have shown its application to the classical vehicle routing problem under capacity restrictions.

Our intention was not to provide a computational exploration of scatter search. Rather, the numerous recent studies demonstrating the computational and practical efficacy of the approach provided one of the key motivations for the present paper. Another primary motivation for this tutorial paper was the fact that, in spite of its growing applications, scatter search has not yet been

described in a step by step manner in application to the routing context, accompanied by a detailed numerical illustration. For this reason, the access to applying the method to routing problems is undoubtedly somewhat less than it might otherwise be, and the amount of effort to launch a new scatter search study in this domain is also accordingly somewhat greater than would be necessary, due to the need to digest the elements of the method apart from an illustrated presentation of their nature and relevance.

The illustrative example used to demonstrate the scatter search provides a new evolutionary approach and a general framework for designing scatter search algorithms for vehicle routing. Moreover, because the problem constraints are handled separately from the solution generation procedures, and are therefore independent of the problem context, our scatter search design can be directly used to solve other classes of vehicle routing problems by applying any domain-specific (local search) heuristic that is able to start from infeasible solutions.

Finally, our illustrative approach affords various possibilities for using adaptive memory programming as a basis for creating effective scatter search algorithms for solving practical instances. In particular, we show the relevance of using memory to dynamically modify weights in the solution and to introduce an appropriate balance between intensification and diversification of the search.

Appendix

Table 1.10 shows the data of the problem used along this tutorial. The corresponding spatial distribution is depicted in Figure 1.4. The depot is represented by index 0, therefore q_0 is set to be zero.

Table 1.10. Problem data for the VRP instance

v_i	x_i	y_i	q_i
0	11.5	14.4	0
1	5.8	9.0	6
2	5.5	23.4	20
3	3.5	12.7	9
4	2.3	11.3	8
5	14.9	11.4	9
6	14.3	13.0	10
7	7.6	11.1	8
8	6.5	10.4	7
9	6.8	8.5	5
10	8.6	8.7	5
11	5.5	7.3	4
12	14.4	6.4	3
13	16.0	15.4	4
14	18.2	18.7	7

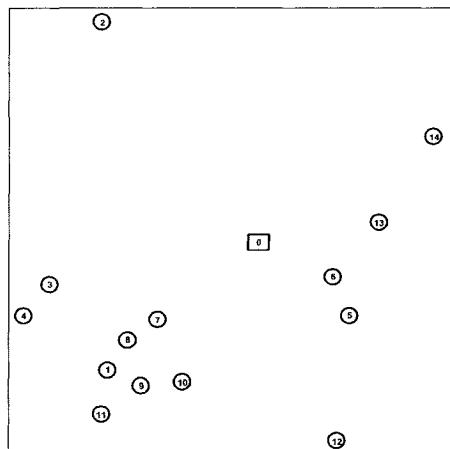


Figure 1.4. Spatial distribution of the problem vertices

References

- Atan, T. and N. Secomandi (1999) "A Rollout-Based Application of Scatter Search/Path Relinking Template to the Multi-Vehicle Routing Problem with Stochastic Demands and Restocking," PROS Revenue Management, Inc. Houston, TX.
- Campos, V., F. Glover, M. Laguna and R. Martí (1999) "An Experimental Evaluation of a Scatter Search for the Linear Ordering Problem," Research Report HCES-06-99, Hearin Center for Enterprise Science, School of Business Administration, University of Mississippi.
- Cung, V-D., T. Mautor, P. Michelon and A. Tavares (1996) "Scatter Search for the Quadratic Assignment Problem," Proceedings of the 1996 IEEE International Conference on Evolutionary Computation, 165-169.
- Fleurent, C., F. Glover, P. Michelon and Z. Valli (1996) "A Scatter Search Approach for Unconstrained Continuous Optimization," Proceedings of the 1996 IEEE International Conference on Evolutionary Computation, 643-648.
- Gendreau, M., A. Hertz and G. Laporte (1994) "Tabu Search Heuristic for the Vehicle Routing Problem," *Management Science*, 40:1276-1290.
- Glover, F. (1965) "A Multiphase-Dual Algorithm for Zero-One Integer Programming Problem," *Operations Research*, 13:879-919.
- Glover, F. (1977) "Heuristics for Integer Programming Using Surrogate Constraints," *Decision Sciences*, 8(1):156-166.
- Glover, F. (1994) "Genetic Algorithms and Scatter Search: Unsuspected Potentials," *Statistics and Computing*, 4:131-140.

- Glover, F. (1997) "A Template for Scatter Search and Path Relinking," in Lecture Notes in *Computer Science*, 1363, J.K. Hao, E. Lutton, E. Ronald, M. Schoenauer, D. Snyers (Eds.), 13-54.
- Glover, F., A. Løkketangen and D. Woodruff (1999) "Scatter Search to Generate Diverse MIP Solutions," Research Report, University of Colorado, Boulder.
- Glover F. and M. Laguna (1997) "*Tabu Search*," Kluwer Academic Publishers, Boston, MA.
- Glover F., M. Laguna and R. Martí (2001) "Scatter Search," to appear in Theory and Applications of Evolutionary Computation: Recent Trends, A. Ghosh and S. Tsutsui (Eds.), Springer-Verlag.
- Johnson, D.S. and L.A. McGeoch (1997) "The Traveling Salesman Problem: A Case Study in Local Optimization," in Local Search in Combinatorial Optimization, E. H. L. Aarts and J. K. Lenstra (Eds), John-Wiley and Sons, Ltd., 215-310.
- Kelly, J., B. Rangaswamy and J. Xu (1996) "A Scatter Search-Based Learning Algorithm for Neural Network Training," *Journal of Heuristics*, 2:129-146.
- Laguna, M. (2001) "Scatter Search," to appear in Handbook of Applied Optimization, P.M. Pardalos and M.G.C. Resende (Eds), Oxford Academic Press.
- Laguna, M., H. Lourenço and R. Martí (2000) "Assigning Proctors to Exams with Scatter Search," in Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research, M. Laguna and J. L. González-Velarde (Eds.), Kluwer Academic Publishers, 215-227.
- Laguna, M. and R. Martí (2000) "Experimental Testing of Advanced Scatter Search Designs for Global Optimization of Multimodal Functions," Working paper, University of Colorado, Boulder.
- Laporte, G. and I.H. Osman (1995) "Routing problems: A bibliography," *Annals of Operations Research*, 61:227-262.
- Rego, C. (1996) "Relaxed Tours and Path Ejections for the Traveling Salesman Problem," in Tabu Search Methods for Optimization, *European Journal of Operational Research*, 106:522-538.
- Rego, C. (1998) "A Subpath Ejection Method for the Vehicle Routing Problem," *Management Science*, 44(10):1447-1459.
- Rego, C. (2000) "Node Ejection Chains for the Vehicle Routing Problem: Sequential and Parallel Algorithms," Parallel Computing, forthcoming.
- Rochat Y. and E. Taillard (1995) "Probabilistic Diversification and Intensification in Local Search for Vehicle Routing," *Journal of Heuristics*, 1:147-167.
- Toth P. and D. Vigo (2003) "The Granular Tabu Search and Its Application to the Vehicle Routing Problem," *INFORMS Journal on Computing* 15(4):333-346.
- Xu, J., S. Chiu and F. Glover (2000) "Tabu Search and Evolutionary Scatter Search for 'Tree-Star' Network Problems, with Applications to Leased-Line Network Design," to appear in Telecommunications Optimization, David Corne (Ed.) Wiley.

Chapter 2

A MULTISTART SCATTER SEARCH HEURISTIC FOR SMOOTH NLP AND MINLP PROBLEMS

Zsolt Ugray¹, Leon Lasdon², John C. Plummer³, Fred Glover⁴, Jim Kelly⁵
and Rafael Martí⁶

¹*Business Information Systems Department, Utah State University, 3515 Old Main Hill
Logan, UT 84322-3515, zsolt.ugray@ucr.edu*

²*The University of Texas at Austin, McCombs School of Business, Management and
Information Systems Dept. B6500, Austin, TX 78712-0212, asdon@mail.utexas.edu*

³*Dept of CIS/QMST, Texas State University, 601 University Dr, San Marcos, TX 78666,
jcplummer@mail.utexas.edu*

⁴*Leads School of Business, University of Colorado, Boulder, CO 80309-0419,
fred.glover@colorado.edu*

⁵*OptTek Systems, Inc 1919 Seventh St., Boulder, CO 80302, Kelly@opttek.com*

⁶*Departamento de Estadística e I.O., Facultad de Matemáticas, Universitat de Valencia,
Dr. Moliner 50, 46100 Burjassot, Valencia, Spain, rafael.marti@uv.es*

Abstract: The algorithm described here, called OptQuest/NLP or OQNLP, is a heuristic designed to find global optima for pure and mixed integer nonlinear problems with many constraints and variables, where all problem functions are differentiable with respect to the continuous variables. It uses OptQuest, a commercial implementation of scatter search developed by OptTek Systems, Inc., to provide starting points for a gradient-based local NLP solver. This solver seeks a local solution from a subset of these points, holding discrete variables fixed. The procedure is motivated by our desire to combine the superior accuracy and feasibility-seeking behavior of gradient-based local NLP solvers with the global optimization abilities of OptQuest. Computational results include 144 smooth NLP and MINLP problems due to Floudas et al, most with both linear and nonlinear constraints, coded in the GAMS modeling language. Some are quite large for global optimization, with over 100 variables and many constraints. Global solutions to almost all problems are found in a small number of NLP solver calls, often one or two.

Keywords: Global Optimization, Multistart Heuristic, Mixed Integer Nonlinear Programming, Scatter Search, Gradient Methods

1. Introduction

This paper describes a multistart heuristic algorithm designed to find global optima of smooth constrained nonlinear programs (NLPs) and mixed integer nonlinear programs (MINLPs). It uses the widely used scatter search software OptQuest (Laguna and Martí, 2000) to generate trial points, which are candidate starting points for a local NLP solver. These are filtered to provide a smaller subset from which the local solver attempts to find a local optimum. Our implementation uses the generalized reduced gradient NLP solver GRG (Smith and Lasdon, 1993), but in principle any local NLP solver can be used, including ones that do not require derivatives. However, we focus on gradient-based solvers because they are by far the most widely used, and currently are the only ones capable of solving NLPs with hundreds or thousands of variables and constraints.

- Problem Statement

The most general problem this algorithm can solve has the form

$$\text{minimize } f(x, y) \quad (1.1)$$

subject to the nonlinear constraints

$$gl \leq G(x, y) \leq gu \quad (1.2)$$

the linear constraints

$$l \leq A_1 x + A_2 y \leq u \quad (1.3)$$

$$x \in S, y \in Y \quad (1.4)$$

where x is an n -dimensional vector of continuous decision variables, y is a p -dimensional vector of discrete decision variables, and the vectors gl , gu , l , u , contain upper and lower bounds for the nonlinear and linear constraints respectively. The matrices A_1 and A_2 are m_2 by n and m_2 by p respectively, and contain the coefficients of any linear constraints. The set S is defined by simple bounds on x , and we assume that it is closed and bounded, i.e., that each component of x has a finite upper and lower bound. This is required by the OptQuest procedure. The set Y is assumed to be finite, and is often the set of all p -dimensional binary or integer vectors y . The objective function f and the m_1 -

dimensional vector of constraint functions G are assumed to have continuous first partial derivatives at all points in $S \times Y$. This is needed in order that a gradient-based local NLP solver be applicable to relaxed NLP subproblems formed from (1)-(3) by allowing the y variables to be continuous.

- Multi-start Algorithms

In this section, which reviews past work on multi-start algorithms, we focus on unconstrained problems where there are no discrete variables, since to the best of our knowledge multi-start algorithms have been investigated theoretically only in this context. These problems have the form

$$\text{minimize } f(x) \quad (1.5)$$

$$\text{subject to } x \in S \quad (1.6)$$

where all global minima of f are assumed to occur in the interior of S . By multi-start we mean any algorithm that attempts to find a global solution to (1.5)-(1.6) by starting a local NLP solver, denoted by L , from multiple starting points in S . The most basic multi-start method generates uniformly distributed points in S , and starts L from each of these. This is well known to converge to a global solution with probability one as the number of points approaches infinity--in fact, the best of the starting points converges as well. This procedure is very inefficient because the same local solution is located many times. A convergent procedure that largely overcomes this difficulty is called multi-level single linkage (MLSL) (Rinnooy Kan and Timmer, 1987). This uses a simple rule to exclude some potential starting points. A uniformly distributed sample of N points in S is generated, and the objective, f , is evaluated at each point. The points are sorted according to their f values, and the qN best points are retained, where q is an algorithm parameter between 0 and 1. L is started from each point of this reduced sample, except if there is another sample point within a certain critical distance which has a lower f value. L is also not started from sample points which are too near the boundary of S , or too close to a previously discovered local minimum. Then, N additional uniformly distributed points are generated, and the procedure is applied to the union of these points and those retained from previous iterations. The critical distance referred to above decreases each time a new set of sample points is added. The authors show that, if the sampling continues indefinitely, each local minimum of f will be located, but the total number of local searches is finite with probability one. They also develop Bayesian stopping rules, which incorporate assumptions about the costs and

potential benefits of further function evaluations, to determine when to stop the procedure.

When the critical distance decreases, a point from which L was previously not started may become a starting point in the next cycle. Hence all sample points generated must be saved. This also makes the choice of the sample size, N , important, since too small a sample leads to many revised decisions, while too large a sample will cause L to be started many times. Recently, Locatelli and Schoen (1999) introduce a class of “Random Linkage” (RL) multi-start algorithms that retain the good convergence properties of MLSL, and do not require that past starting decisions be revised. Uniformly distributed points are generated one at a time, and L is started from each point with a probability given by a nondecreasing function $\phi(d)$, where d is the distance from the current sample point to the closest of the previous sample points with a better function value. Assumptions on this function that give RL methods the same theoretical properties as MLSL are derived in the above reference.

Recently, Fylstra et al. have implemented a version of MLSL which can solve constrained problems (Frontline Systems, Inc., 2000). See also www.frontsys.com. Limited to problems with no discrete variables y , it uses the L_1 exact penalty function, defined as

$$P_1(x, w) = f(x) + \sum_{i=1}^m w_i \text{viol}(g_i(x)) \quad (1.7)$$

where the w_i are nonnegative penalty weights, $m = m_1 + m_2$, and the vector g has been extended to include the linear constraints (1.4). The function $\text{viol}(g_i(x))$ is equal to the absolute amount by which the i th constraint is violated at the point x . It is well known (see (Nash and Sofer, 1996) that if x^* is a local optimum of (1.1)-(1.4), u^* is a corresponding optimal multiplier vector, and the second order sufficiency conditions are satisfied at (x^*, u^*) , then if

$$w_i > \text{abs}(u_i^*) \quad (1.8)$$

x^* is a local unconstrained minimum of P_1 . If (1.1)-(1.4) has several local minima, and each w_i is larger than the maximum of all absolute multipliers for constraint i over all these optima, then P_1 has a local minimum at each of these local constrained minima. Even though P_1 is not a differentiable function of x , MLSL can be applied to it, and when a randomly generated trial point satisfies the MLSL criterion to be a starting point, any local solver for the smooth NLP problem can be started from that point. The local solver need not make any reference to the exact penalty function P_1 , whose only role is to provide function values to MLSL. We will use P_1 in the same way in our OQNLP algorithm. We are not aware of any theoretical investigations of this extended MLSL

procedure, so it must currently be regarded as a heuristic. Comparative testing of this extension of MLSL and OQNLP is planned.

2. Scatter Search and the Optquest Callable Library

We provide only a brief description of Scatter Search and its implementation in the OptQuest callable library here, since it is discussed in Laguna and Martí (2001) and in Laguna and Martí (2000). See also www.opttek.com. Like genetic algorithms (GAs) and other metaheuristics, scatter search operates on a set of solutions, called the *population* in the GA literature and the *reference set* in scatter search papers, that is maintained and updated from iteration to iteration. Scatter search (SS) is a novel instance of evolutionary methods, because it violates the premise that evolutionary approaches must be based solely on randomization. SS is also novel, in comparison to GAs, by being founded on strategies that were proposed as augmentations to GAs more than a decade after their debut in scatter search. Scatter search embodies principles and strategies that are still not emulated by other evolutionary methods, and that prove advantageous for solving a variety of complex optimization problems.

Scatter Search is designed to operate on a set of points, called reference points, which constitute good solutions obtained from previous solution efforts. Notably, the basis for defining “good” includes special criteria such as diversity that purposefully go beyond the objective function value. The approach systematically generates combinations of the reference points to create new points, each of which may (optionally) be mapped into an associated feasible point. The underlying combination mechanism uses linear combinations.

OptQuest is available as a callable library written in C, which can be invoked from any C program, or as a dynamic linked library (DLL) which can be called from a variety of languages including C, Visual Basic, and Java. The callable library consists of a set of functions which (a) input the problem size and data, (b) set options and tolerances, (c) create an initial reference set, (d) retrieve a trial solution to be evaluated and, (e) communicate these objective and constraint values back to OptQuest, which uses them as the input to update the reference set. For a complete description, see Laguna and Martí (2001).

3. Gradient-Based NLP Solver and GRG

There are many papers and texts discussing gradient-based NLP solvers, e.g., Nash and Sofer (1996), Nocedal and Wright (1999), Edgar, Himmelblau and Lasdon (2001). These solve problems of the form (1.1)-(1.4), but with no discrete (y) variables. They require a starting point as input, and use values and gradients of the problem functions to generate a sequence of points which, under fairly general smoothness and regularity conditions, converges to a local optimum. The main classes

of algorithms in widespread use today are Successive Quadratic Programming (SQP) and Generalized Reduced Gradient (GRG)—see Edgar, Himmelblau and Lasdon (2001), Chapter 8. The algorithm implemented in the widely used MINOS solver (Murtagh and Saunders, 1982) is similar to SQP. If there are nonlinear constraints, SQP and MINOS generate a sequence of points that usually violate the nonlinear constraints, with the violations decreasing to within a specified feasibility tolerance as the sequence converges to a local optimum. GRG algorithms have a simplex-like phase 1-phase 2 structure. Phase 1 begins with the given starting point and, if it is not feasible, attempts to find a feasible point by minimizing the sum of constraint violations. If this effort terminates with some constraints violated, the problem is assumed to be infeasible. However, this local optimum of the phase 1 objective may not be global, so a feasible point may exist. If a feasible point is found, phase 2 uses it as its starting point, and proceeds to minimize the true objective. Both phases consist of a sequence of line searches, each of which produces a feasible point with an objective value no worse (and usually better) than its predecessor.

There are several parameters and options that strongly influence the reliability and efficiency of a GRG implementation. The *feasibility tolerance*, ft , (default value 1.e-4) determines when a constraint is satisfied. If the constraint has the form $g(x) \geq l$, it is considered satisfied in the GRG code used here if

$$abs(g(x) - l) \geq -ft(1.0 + abs(l)).$$

The optimality tolerance, ot , (default value 1.e-4) and a number of consecutive iterations, $nstop$, (default value 10) determine when the current point is declared optimal. This occurs when

$$kterrnorm \leq ot$$

where $kterrnorm$ is the infinity norm of the error in the Kuhn-Tucker conditions, or when

$$abs(f(x_k) - f(x_{k+1})) \leq ot(1.0 + abs(f(x_k)))$$

for $nstop$ consecutive values of the iteration index, k .

Several good commercially available implementations of GRG and SQP solvers exist—see Nash (1998) for a review. As with any numerical analysis software, a local NLP solver can fail to find a local solution from a specified starting point. The problem may be too badly conditioned, badly scaled, or too large for the solver, causing it to terminate at a point (feasible or infeasible) which is not locally optimal. While the reliability of the best current NLP solvers is quite high, these

difficulties occurred several times in our computational testing, and we discuss this in more detail later.

Let L be a local NLP solver capable of solving (1.1)-(1.4), and assume that L converges to a local optimum for any starting point $x_0 \in S$. Let $L(x_0)$ be the locally optimal solution found by L starting from x_0 , and let $x_i^*, i = 1, 2, \dots, n_{loc}$ be all the local optima of the problem. The basin of attraction of the i th local optimum relative to L , denoted by $B(x_i^*)$, is the set of all starting points in S from which the sequence of points generated by L converges to x_i^* . Formally:

$$B(x_i^*) = \{x_0 \mid x_0 \in S, L(x_0) = x_i^*\}. \quad (3.1)$$

One measure of difficulty of a global optimization problem with unique global solution x_1^* is the volume of $B(x_1^*)$ divided by the volume of the rectangle, S , the *relative volume* of $B(x_1^*)$. The problem is trivial if this relative volume is 1, as it is for convex programs, and problem difficulty increases as this relative volume approaches zero.

4. Comparing Search Methods and Gradient-Based NLP Solvers

For smooth problems, the relative advantages of a search method like OptQuest over a gradient-based NLP solver are its ability to locate an approximation to a good local solution (often the global optimum), and the fact that it can handle discrete variables. Gradient-based NLP solvers converge to the “nearest” local solution, and have no facilities for discrete variables, unless they are imbedded in a rounding heuristic or branch-and-bound method. Relative disadvantages of search methods are their limited accuracy, and their weak abilities to deal with equality constraints (more generally, narrow feasible regions). They find it difficult to satisfy many nonlinear constraints to high accuracy, but this is a strength of gradient-based NLP solvers. Search methods also require an excessive number of iterations to find approximations to local or global optima accurate to more than 2 or 3 significant figures, while gradient-based solvers usually achieve 4 to 8-digit accuracy rapidly.

The motivation for combining search and gradient-based solvers in a multi-start procedure is to achieve the advantages of both while avoiding the disadvantages of either. Surprisingly, we have been unable to locate any published efforts in this direction, besides the Frontline extended MLSL method discussed in Section 2.

5. The OQNLP Algorithm

A pseudo-code description of the simplest OQNLP algorithm follows:

- INITIALIZATION
 Read_Problem_Parameters (n, p, m_1, m_2 , bounds, starting point);
 Setup_OptQuest_Parameters (size, iteration limits, population, accuracy, variables, bounds, constraints);
 Initialize_OptQuest_Population;
- STAGE 1: INITIAL OPTQUEST ITERATIONS AND FIRST GRG CALL
WHILE (unevaluated trial points from initial population remain) **DO**
 {
 Get (trial solution from OptQuest);
 Evaluate (objective and constraint values at trial solution,);
 Put (trial solution , objective and constraint values to OptQuest database);
 }
ENDDO
 Get_Best_Point_from_OptQuest_database (starting point);
Call_GRG (starting point, local solution);
 threshold = P_1 value of local solution;
- STAGE 2: MAIN ITERATIVE LOOP
WHILE (stopping criteria not met) **DO**
 {
 Get (trial solution from OptQuest);
 Evaluate (objective and constraint values at trial solution,);
 Put (trial solution, objective and constraint values to OptQuest database);
 Calculate_Penalty_Function (trial solution, P_1);
 IF (distance and merit filter criteria are satisfied) **THEN**
 {
 Call_GRG (trial solution, local solution);
 Analyze_Solution (GRG Terminating Condition);
 Update_Local_Solutions_Found;
 Update_Largest_Lagrange_Multipliers_Found;
 }
 ELSE IF ($P_1 >$ threshold for *waitcycle* consecutive iterations)
 increase *threshold*
 }
ENDDO

After initialization, there are two main stages. In the “initial OptQuest iterations” stage, the objective and constraint values at all trial points generated by the initial OptQuest population (including the

population points themselves) are evaluated, and these values are returned to OptQuest, which computes its penalty function, P , at each point. The point with the best P value is selected, and GRG is started from this point. If there are any discrete variables, y , they are fixed at their current values during the GRG solution. In general, the trial points are scattered within the rectangle defined by the bounds on the variables, so choosing the best corresponds to performing a coarse search over this rectangle. If the best point falls inside the basin of attraction of the global optimum relative to the GRG solver (as it often does), then if the subsequent GRG call is successful, it will find a global optimum. This call also determines optimal Lagrange multiplier values, u^* , for the constraints. These are used to determine initial values for the penalty weights, w_i , satisfying (1.8), which are used in the exact penalty function, P_1 , defined in (1.7). All local optima found are stored in a linked list, along with the associated Lagrange multipliers and objective values. Whenever a new local optimum is found, the penalty weights are updated so that (1.8) is satisfied over all known local optima.

The main iterative loop of stage 2 obtains trial points from OptQuest, and starts GRG from the subset of these points determined by two filters. The distance filter helps insure that the GRG starting points are diverse, in the sense that they are not too close to any previously found local solution. Its goal is to prevent GRG from starting more than once within the basin of attraction of any local optimum, so it plays the same role as the rule in the MSL algorithm of Section 2, which does not start at a point if it is within a critical distance of a better point. When a local solution is found, it is stored in a linked list, ordered by its objective value, as is the Euclidean distance between it and the starting point that led to it. If a local solution is located more than once, the maximum of these distances, $maxdist$, is updated and stored. For each trial point, t , if the distance between t and any local solution already found is less than $distfactor * maxdist$, GRG is not started from the point, and we obtain the next trial solution from OptQuest.

This distance filter implicitly assumes that the attraction basins are spherical, with radii at least $maxdist$. The default value of $distfactor$ is 0.75, and it can be set to any positive value. As $distfactor$ approaches zero, the filtering effect vanishes, as would be appropriate if there were many closely spaced local solutions. As it becomes larger than 1, the filtering effect increases until eventually GRG is never started.

The merit filter helps insure that the GRG starting points have high quality, by not starting from candidate points whose exact penalty function value P_1 (see (1.7)) is greater than a threshold. This threshold is set initially to the P_1 value of the best candidate point found in the first stage of the algorithm. If trial points are rejected by this test for more than $waitcycle$ consecutive iterations, the threshold is increased by the updating rule:

$$\text{threshold} \leftarrow \text{threshold} + \text{thfact} * (1.0 + \text{abs}(\text{threshold})) \quad (4.2)$$

where the default value of *thfact* is 0.2 and that for *waitcycle* is 20. The additive 1.0 term is included so that *threshold* increases by at least *thfact* when its current value is near zero. When a trial point is accepted by the merit filter, *threshold* is decreased by setting it to the P_1 value of that point.

The combined effect of these 2 filters is that GRG is started at only a few percent of the OptQuest trial points, yet global optimal solutions are found for a very high percentage of the test problems. Some insight is gained by examining Figure 2.1, which shows the stationary point at the origin and the 6 local minima of a 2 variable unconstrained function (called the six-hump camelback function) as dark squares, labeled with their objective value. The ten points from which OQNLP starts GRG are shown as white diamonds. The local minima occur in pairs with equal objective value, located symmetrically about the origin. There were 144 trial points generated in stage 1, and 10 points in the initial population. The best of these 154 points is the population point (0,0), so this becomes the first starting point for GRG. This happens to be a stationary point of F, so it satisfies the GRG optimality test (that the norm of the gradient of the objective be less than the optimality tolerance), and GRG terminates there. The next GRG start is at iteration 201, and this locates the global optimum at (.0898, -.7127), which is located two times. The other global optimum at (-.0898, .7127) is found first at iteration 268, and is located 6 times.

The limit on total OQNLP iterations in this run was 1000. GRG was started at only 9 of the 846 OptQuest trial points generated in the main iterative loop of stage 2. All but 2 of the starting points are in the basin of attraction of one of the two global optima. This is mainly due to the merit filter. In particular, the threshold values are always less than 1.6071, so no starts are ever made in the basin of attraction of the two local optima with this objective value. The merit filter alone rejected 498 points, the distance filter alone 57, and both rejected 281.

Figure 2.2 illustrates the dynamics of the merit filtering process for iterations 155 to 407 of this problem, displaying the objective values for the trial points as white diamonds, and the threshold values as dark lines. All objective values greater than 2.0 are set to 2.0.

The initial threshold value is zero, and it is raised twice to a level of 0.44 at iteration 201, where the trial point objective value of -0.29 falls below it. GRG is then started and locates the global optimum at (.0898, -.7127), and the threshold is reset to -0.29. This cycle then repeats. Nine of the ten GRG starts are made in the 252 iterations shown in the graph. In this span, there are 12 points where the merit filter allows a start and the threshold is decreased, but GRG is not started at three of these because the distance filter rejects them.

Figure 2.3 shows the same information for iterations 408 to 1000. There is only one GRG start in this span. This is not due to a lack of high quality trial points: there are more good points than previously, many with values near or equal to -1.0310 (the global minimum is -1.0316), and the merit threshold is usually -1.0310 as well. Every time this threshold is raised, the merit filter accepts one of the next trial points, but 51 of the 52 accepted points are too near one of the 2 global optima, and they are rejected by the distance filter.

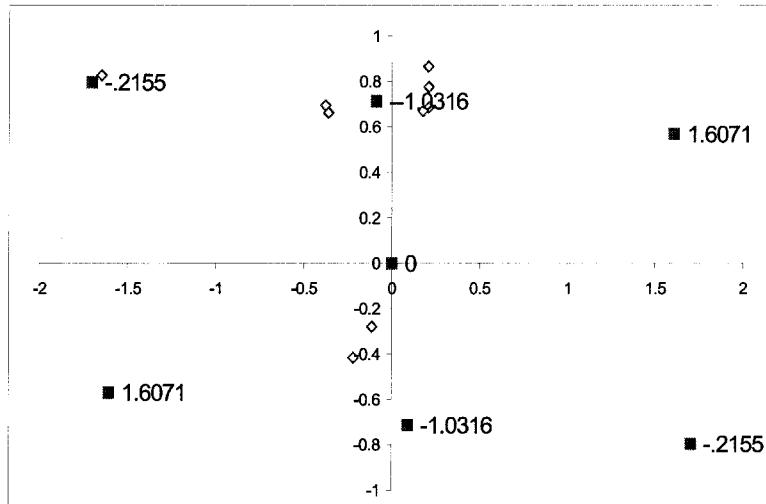


Figure 2.1. Local Optima and 10 GRG starting points for Six-Hump Camelback function

This simple example illustrates a number of important points:

1. Setting the bounds on the continuous or discrete variables to be too large in magnitude is likely to slow the OQNLP algorithm (or any search algorithm) and may lead to a poorer final solution. In the above example, if the variable bounds had been $[-2,2]$ rather than $[10,10]$, the trial points generated by the initial population would have had much lower objective values. OptQuest can overcome this when the initial population is updated.
2. GRG found a highly accurate approximation to the global solution of this unconstrained problem at its second call. OptQuest alone would have taken many more iterations to achieve this accuracy.
3. The best trial point generated by the initial reference set may not have as good an objective value as those generated from the second or succeeding ones, especially if the variable bounds are too large. Using the best “first generation” point as the initial GRG starting point may not lead to as good a local solution as if some “second generation” points had been considered. For this reason our base case computational results use a first stage of 200 OptQuest trial points,

which in this example would include all 144 first generation points and 56 from the second generation.

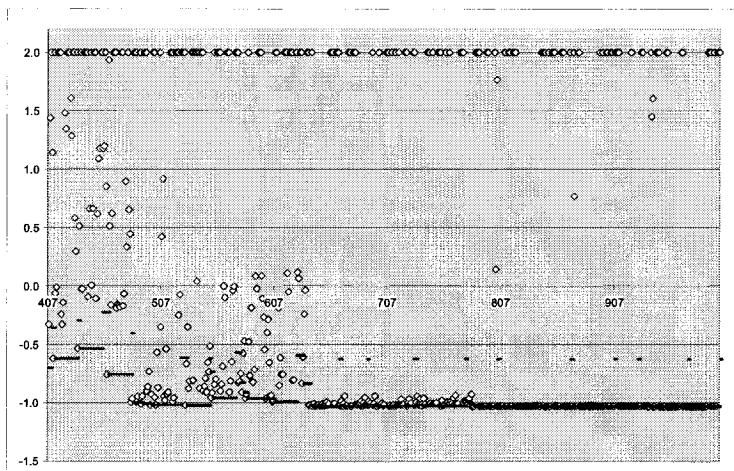


Figure 2.2. Objective and threshold values for Six-Hump Camelback function for iterations 155 to 407

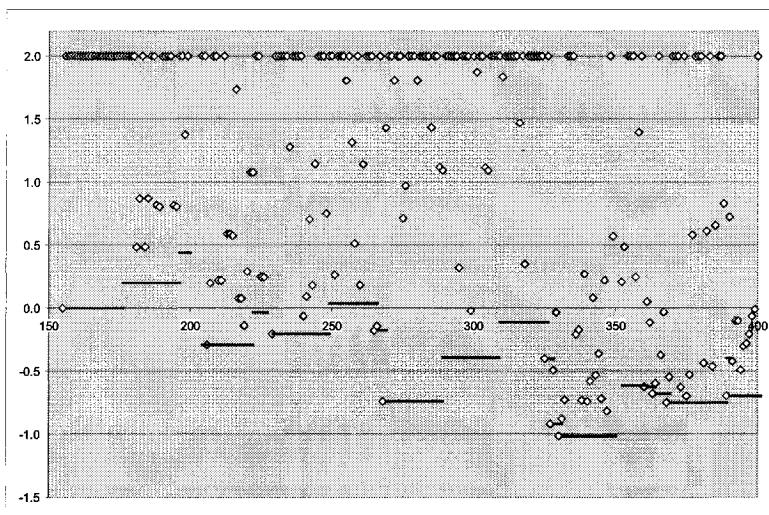


Figure 2.3. Objective and threshold values for Six-Hump Camelback function: iterations 408 to 100

Filtering Logic for Problems with Discrete Variables

The filtering logic described above must be extended when there are discrete variables (the y variables in the problem statement (1.1)-(1.4)). When a trial point (xt, yt) provided by OptQuest passes the two filtering tests and is passed to GRG, xt acts as a starting point and is changed by GRG, but the yt values are fixed and are not changed. Each new set of yt values defines a different NLP for GRG to solve, say $NLP(yt)$, with its own set of local minima in x space, so both filters must be made specific to $NLP(yt)$. For the distance filter, it is irrelevant if xt is close to any local minima (in x space) previously found which correspond to problems $NLP(y)$ with y different from yt . Hence the distance filter is based on the distance from xt to local minima of $NLP(yt)$ only. Similarly, the tests and threshold values in the merit filter must be specific to the problem $NLP(yt)$ currently being solved. However, the weights w in the exact penalty function $P_1(x, y, w)$ used in the merit filter are based on the maximum absolute multipliers over *all* local optima for *all* vectors yt , because these weights are large enough to ensure that this function is exact for all problems $NLP(y)$.

Therefore, in stage 2 of the algorithm, the exact penalty function, $P_1(xt, yt, w)$, is calculated at each trial point (xt, yt) , and GRG is started at (xt, yt) if P_1 is smaller than the current threshold for $NLP(yt)$. This threshold is initialized to plus infinity, so if the values yt have not occurred in a previous stage 2 trial point, GRG will be called at this point. This leads to many more GRG calls in problems with discrete variables, as we show later in the computational results sections.

This OQNLP algorithm should be regarded as a base case from which extensions will be explored and compared. The most significant of these involves the return of information from GRG to OptQuest, which is absent in the above procedure, i.e. local solutions found by GRG are not returned to OptQuest. Such solutions are generally of very high quality, and might aid the search process if they were incorporated into the OptQuest population, because at least a subset would likely be retained there. However, this should be done so as to preserve the diversity of the population. We discuss this option further in Section 9.

6. C Language Implementation, Games Interface, and the Floudas Test Problem Set

The algorithm described in the previous section has been implemented as a callable C-language function. In this form, the user supplies a C function that evaluates the objective and constraint functions, an optional routine that evaluates their first partial derivatives (finite difference approximations are used otherwise), and a calling program that supplies problem size, bounds, and an initial point, and invokes the algorithm. Algorithm parameters and options are in an options text file. We have developed an interface between this C implementation and the GAMS algebraic modeling language (see www.gams.com), using C library

routines generously provided by GAMS Development Company. The user function routine is replaced by one that calls the GAMS interpreter, and a special derivative routine accesses and evaluates expressions developed by GAMS for first derivatives of all nonlinear problem functions. GAMS identifies all linear terms in each function, and supplies their coefficients separately, thus identifying all linear constraints. This enables us to invoke the OptQuest option which maps each trial point (generated as described in Section 2) into a point which satisfies the linear constraints. This is done by solving a linear program that minimizes the L_1 distance between the trial point and the feasible region defined by the linear constraints. The derivative information supplied by GAMS also significantly enhances the performance of gradient-based NLP solvers, since only non-constant derivatives are re-evaluated, and these are always available to full machine precision.

Part of the motivation for developing this GAMS interface was the existence of a large set of global optimization test problems coded in GAMS, described in Floudas et al. (1999). This text describes some problems that cannot be represented in GAMS, but there are many that can, and these can be downloaded from <http://titan.princeton.edu/TestProblems/> or from www.gams.com, linking to gams world and then global. Characteristics of 142 of these problems (excluding the 8_6_1 and 8_6_2 sets) are contained in Table 2.1.

Most of these problems arise from chemical engineering, but some are from general problem classes. Most are small, but a few have over 100 variables and comparable numbers of constraints, and some have both continuous and discrete variables. Almost all of the problems without discrete variables have local solutions distinct from the global solution, and the majority of problems have constraints. Sometimes all constraints are linear, as with the concave quadratic programs of series EX2_1_x, but many problems have nonlinear constraints, and these are often the source of the nonconvexities. For example, there are many problems arising from pooling and blending applications with bilinear constraints. The best known objective value and (in most cases) the corresponding variable values are provided in Floudas, et al. (1999). The symbol N in the rows for the series EX8_6_1 and EX8_6_2 is the number of particles in a cluster whose equilibrium configuration is sought via potential energy minimization. Each particle has 3 coordinates, so there are 3N variables.

Table 2.1. Characteristics of Floudas GAMS test problems

SERIES	PROBLEMS	MAX VARS	MAX DISCRETE VARS	MAX LINEAR CONSTRAINTS	MAX NONLINEAR CONSTRAINTS	PROBLEM TYPE
EX2_1_x	14	24	0	10	0	concave QP (min)
EX3_1_x	4	8	0	4	6	quadratic obj and constraints
EX4_1_x	9	2	0	0	2	obj or constraints polynomial
EX5_2_x	2	32	0	8	11	bilinear-pooling
EX5_3_x	2	62	0	19	34	distillation column sequencing
EX5_4_x	3	27	0	13	6	heat exchanger network
EX6_1_x	4	12	0	3	6	gibbs free energy min
EX6_2_x	10	9	0	3	0	gibbs free energy min
EX7_2_x	4	8	0	3	12	generalized geometric prog
EX7_3_x	6	17	0	10	11	robust stability analysis
EX8_1_x	8	6	0	0	5	small unconstrained, constrained
EX8_2_x	5	55	0	6	75	batch plant design-uncertainty
EX8_3_x	14	141	0	43	65	reactor network synthesis
EX8_4_x	8	62	0	0	40	constrained least squares
EX8_5_x	6	6	0	2	2	min tangent plane distance
EX8_6_1	N from 4 to 147	3N	0	0	0	Lenard-Jones energy min
EX8_6_2	N from 5 to 80	3N	0	0	0	Morse energy min
EX9_1_x	10	29	0	27	5	bilevel LP
EX9_2_x	9	16	0	11	6	bilevel QP
EX12_2_x	6	11	8	9	4	MINLP
EX14_1_x	9	10	0	4	17	infinity norm solution of equations
EX14_2_x	9	7	0	1	10	infinity norm solution of equations
Total	142					

7. Computational Results on the Floudas Set of Test Problems

This section describes the results obtained when the OQNLP algorithm described in Section 6 is applied to the Floudas GAMS test problems. The main algorithm parameters and options used are shown in Table 2.2 below.

Table 2.2. OptQuest, GRG, and OQNLP parameters and options used

OptQuest and OQNLP Parameters	GRG Parameters
Use linear constraints = yes	Feasibility tolerance = 1.e-4, except series 8_3_x uses 1.e-6
Total iterations = 1000	Optimality tolerance = 1.e-4, except series 8_3_x uses 1.e-6
Total stage 1 iterations = 200	Consecutive iterations for fractional change termination = 20
<i>Waitcycle</i> = 20	
<i>Thfact</i> = 0.2 (see (5.2))	
<i>Distfact</i> = 0.75	
OptQuest search type = <i>boundary</i>	
Boundary search parameter = 0.5	
OptQuest Variable Precision = 1.e-4	
Check for duplicates in database = yes	

As discussed in Section 6, OptQuest can insure that all trial points satisfy any linear constraints, and we use this option in our tests below. The boundary search strategy is the OptQuest default, as is its parameter value of

As discussed in Section 6, OptQuest can insure that all trial points satisfy any linear constraints, and we use this option in our tests below. The boundary search strategy is the OptQuest default, as is its parameter value of 0.5. This strategy directs the trial points generated towards the boundary of the region defined by the variable bounds and general linear constraints 50% of the time. For the problem series 8_3_x, the largest of the group with 9 of 10 problems having over 100 variables, we used GRG optimality and feasibility tolerances of 1.e-6 because the default values of 1.e-4 led to GRG termination significantly short of local optimality.

Table 2.3, found in the Appendix, contains the results for 120 of the 131 problems with no discrete variables, sorted by increasing number of variables, with averages for six groups of problems. We exclude the 8_6_1 and 8_6_2 series, which are described separately below. We also exclude eleven problems which were either extremely large, or for which GRG could not find local solutions from most or all starting points. This was almost always due to failure to find a feasible solution, due to termination at a local minimum of the phase one objective. These computations used a 1.3 ghz Dell Optiplex GX400 PC with the Windows 2000 OS. The “fcn call” columns record the total number of times all problem functions are evaluated. The column headed “% gap” is the percentage difference between the best feasible objective value found by OQNLP and the best known value, i.e., the ratio $\text{gap} = 100 * (\text{OQNLP obj} - \text{bestobj}) / (1 + \text{abs(bestobj)})$.

For minimization problems, and its negative for maximization. Hence a negative *gap* indicates that OQNLP found a feasible point with better objective value than the “best known” value provided in Floudas, et al. (1999). All but 6 of the 120 problems have gaps less than 1%, most much smaller. These 6 are solved to very small gaps using 5000 iterations

and, for 2 problems, increasing the boundary parameter, as is discussed shortly. There are 2 problems with sizeable negative gaps, indicating that OQNLP found a better value than the best reported. The column headed “max abs x” is the largest absolute component of the decision vector, x , in the best solution found. Large values indicate a more difficult problem, in the sense that the rectangle defined by the variable bounds is larger, so the search must cover a larger volume.

This “base case” OQNLP algorithm finds its best solution very quickly. The best OQNLP solution is found by the first GRG call in 88 of the 120 problems, and in the second GRG call in 7 more, confirming the effectiveness of stage 1 of the algorithm in finding an initial GRG starting point within the basin of attraction of the global optimum. This happens most often in the smaller problems, but occurs 11 times in the 20 largest.

Table 2.4 aggregates the averages for the six groups of problems in Table 2.3, and includes the following ratios: GRG ratio = average GRG calls to best/average total GRG calls with similar definitions for iterations, function calls, and computation time.

The groups are ordered in terms of increasing number of variables, and the number of local optima found increases with problem size, with an average of 22.6 for the largest group. The measures of computational effort to find the best solution (iterations to best, grg calls to best, function calls to best, and time to best) are all gratifyingly small, and most increase slowly with problem size. Function calls are much higher for the largest group (110 to 141 variables), reflecting the GRG effort required to solve these problems, which have many nonlinear constraints.

Table 2.4. Average performance statistics for 6 groups of problems

VAR RANGE	NO. OF PROBS	ITNS TO BEST	GRG CALLS TO BEST	TOTAL GRG CALLS	GRG RATIO	LOCALS FOUND	FCN CALLS TO BEST	TOTAL FCN CALLS	FCN RATIO	TIME TO BEST	TOTAL TIME	TIME RATIO
1 to 4	31	213.9	1.3	9.2	0.14	2.3	281.7	1582.8	0.18	0.5	2.4	0.21
5 to 7	31	212.5	1.3	11.0	0.12	3.0	361.0	3827.5	0.09	0.3	1.1	0.28
8 to 12	21	293.3	3.2	17.2	0.19	6.1	841.5	3416.6	0.25	0.5	1.5	0.35
14 to 21	17	212.8	1.8	25.6	0.07	6.8	446.8	5030.7	0.09	1.7	4.4	0.38
22 to 78	11	310.2	7.3	34.0	0.21	12.4	1423.4	10528.3	0.14	1.5	4.4	0.33
110 to 141	9	392.4	9.3	26.7	0.35	22.6	20654.7	55580.3	0.37	23.6	54.4	0.43
Overall	120	272.5	4.0	20.6	0.18	8.8	4001.5	13327.7	0.19	4.7	11.4	0.33

Average total GRG calls are fairly stable at between 17 to 34 over the last four groups, and do not increase rapidly with problem size. This further demonstrates the effectiveness of the distance and merit filters described in Section 5.

The ratio columns provide additional evidence that the best solution is found early in the iterative process. The smallest of these is the GRG ratio, which varies from 0.07 to 0.35, meaning that the best solution is found in the first 7% to 35% of GRG calls. This ratio is highly correlated with the function call ratio, because function calls due to GRG (all those over 1000) dominate as problem size increases. This implies that, for these problems, a criterion that stops OQNLP when the fractional change in the best feasible objective value found thus far is below a small tolerance for some (reasonably large) number of successive iterations, would rarely terminate before the best solution was found.

Table 2.5 shows the results of using 5000 iterations to solve the 6 problems whose gaps in Table 2.3 are greater than 1%.

All problems are solved to within very small gaps. To achieve an essentially zero gap for problems 2_1_6 and 2_1_7_5, we had to change the OptQuest boundary strategy parameter from its default value of 0.5 to 1.0. This causes a strategy that drives trial points towards the boundary of the feasible region defined by the bounds and linear constraints to be used 100% of the time, rather than 50% (see the OptQuest User guide, page 32). These two problems are quadratic programs with concave objectives (to be minimized), so all locally optimal solutions are at extreme points of the feasible region. One would expect a strategy that generates points near the boundary of the feasible region 100% of the time to be most effective on such problems.

Table 2.5. Solving previously unsolved problems in 5000 iterations

PROB	VARS	ITNS TO BEST	GRG CALLS TO BEST	TOTAL GRG CALLS	LOCALS FOUND	FCN CALLS TO BEST	TOTAL FCN CALLS	TIME TO BEST	TOTAL TIME	GAP	BNDY PARAM
EX8_3_7	126	308	12	164	92	7125	505273	15.4	436.3	0.0009	
EX2_1_1	5	893	2	14	12	896	5055	0.12	0.85	0.0000	
EX2_1_6	11	253	2	8	4	256	5042	0.73	8.17	15.0000	0.5
EX2_1_6	11	2591	18	27	12	2703	5171	5.64	8.92	0.0000	1
EX2_1_8	24	2318	32	44	11	3411	6270	6.8	16.91	0.0000	
EX2_1_7_5	21	362	9	36	22	975	7360	3.7	37.7	1.0866	0.5
EX2_1_7_5	21	520	23	90	15	2397	11341	7.18	52.23	0.0002	1
EX14_1_7	11	299	9	220	64	6518	119049	0.51	11.1	0.0000	

Solving Problems with Discrete Variables

Table 2.6 contains results of solving the 11 problems in the Floudas test set which have discrete variables, using 1000 total and 200 stage one iterations. The problems are sorted first by number of discrete variables, then by number of all variables.

Table 2.6. Solution statistics for 13 problems with discrete variables

PROB	VARS	DISC VARS	LIN CONST	NLIN CONST	ITNS TO BEST	GRG CALLS TO BEST	TOTAL GRG CALLS	TOTAL ENUM	LOCAL FOUND	FCN CALLS TO BEST	TOTAL FCN CALLS	TIME TO BEST	TOTAL TIME	GAP, %
EX12_2_2	4	1	2	1	201	1	1	2	1	209	1008	0.53	2.04	0.0000
EX12_2_1	6	3	3	2	249	6	18	8	7	296	1164	0.54	1.25	0.0000
EX12_2_6	6	3	4	1	201	1	7	6	2	202	1045	0.31	0.4	0.0000
EX9_2_9	13	3	11	1	201	1	3	8	3	202	1010	0.42	1.73	0.0000
EX12_2_3_N	8	4	5	4	441	17	20	16	10	1004	1667	0.85	1.25	0.0000
EX12_2_3	12	4	9	4	390	11	20	16	10	1016	1761	1.11	1.96	0.0000
EX9_1_9	18	5	16	1	201	1	21	32	18	202	1069	2.41	8.01	0.0024
EX12_2_5	9	6	9	1	201	1	6	25	1	202	1136	0.91	2.65	0.0000
EX9_1_6	21	6	19	1	201	1	32	64	27	202	1097	1.53	6.73	0.0000
EX9_1_7	24	6	21	1	204	4	41	64	31	256	1328	4	9.64	0.0000
EX9_1_3	30	6	27	1	207	2	42	64	23	214	1212	2.49	11.1	0.0000
EX12_2_4	12	8	4	3	924	78	88	256	16	2145	2501	3.67	4.06	5.4320
EX12_2_4N	12	8	4	3	207	3	40	256	13	218	1191	0.87	2.96	7.1918
averages	13.5	4.8	10.3	1.8	294.5	9.8	26.1	26.1	12.5	489.8	1322	1.5	4.1	1.0

All problems are solved to very small gaps except 12_2_4 and its reformulation, 12_2_4N, which have the same optimal solutions, and have final gaps of 5.4% and 7.2% respectively. Increasing the number of iterations to 5000 or 10,000 does not yield better solutions for these 2 problems. For the other 11 problems, 7 are solved on the first or second GRG call. The column headed “total enum” contains the number of GRG calls needed to solve the problem by complete enumeration of all integer combinations. The total number of GRG calls used by OQNLP is larger than this value in 3 of the 13 problems, and the two averages are about the same. However, the number of GRG calls to find the best solution is larger than that for complete enumeration in only one instance, and the average is 9.8 versus 26.1 for complete enumeration. As with the continuous variable problems, the best solutions are found in roughly the first 30% of the 1000 iterations on average.

Clearly, the number of discrete variables in these problems is too small to infer whether or not this “base-case” OQNLP algorithm will be competitive with alternative MINLP solvers like DICOPT (interfaced to GAMS) or branch-and-bound (Biegler, et al., 1997, Floudas, 1995). We believe that OQNLP performance with discrete variables can be significantly enhanced by sending information on GRG solutions back to OptQuest. For example, an option that begins by calling GRG to solve a relaxed MINLP (with all discrete variables allowed to be continuous), could terminate immediately if all discrete variables had discrete values in the GRG solution. Otherwise, the discrete variables could be rounded,

and the resulting high quality solution could be returned to OptQuest, influencing the generation of successor trial points.

In 12_2_3 and 12_2_4 the discrete variables appear linearly. (This is required by the widely used DICOPT MINLP solver.) Since OQNLP allows discrete variables to appear nonlinearly, we reformulated these problems into 12_2_3N and 12_2_4N, respectively, where the discrete variables appear nonlinearly. The resulting models have the advantage that when the discrete variables are fixed for the NLP solver, the continuous variables appear linearly. That is why all measures of computational effort are much smaller for the reformulated versions.

For comparison purposes we ran the 11 MINLP problems from the Floudas problem set using DICOPT, with CONOPT2 as the NLP solver and CPLEX as the MILP solver. It solves a NLP and a MILP at each major iteration. The only changes to the models were slight adjustments to lower bounds on some variables to avoid numerical problems encountered otherwise. The statistics are shown in Table 2.7 below, with some OQNLP data repeated for easier comparison. DICOPT solved all but one of the problems to the best-known solution. In the case of EX_12_2_1 the DICOPT NLP Solver was unable to find a feasible solution for the relaxed NLP, and the problem was incorrectly diagnosed as infeasible. Five of the problems are MILP's, and DICOPT simply invokes CPLEX to solve them. For the other five problems, DICOPT found the optimum in 2 or 3 major iterations. Its runtimes are much shorter than OQNLP, and the number of NLP solver calls is much less. Termination was caused by the NLP solver objective worsening, an infeasible MILP, and the relaxed NLP having an integer solution.

It is difficult to infer much from such small problems. However, we expect that DICOPT will be much faster than OQNLP when it succeeds. As a primal method, OQNLP has the potential to find a good solution in cases where DICOPT fails to find a feasible integer solution, and it may sometimes be useful to study the multiple integer feasible solutions that OQNLP can provide.

Varying the Length of Stage One

We have solved the 120 Floudas problems with no discrete variables with three values for the number of stage one iterations: 200 as described above, 300, and as many as are required to generate all "first generation" trial points (those created from the initial population), called the "1gen" strategy. With 1gen and 200 stage one iterations, there were 1000 total iterations, while with 300 we used 1100 total, in order to provide at least 800 iterations in stage 2 for all strategies. The averages for various measures of computational effort and achievement over all 120 problems for these three stage one strategies are shown in Table 2.8 below. The column headed "probs < 1%", shows the number of problems solved to a gap of 1% or less by the strategy, while the last column gives the number

of these successful runs where the best solution was found at the first or second GRG call.

Table 2.7. OQNLP and DICOPT results for MINLP problems

	PROB	OQNLP: GRG CALLS	OQNLP: TOTAL RUNTIME	OQNLP: TIME TO BEST SOL.	OQNLP: % GAP TP BEST KNOWN SOL.	DICOPT ITNS	DICOPT TERMINATION	DICOPT: RUNTIME	DICOPT: % GAP TP BEST KNOWN SOL.
EX12_2_2	1	2.04	0.53	0.00	2	infes mip		2.26	0.00
EX12_2_1	18	1.25	0.54	0.00	F	nlp 1 infes		0.06	F
EX12_2_6	7	0.4	0.31	0.00	0	Relaxed nlp integer		0.05	0.00
EX9_2_9	3	1.73	0.42	0.00	0	milp		0.05	0.00
EX12_2_3	20	1.96	1.11	0.00	3	worsen		0.48	0.00
EX9_1_9	21	8.01	2.41	0.00	0	milp		0.11	0.00
EX12_2_5	6	2.65	0.91	0.00	3	worsen		0.4	0.00
EX9_1_6	32	6.73	1.53	0.00	0	milp		0.11	0.00
EX9_1_7	41	9.64	4	0.00	0	milp		0.28	0.00
EX9_1_3	42	11.1	2.49	0.00	0	milp		0.11	0.00
EX12_2_4	88	4.06	3.67	5.43	3	worsen		0.45	0.00

Table 2.8. Effects of varying the number of stage one iterations

STAGE 1 ITNS	ITNS TO BEST	GRG CALLS TO BEST	TOTAL GRG CALLS	LOCALS FOUND	FCN CALLS TO BEST	TOTAL FCN CALLS	TIME TO BEST	TOTAL TIME	PROBS <1%	PROBS IN 1 OR 2 GRG CALLS
163.4	235.6	5.1	27.4	12.6	2127.2	13263.2	5.0	15.9	114	(78,11)
200	272.5	4.0	20.6	8.8	4001.5	13327.7	4.7	11.4	114	(87,7)
300	392.4	5.0	23.0	12.1	6549.9	17672.4	8.1	16.4	109	(90,2)

Examining this table, we see that the iterations and function calls to find the best solution increase with the number of stage one iterations. This is as expected, since the first GRG call comes at the end of stage one, whose purpose is to provide a high quality starting point for GRG.

However, several other effort measures show a minimum at 200 initial iterations: grg calls to best, total grg calls, time to best, and total time. The number of problems solved in one or two GRG calls is also maximized for 200 initial iterations, although the differences between the three strategies are small. Since both the “1gen” and “200” strategies have the same number of problems solved to within a 1% gap, these results imply a mild preference for the (200,1000) strategy.

The benefits of starting stage 2 earlier are: (1) the best solution is often found earlier, since the first GRG call usually finds the best solution, and (2) trial points which would be skipped in a longer stage one are eligible to be GRG starting points, and can lead to good GRG solutions. Since the population loses diversity as it is updated by the aggressive update currently used in OptQuest, these missed opportunities may not recur before the population is reinitialized. The advantages of a longer stage one are: (1) The best point found by OptQuest in a longer stage one should, on average, have higher quality than in a shorter one, which leads to somewhat better results on the first GRG call, and (2) these higher quality best points should have lower values for the exact penalty function, P_1 , which becomes the initial value for the merit filter threshold. This lower value leads to fewer GRG calls in stage 2, as shown in Table 2.8. The number of GRG calls is also influenced by other factors, so the effect is not monotonic. We believe that the superior performance of the (200,1000) strategy is due to its achieving a best balance between these competing effects.

8. Minimizing the Potential Energy of a Cluster of Particles

The Floudas set of test problems includes two GAMS models that minimize the potential energy of a cluster of N particles, using two different potential energy functions. The decision variables are the x , y , and z components of each particle. For the Lennard-Jones family of potential energy minimization problems the objective is the summed difference between the sixth and third powers of the reciprocal of the squared Euclidean distance between each distinct pair of particles, where the sixth power term arises from a strong short-range repulsive force and the other term from a longer-range attractive force. Nonlinear constraints are included to avoid objective function singularities where the distance between one or more pairs of points is very small. If they are not included, GAMS encounters many thousands of domain violations-these still occur in the above formulation, but are less frequent. Particle 1 is located at the origin, and three position components of particles 2 and 3 are fixed, so this family of problems has $N-6$ variables and $N(N-1)$ nonlinear constraints.

The second set of problems uses the Morse potential, where the Euclidean distance appears in the argument of an exponential function. There is no need to protect against domain violations here, so there are no

constraints except for upper and lower bounds of 5 and -5 on the coordinates of each particle, as in the Lennard-Jones case. According to Floudas (1999), pp. 186-194, these problems have a large number of local minima, and this number increases rapidly with problem size. Thus they are a rigorous test for global optimization algorithms.

Results of applying OQNLP to these two problem classes using 200 stage one and 1000 total iterations for several values of N are shown in Tables 2.9 and 2.10 below.

Table 2.9. Minimizing the Lennard-Jones potential function

N	VARS	NLIN CONST	ITNS TO BEST	GRG TO BEST	TOTAL GRG	GRG RATIO	LOCALS FOUND	FNC TO BEST	TOTAL FCN	FCN RATIO	TIME TO BEST	TOTAL TIME	TIME RATIO	GAP , %
5	9	10	201	1	16	0.06	8	447	6669	0.07	0.1	1.21	0.08	0.00
10	24	45	437	7	15	0.47	14	6296	13220	0.48	5.07	10.32	0.49	0.00
15	39	105	252	4	17	0.24	17	4775	20641	0.23	9.07	37.39	0.24	0.00
20	54	190	476	11	21	0.52	21	11607	20402	0.57	39.49	69.25	0.57	0.00
25	69	300	478	9	58	0.16	58	13900	109183	0.13	74.87	570.2	0.13	2.71
30	84	435	730	27	48	0.56	48	51221	85873	0.60	399.17	668.8	0.60	1.71
avg	46.5	180.8	429	9.8	29.2	0.3	27.7	14707.7	42664.7	0.3	88.0	226.2	0.4	0.7

Table 2.10. Minimizing the Morse potential function

N	VARS	NLIN CONST	ITNS TO BEST	GRG TO BEST	TOTAL GRG	GRG RATIO	LOCALS FOUND	FNC TO BEST	TOTAL FCN	FCN RATIO	TIME TO BEST	TOTAL TIME	TIME RATIO	GAP, %
5	9	0	201	1	12	0.08	9	328	2666	0.12	0.05	0.31	0.16	0.00
10	24	0	201	1	7	0.14	7	663	3420	0.19	0.37	1.66	0.22	0.00
15	39	0	201	1	14	0.07	14	687	7826	0.09	0.9	8.32	0.11	0.00
20	54	0	284	4	23	0.17	23	1593	10427	0.15	3.42	20	0.17	0.00
25	69	0	300	3	59	0.05	59	2125	31294	0.07	7.2	88.67	0.08	0.00
30	84	0	268	4	41	0.10	41	3196	28385	0.11	15.23	117.2	0.13	0.00
40	114	0	476	12	51	0.24	51	9848	41782	0.24	76.89	306	0.25	0.00
50	144	0	262	4	11	0.36	11	4233	11824	0.36	56.39	151.6	0.37	0.14
avg			274.13	3.75	27.25	0.15	26.88	2834.13	17203	0.17	20.06	86.72	0.19	0.02

OQNLP finds the Morse potential easier to minimize, and solves the 7 smallest instances to essentially zero gaps and the N=50 case to a .14% gap. The 4 smallest instances of the Lennard-Jones problems are also solved to near-zero gaps, but the two largest have gaps of 2.7% and 1.7% respectively. We attribute this partially to the occurrence of many domain violations during the GRG runs. The N=25 run had 93,926

divides by zero and 6717 integer power overflows, while the corresponding figures for N=30 were 133,490 and 9551.

The computational effort needed to achieve these excellent results is quite modest. As before, the ratio columns are the effort to find the best solution divided by total effort, and these ratios are generally less than 0.3 for the Morse potential, but occasionally above 0.5 for the Lennard-Jones. Both function calls and GRG calls to achieve the best solution are quite small, and, for the Morse function, they do not increase rapidly with N. The number of local minima found increases rapidly with N, and is around 60 for N=30 or above. This number is usually equal to the number of GRG calls, so GRG almost always finds a different local solution at each start.

Results of solving the two Lennard-Jones problems with gaps larger than 1% using 200 stage one and 5000 total iterations are shown in Table 2.11 below

Table 2.11. Solving with 5000 iterations

N	VARS	NLIN CONST	ITNS TO BEST	GRG TO BEST	TOTAL GRG	GRG RATIO	LOCALS FOUND	FCN TO BEST	TOTAL FCN	FCN RATIO	TIME TO BEST	TOTAL TIME	TIME RATIO	GAP, %	BND
25	69	300	4089	339	436	0.78	436	664754	820883	0.81	3472.6	4290	0.81	0.00	5
30	84	435	240	5	33	0.15	33	9856	69471	0.14	80.11	554.8	0.14	3.18	5
30	84	435	2486	162	414	0.39	414	188479	432148	0.44	1496	3405	0.44	0.67	3

The N=25 problem is solved to a near-zero gap, but the gap for the N=30 problem (row 2 of the table) actually increases from 1.71% with 1000 iterations to 3.18% with 5000. This is because some aspects of the OptQuest solution strategy depend on the iteration limit, so the two runs use a different sequence of trial points in their first 1000 iterations. The search is more aggressive when there are only 1000 iterations allowed, and this aggressiveness leads to a better final solution in the shorter run. However, if the parameter bnd (each variable has bounds of (-bnd,bnd)) is decreased from 5 to 3, the gap for 5000 iterations decreases to 0.67%, showing the benefits of searching within a smaller rectangle. The computational effort to achieve these improved outcomes, compared to the shorter runs, increases roughly by factors of 5 to 8.

9. Comparison with Random Starts

OptQuest was chosen as the provider of starting points because we felt it would find good points quickly. As a first step to investigating this, we selected the Morse and Lenard_jones potential functions described in section 8, generated either 100 or 200 independent uniformly distributed starting points, started CONOPT2 from each of these, and observed how

many calls found the best known solution, as well as how many distinct local solutions were found. This was done with a LOOP statement in the GAMS models discussed in section 8, and compared with the number of OQNLP calls required before CONOPT2 found the best solution. The results provide a crude estimate of the relative volume of the basin of attraction of the global optimum, as the ratio $f = \text{nglob}/\text{ncalls}$, where nglob is the number of NLP solver calls leading to the global solution. The expected number of calls before the global solution is first located is simply $1/f$. Results are shown in Tables 2.12 and 2.13 below. Since CONOPT2 is used rather than LSGRG2, the number of OQNLP calls to find the best solution differ from those in Tables 2.9 and 2.10.

Table 2.12. Morse potential function, random starts

ATOMS	VARS	CALLS	NGLOB	DISTINCT LOCALS	EXP CALLS TO BEST	OQNLP CALLS TO BEST
5	9	200	13	17	15.4	1
10	24	200	1	107	200	1
15	39	200	9	161	22.2	1
20	54	200	10	189	20	2
25	69	200	2	185	100	4
30	84	200	2	188	100	17
40	114	200	3	191	66.7	7
50	144	200	3	181	66.7	20

Table 2.13. Lennard-Jones potential, random starts

ATOMS	VARS	CALLS	NGLOB	DISTINCT LOCALS	EXP CALLS TO BEST	OQNLP CALLS TO BEST
5	9	100	99	2	1.0	1
10	24	100	4	27	25.0	21
15	39	100	3	85	33.3	6
20	54	200	2	167	100	67

For both problems, OQNLP finds the best solution in fewer solver calls than the expected number of calls to the best solution for random starts, much fewer for the Morse potential function. For both problem sets, the relative volume estimate f decreases quickly as problem size increases, and many more than 200 solver calls are needed to estimate it accurately. For the larger numbers of atoms, almost every solver call leads to a different local minimum.

10. Summary and Future Research

While the performance of this “base case” OQNLP algorithm on problems with only continuous variables is quite good, there are options which promise improvements. OptQuest’s search is usually more efficient when the initial population contains high quality solutions. No such solutions are supplied in the computational experiments described here. A way to provide one good solution is to call GRG at the start of stage one, communicating the local optimum found to OptQuest as a possible member of its initial population. GRG’s starting point could be either user-provided, or the best point found in some initial set of OptQuest iterations (perhaps a few hundred as in the current stage one). The latter option is equivalent to adding a new stage one consisting of these initial OptQuest iterations, calling GRG from the best stage one solution (stage 2), and doing a stage 3 by placing the GRG solution as a candidate point in a newly initialized population, before performing another set of OptQuest iterations. In our computational experiments, the GRG solution resulting from a start at the best point from 200 or so OptQuest iterations is globally optimal in about 75% of the problems solved. Hence the initial stage 3 population would often contain the global optimum, plus points diverse from it. The final stage (4) would be the current stage 2, where GRG is called repeatedly at points which are accepted by the distance and merit filters. We are currently implementing this option.

The above idea is naturally extended by communicating other GRG solutions to OptQuest during (the current) stage 2, but this must be done in a way that maintains enough diversity in the population. Hence only unique local solutions should be sent to OptQuest, and these should not be too close to one another. Some distance threshold must be devised, and OptQuest would receive only local solutions whose distance from the nearest previously found local solution is greater than the threshold.

In problems with discrete variables, high quality points for OptQuest’s initial population can be determined by solving the relaxed MINLP, where all discrete variables are allowed to be continuous within their bounds. If all discrete variables take on allowed values, this solution is at least locally optimal. If not, various rounding procedures can be applied to it to generate one or more high quality discrete solutions. We are currently implementing this option as well.

Another promising option for MINLP’s is to “hide” the continuous variables from OptQuest, which searches only over the space of discrete variables. It is aware only of the constraints involving only discrete variables. This allows it to focus its attention on these key variables, which GRG cannot vary. That is, OptQuest is applied to the projection of the problem (1.1)-(1.4) onto y -space. This projected problem is to minimize

$$F(y) = \min_x (f(x, y) \mid gl \leq G(x, y) \leq gu, l \leq A_1x + A_2y \leq u, x \in S)$$

over all constraints involving only y . GRG is then applied to the x subproblem on the right hand side of the above equation. As is done currently, GRG would fix the discrete variables at values specified by OptQuest, and optimize over the continuous variables. If GRG finds a feasible solution, its optimal objective value is returned to OptQuest. If not, the exact penalty function value of this solution is returned, using some set of sufficiently large penalty weights. These GRG solutions should be of much higher quality than the continuous variable values generated by OptQuest in the current algorithm, where the continuous variables are nowhere near locally optimal for the associated discrete variables.

Comparative tests of OQNLP and alternative global optimization methods are also needed. GAMS Development Company has interfaced several global and MINLP solvers, including OQNLP, to GAMS, and this will make the comparison process much easier by providing a common computing environment and model base. The model base has been expanded by a website recently introduced by GAMS Development Company called "GAMS World" at www.gamsworld.org (see ad on the back cover of ORMS Today, Aug 2001). This is divided into "MINLP world" and "Global world". MINLP world currently contains a set of MINLP test problems coded in GAMS, facilities for converting models in several other algebraic modeling languages to GAMS models, and other information on MINLP. Global world contains similar information for global optimization. We also plan comparative tests with the global optimizers in Frontline Systems Premium Excel Solver.

Appendix

Detailed computational results for 120 Floudas test problems.

Table 2.3. Results for Floudas GAMS problems with no discrete variables

PROBLEM NAME	VARS	LIN CONST	NONLIN CONST	INIT ITNS	ITNS TO BEST	GRG CALLS TO BEST	TOTAL GRG CALLS	LOCALS FOUND	FCN CALLS TO BEST	TOTAL FCN CALLS	TIME TO BEST (SEC)	TOTAL TIME (SEC)	MAX ABS X COMP	GAP, %
EX4_1_1	1	0	0	200	201	1	5	1	211	1139	0.17	0.61	1.19	0.000
EX4_1_2	1	0	0	200	201	1	6	1	211	1208	0.22	0.99	1.09	0.000
EX4_1_3	1	0	0	200	201	1	4	1	212	1068	0.17	0.5	6.33	0.000
EX4_1_4	1	0	0	200	201	1	8	2	202	1090	0.16	0.49	2	0.000
EX4_1_6	1	0	0	200	201	1	7	2	212	1119	0.17	0.55	3	0.000
EX4_1_7	1	0	0	200	201	1	12	1	207	1190	0.16	0.71	1	0.000

Table 2.3. Results for Floudas GAMS problems with no discrete variables (cont'd)

PROBLEM NAME	VARS	LIN CONST	NONLIN CONST	INIT ITNS	ITNS TO BEST	GRG CALLS TO BEST	TOTAL GRG CALLS	LOCALS FOUND	FCN CALLS TO BEST	TOTAL FCN CALLS	TIME TO BEST (SEC)	TOTAL TIME (SEC)	MAX ABS X COMP	GAP, %
EX8_1_2	1	0	0	200	201	1	15	3	208	1161	0.16	0.71	5.17	0.010
EX14_1_9	2	0	2	200	201	1	3	2	215	1069	0.22	0.6	347.31	0.000
EX4_1_5	2	0	0	200	201	1	8	2	202	1113	0.17	0.61	1.75	0.000
EX4_1_8	2	0	1	200	201	1	3	3	839	1392	0.66	0.94	3	-0.001
EX4_1_9	2	0	2	200	251	4	9	4	331	1186	0.28	0.66	3.44	-0.001
EX8_1_1	2	0	0	200	201	1	6	2	210	1110	0.16	0.6	2	0.000
EX8_1_3	2	0	0	200	201	1	2	1	226	1054	0.17	0.44	1	0.000
EX8_1_4	2	0	0	200	201	1	11	1	202	1154	0.16	0.66	0	0.000
EX8_1_5	2	0	0	200	201	1	6	2	218	1134	0.17	0.61	0.71	0.000
EX8_1_6	2	0	0	200	205	2	6	3	231	1098	0.22	0.55	8	0.000
EX14_1_1	3	0	4	200	201	1	7	4	223	1367	0.28	0.93	3.39	0.003
EX14_1_3	3	0	4	200	201	1	12	6	216	2016	0.17	1.6	6.94	0.002
EX14_1_4	3	0	4	200	201	1	9	4	235	1633	0.27	1.26	3.14	0.000
EX3_1_4	3	2	1	200	201	1	13	3	202	1329	0.39	1.93	3	0.000
EX6_2_11	3	1	0	200	201	1	9	2	229	1529	1.15	5.66	0.99	0.000
EX6_2_6	3	1	0	200	201	1	10	2	211	1379	1.1	5.11	0.94	0.000
EX6_2_8	3	1	0	200	201	1	23	3	235	1893	1.26	6.37	0.97	-0.001

Table 2.3. Results for Floudas GAMS problems with no discrete variables (cont'd)

PROBLEM NAME	VARS	LIN CONST	NONLIN CONST	INIT TNS	ITNS TO BEST	GRG CALLS TO BEST	TOTAL GRG CALLS	LOCALS FOUND	FCN CALLS TO BEST	TOTAL FCN CALLS	TIME TO BEST (SEC)	TOTAL TIME (SEC)	MAX ABS X COMP	GAP, %
EX14_1_8_N	3	0	0	200	548	6	9	2	726	1256	0.54	0.98	0.65	0.013
EX6_1_2	4	1	2	200	201	1	8	2	233	1890	0.82	4.06	1	0.006
EX6_2_12	4	2	0	200	201	1	14	4	232	1716	0.99	5.06	0.5	0.000
EX6_2_14	4	2	0	200	201	1	17	2	207	1221	0.66	3.63	0.5	-57.797
EX6_2_9	4	2	0	200	201	1	23	2	260	2885	1.05	7.03	0.5	0.000
EX7_3_1	4	6	1	200	201	1	10	1	805	8361	1.31	11.59	1073.39	0.003
EX7_3_2	4	6	1	200	201	1	4	1	381	1288	1.53	5.43	1.28	0.000
EX9_2_8	4	3	2	200	201	1	5	1	202	1020	0.44	2.64	1	0.000
averages	2.5	0.9	0.8	200.0	213.9	1.3	9.2	2.3	281.7	1582.8	0.5	2.4	47.9	-1.9
EX14_2_1	5	1	6	200	201	1	13	1	247	1812	0.34	1.11	54.25	0.000
EX14_2_4	5	1	6	200	201	1	13	1	271	1990	0.38	1.22	72.97	0.000
EX14_2_6	5	1	6	200	201	1	9	1	245	1549	0.38	1.16	61.59	0.000
EX14_2_8	5	1	4	200	201	1	1	1	230	1029	0.29	0.78	55.73	0.000
EX2_1_1	5	1	0	200	201	1	4	4	202	1007	0.05	0.16	1	2.778
EX3_1_2	5	0	6	200	201	1	8	1	235	1364	0.04	0.14	78	-0.002
EX7_3_3	5	6	2	200	201	1	7	2	254	11992	0.36	2	2.81	0.002
EX8_1_7	5	0	5	200	291	3	7	3	905	3191	0.06	0.16	2.84	0.001
EX8_5_3	5	3	2	200	201	1	8	3	223	1393	0.39	1.24	0.98	-0.013
EX8_5_4	5	3	2	200	201	1	12	4	303	2004	0.43	1.53	0.78	-0.006
EX8_5_5	5	3	2	200	346	4	10	3	692	2066	0.53	1.41	0.8	-0.345
EX14_1_2	6	0	9	200	201	1	20	1	480	6125	0.05	0.33	31.33	0.000
EX14_1_5	6	4	2	200	201	1	5	2	202	1084	0.47	1.48	1.42	0.000
EX14_2_2	6	1	4	200	201	1	9	1	222	1207	0.34	0.86	58.13	0.000
EX14_2_5	6	1	4	200	201	1	12	1	225	1320	0.37	1	77.19	0.000
EX14_2_7	6	1	8	200	201	1	5	1	233	1229	0.61	2.02	63.56	0.000

Table 2.3. Results for Floudas GAMS problems with no discrete variables (cont'd)

PROBLEM NAME	VARS	LIN CONST	NONLIN CONST	INIT TNS	ITNS TO BEST	GRG CALLS TO BEST	TOTAL GRG CALLS	LOCALS FOUND	FCN CALLS TO BEST	TOTAL FCN CALLS	TIME TO BEST (SEC)	TOTAL TIME (SEC)	MAX ABS X COMP	GAP, %
EX14_2_9	6	1	4	200	201	1	12	1	219	1253	0.36	1.08	60.63	0.000
EX2_1_2	6	2	0	200	201	1	16	1	202	1246	0.16	0.63	20	0.000
EX2_1_4	6	5	0	200	201	1	8	1	208	1116	0.35	0.76	6	0.000
EX3_1_3	6	4	2	200	201	1	9	5	202	1062	0.18	0.5	10	0.000
EX6_1_4	6	1	3	200	201	1	4	2	546	1974	0.45	1.25	1	-0.004
EX6_2_10	6	3	0	200	201	1	6	1	242	1426	0.46	2.01	0.4	-0.557
EX6_2_13	6	3	0	200	201	1	16	13	202	1695	0.45	2.28	0.62	-0.001
EX7_2_2	6	0	5	200	201	1	9	6	242	1458	0.04	0.14	11.02	0.000
EX8_1_8	6	0	5	200	201	1	9	6	242	1458	0.04	0.14	11.02	-0.001
EX8_5_1	6	3	2	200	224	3	13	11	559	3437	0.46	1.77	0.83	0.931
EX8_5_2	6	2	2	200	282	2	3	2	591	1860	0.47	1.75	0.69	-0.123
EX8_5_6	6	2	2	200	201	1	8	1	283	1661	0.48	1.64	0.67	0.003
EX14_2_3	7	1	8	200	201	1	3	1	240	1150	0.5	1.58	57.16	0.000
EX5_2_4	7	3	3	200	201	1	50	1	335	7541	0.24	0.98	100	0.000
EX7_2_1	7	2	12	200	218	3	31	10	1709	49954	0.32	2.29	3031.6	-2.176
averages	5.7	1.9	3.7	200.0	212.5	1.3	11.0	3.0	361.0	3827.5	0.3	1.1	125.0	0.0
EX3_1_1	8	3	3	200	201	1	11	1	1084	8837	0.29	0.74	5109.9	-0.001
EX5_4_2	8	3	3	200	201	1	11	1	539	4761	0.24	0.75	5485.3	0.000
EX6_1_1	8	2	4	200	872	12	14	10	1622	1762	0.66	0.76	1	0.246
EX7_2_3	8	3	3	200	291	5	5	3	2388	3097	0.33	0.8	5110.2	0.003
EX7_2_4	8	0	4	200	657	6	12	6	3586	8191	0.17	0.33	9.81	-0.673
EX9_2_4	8	5	2	200	201	1	34	1	210	1357	0.32	1.28	3	0.000
EX9_2_5	8	4	3	200	201	1	2	1	211	1019	0.55	1.64	7	0.000
EX14_1_6	9	1	14	200	201	1	13	4	290	2822	0.57	1.96	1	0.000
EX6_2_5	9	3	0	200	519	16	22	20	1961	3205	1.52	2.65	31.46	-0.003
EX6_2_7	9	3	0	200	201	1	14	3	272	2373	0.51	2.25	0.45	-0.004
EX14_1_7	11	0	17	200	209	2	22	13	1099	10871	0.14	1.02	10	13.499
EX2_1_5	10	11	0	200	201	1	85	1	266	5570	0.84	3.4	1	0.002
EX2_1_6	11	0	5	200	253	2	3	2	256	1007	0.77	1.85	1	15.000
EX2_1_9	11	1	0	200	201	1	11	3	212	1333	0.31	1.09	0.33	0.000
EX9_1_2	10	5	4	200	201	1	4	1	202	1013	0.37	1.27	4	0.000

Table 2.3. Results for Floudas GAMS problems with no discrete variables (cont'd)

PROBLEM NAME	VARS	LIN CONST	NONLIN CONST	INIT ITNS	ITNS TO BEST	GRG CALLS TO BEST	TOTAL GRG CALLS	LOCALS FOUND	FCN CALLS TO BEST	TOTAL FCN CALLS	TIME TO BEST (SEC)	TOTAL TIME (SEC)	MAX ABS X COMP	GAP, %
EX9_1_4	10	5	4	200	201	1	2	1	202	1009	0.54	1.58	24	0.000
EX9_2_1	10	5	4	200	293	2	17	15	296	1038	0.68	2.02	16.38	0.000
EX9_2_2	10	7	4	200	201	1	31	3	207	1216	0.36	1.36	20	-0.021
EX9_2_7	10	5	4	200	293	2	17	15	296	1038	0.69	1.95	16.38	0.000
EX6_1_3	12	0	9	200	361	8	24	23	1679	4837	1.07	2.26	1	0.354
EX7_3_4	12	0	17	200	201	1	7	1	793	5393	0.04	0.25	731.97	0.000
averages	9.5	3.1	5.0	200	293.3	3.2	17.2	6.1	841.5	3416.6	0.5	1.5	789.8	1.4
EX2_1_3	14	9	0	200	201	1	13	5	202	1199	0.31	0.9	3	0.000
EX9_1_1	14	7	5	200	201	1	1	1	211	1010	0.72	2.69	14	0.000
EX9_1_5	14	7	5	200	201	1	12	12	202	1023	0.34	1.33	50.59	0.000
EX8_4_6	14	0	8	200	263	3	52	28	3293	44031	0.23	2.17	10	0.001
EX9_1_10	15	7	5	200	201	1	27	9	209	1197	0.32	1.41	100	0.000
EX9_1_8	15	7	5	200	201	1	27	9	209	1197	0.35	1.49	100	-54.545
EX8_4_5	16	0	11	200	201	1	14	2	341	2704	0.1	0.3	0.23	0.000
EX5_4_3	17	9	4	200	340	12	58	2	789	3762	0.77	1.95	310	0.001
EX9_2_3	17	9	6	200	201	1	5	1	202	1051	0.7	2.36	30	0.000
EX9_2_6	17	6	6	200	201	1	8	1	202	1051	0.37	1.28	1	0.000
EX8_4_4	18	0	12	200	201	1	71	1	293	8462	0.07	0.67	5.13	0.000
EX2_1_10	21	10	0	200	201	1	79	3	283	9854	0.73	2.08	66.35	0.000
EX2_1_7_1	21	10	0	200	201	1	10	6	217	1474	4.91	12.62	28.8	0.000
EX2_1_7_2	21	10	0	200	201	1	20	13	217	2245	4.92	10.62	28.8	0.000
EX2_1_7_3	21	10	0	200	201	1	18	10	217	2078	4.84	10.56	28.8	0.000
EX2_1_7_4	21	10	0	200	201	1	12	7	217	1600	4.99	12.43	28.8	0.000
EX2_1_7_5	21	10	0	200	201	1	9	6	291	1584	3.66	9.95	28.8	1.087
averages	17.5	7.1	3.9	200	212.8	1.8	25.6	6.8	446.8	5030.7	1.7	4.4	49.1	-3.1

Table 2.3. Results for Floudas GAMS problems with no discrete variables (cont'd)

PROBLEM NAME	VARS	LIN CONST	NONLIN CONST	INIT ITNS	ITNS TO BEST	GRG CALLS TO BEST	TOTAL GRG CALLS	LOCALS FOUND	FCN CALLS TO BEST	TOTAL FCN CALLS	TIME TO BEST (SEC)	TOTAL TIME (SEC)	MAX ABS X COMP	GAP, %
EX8_4_1	22	0	10	200	201	1	8	1	257	1774	0.08	0.26	7.5	0.001
EX5_3_2	22	7	9	200	201	1	42	1	471	13484	0.56	1.58	300	0.000
EX2_1_8	24	10	0	200	605	15	20	6	1041	1506	1.6	2.5	24	27.391
EX8_4_2	24	0	10	200	201	1	4	1	300	1400	0.1	0.3	7.45	-0.004
EX5_4_4	27	13	6	200	564	29	75	6	5346	14621	2.15	4.17	200	0.002
EX5_2_5	32	8	11	200	556	21	50	43	4550	15865	2.15	4.21	200	0.000
EX8_4_3	52	0	25	200	201	1	10	1	260	3176	0.36	1.42	4.51	0.000
EX8_2_1	55	6	25	200	222	2	40	7	447	4851	1.44	5.26	236.25	0.001
EX8_2_4	55	6	75	200	201	1	37	7	377	7041	1.74	6.18	216	0.003
EX8_4_7	62	0	40	200	201	1	21	2	340	30603	0.75	5.13	754.96	0.022
EX8_3_9	78	18	27	200	259	7	67	61	2268	21490	5.11	17.32	1000	-0.100
averages	41.2	6.2	21.6	200.0	310.2	7.3	34.0	12.4	1423.4	10528.3	1.5	4.4	268.2	2.5
EX8_3_1	115	17	59	200	307	10	38	28	67165	152573	30.72	81.99	10000	0.29
EX8_3_2	110	27	49	200	945	15	18	18	12254	16103	33.82	35.21	243.85	0.00
EX8_3_3	110	27	49	200	201	1	19	18	2357	28809	8.65	37.17	223.22	0.00
EX8_3_4	110	27	49	200	203	2	23	22	4875	31568	10.49	39.66	105.54	0.00
EX8_3_5	110	27	49	200	201	1	13	11	2442	12746	10.54	39.59	127.28	0.10
EX8_3_6	110	27	49	200	511	14	37	29	6760	56896	20.21	44.97	1000	0
EX8_3_7	126	27	65	200	201	1	8	7	2868	6401	10.99	32.76	100	9.7517
EX8_3_8	126	28	65	200	414	20	45	42	79983	156082	51.32	101.55	10000	0.0004
EX8_3_10	141	43	65	200	549	20	39	28	7188	39045	35.68	76.39	6000	0.3604
averages	117.6	27.8	55.4	200.0	403.1	9.3	25.3	21.9	14840.9	43456.3	22.7	50.9	2225.0	1.3

References

- Biegler, L.T., I.E. Grossman and A.W. Westerberg (1997) *Systematic Methods of Chemical Process Design*, Prentice-Hall, Englewood Cliffs, NJ.
- Brooke, A., D. Kendrick, A. Meeraus and R. Raman (1992) *GAMS: A User's Guide*, Boyd and Fraser, Danvers, MA.
- Dixon, L. and G.P. Szego (1975) "Towards Global Optimization," in *Proceedings of a Workshop at the University of Cagliari, Italy*, North Holland.

- Drud, A. (1994) "CONOPT—A Large-Scale GRG-Code," *ORSA Journal on Computing*, 6(2):207-216.
- Floudas, C. (1995) *Nonlinear and Mixed-Integer Optimization*, Oxford University Press, New York.
- Floudas C.A., P.M. Pardalos, C.S. Adjiman, W.R. Esposito, Z. Gumus, S.T. Harding, J.L. Klepeis, C.A. Meyer and C.A. Schweiger (1999) *Handbook of Test Problems in Local and Global Optimization*, Kluwer Academic Publishers.
- Frontline Systems, Inc. (2000) "Premium Solver Platform User Guide," 95-96.
- Laguna, M. and R. Martí (2000) "Experimental Testing of Advanced Scatter Search Designs for Global Optimization of Multimodal Functions," Working Paper, Department D'Estadistica i Investigacio Operativa, Universitat de Valencia, Burjassot 46100, Spain.
- Laguna, M. and R. Martí (2001) "The OptQuest Callable Library," to appear in *Optimization Software Class Libraries*, Stefan Voß and D. Woodruff, eds., Kluwer Academic Publishers, Boston.
- Locatelli, M. and F. Schoen (1999) "Random Linkage: A Family of Acceptance/Rejection Algorithms for Global Optimization," *Math. Programming*, 85(2): 379-396.
- Murtagh, B.A. and M.A. Saunders (1982) "A Projected Lagrangian Algorithm and Its Implementation for Sparse Nonlinear Constraints," *Mathematical Programming Study*, 16: 84-117.
- Nash, S.G. (1998) "Nonlinear Programming," *OR/MS Today*, 36-45.
- Nash, S.G and A. Sofer (1996) *Linear and Nonlinear Programming*, the McGraw-Hill Companies, Inc.
- Nocedal, J., and S. J. Wright (1999) *Numerical Optimization*, Springer Series in Operations Research.
- Rinnooy Kan, A.H.G. and G.T. Timmer (1987) "Stochastic Global Optimization Methods; part I: Clustering Methods", *Mathematical Programming*, 37: 27-56.
- Rinnooy Kan, A.H.G. and G.T. Timmer (1987) "Stochastic Global Optimization Methods; part II: Multi Level Methods", *Mathematical Programming*, 37: 57-78.
- Smith, S. and L. Lasdon (1992) "Solving Large Sparse Nonlinear Programs Using GRG," *ORSA Journal on Computing*, 4(1): 3-15.

Chapter 3

SCATTER SEARCH METHODS FOR THE COVERING TOUR PROBLEM

Roberto Baldacci¹, Marco A. Boschetti², Vittorio Maniezzo^{3*} and Marco Zamboni³

¹*DISMI, University of Modena and Reggio Emilia, Viale A. Allegri, 15-42100 Reggio Emilia (RE), Italy, baldacci.roberto@unimore.it;*

²*Department of Mathematics, University of Bologna, Cesena, Via Sacchi, 3-47023 Cesena (FC), Italy, boschetti@csr.unibo.it;*

³*Department of Computer Science, University of Bologna, Cesena, Via Sacchi, 3-47023 Cesena (FC), Italy, {maniezzo,zamboni}@csr.unibo.it*

Abstract The Covering Tour Problem (CTP) is a generalization of the Traveling Salesman Problem (TSP) which has several practical applications in the area of distribution network design. Given an undirected graph, the problem asks to identify a minimum cost cycle passing through a subset of vertices such that every vertex not in the cycle lies within a given distance from at least one node in the cycle. Being a generalization of the TSP, CTP is NP-hard. This paper presents three original Scatter Search heuristic algorithms for the CTP. Computational results are reported.

Keywords: Covering Tour Problem, Cutting Planes, Scatter Search Algorithms

1. The Covering Tour Problem

The *Covering Tour Problem* (CTP) is a generalization of the *Traveling Salesman Problem* (TSP) and is defined as follows.

Let $G = (N, E)$ be an undirected graph, where $N = V \cup W$ is the vertex set and E is the edge set. V is the subset of n vertices that *can be visited*, $T \subseteq V$ is the subset of vertices that *must be visited*, while W is the subset of vertices that *must be covered*. Let c_{ij} and d_{ij} , $\{i, j\} \in E$, be the non-negative cost and the distance associated with the edge connecting vertices i and j , respectively.

*Corresponding author

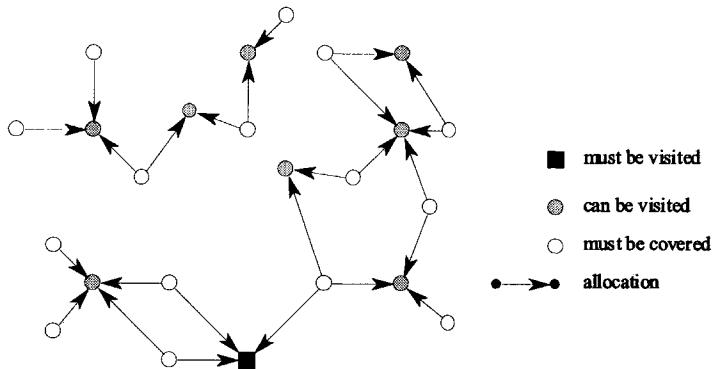


Figure 3.1. Example of CTP instance

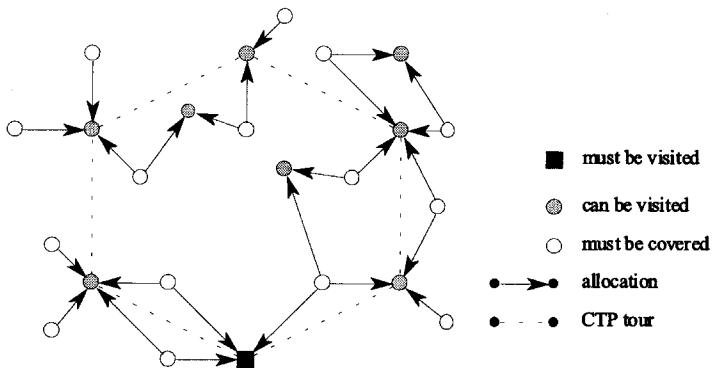


Figure 3.2. Example of CTP tour

The CTP asks for a minimum cost cycle connecting vertices in V such that each vertex $i \in W$ is covered by the cycle. A vertex $i \in W$ is covered if there exists at least one vertex $j \in V$ in the cycle for which $d_{ij} \leq D_j$, where D_j is the *covering distance* associated with vertex j . Figure 3.1 shows a CTP instance where $|T| = 1$, $|V| = 9$ and $|W| = 15$. Figure 3.2 shows a feasible CTP tour of the CTP instance of Figure 3.1.

The CTP is NP-hard as it reduces to the TSP when $D_j = 0$, $\forall j \in V$, and $V = W$.

Current and Schilling, (1989), describe some real world routing problems that correspond to the CTP. An application is the routing of rural health care delivery teams in developing countries, where medical services can only be delivered to a subset of villages, but individuals at villages not directly on the route must be able to reach a visiting medical team at their nearest stop (on this subject, see also Oppong and Hodgson, 1994). Another application is the design of bilevel transportation networks where the tour corresponds to a primary vehicle route and all points not on that route are within easy reach from it (see also Current and Schilling, 1994). For example, this problem arises in the routing of aircraft for overnight delivery system where rather than visit each city directly, it is sufficient for the route to include a stop within some maximal travel distance from each city to be served. Demand at cities not directly on the route would be served by ground transportation linking them to the nearest city on the air route. Another problem is the location of post boxes in a subset of candidate sites in such a way that all users are located within a reasonable distance from a post box and that the cost of a collection route through all post boxes is minimized Labbe and Laporte, (1986). In ReVelle and Laporte, (1993), the CTP is the problem to plan the stops of a circus at a number of locations during a season so that it is always accessible by unvisited populations and it is referred to as the *Traveling Circus Problem*. In this case, any unvisited location may generate a penalty.

Similar but different problems are the *Shortest Covering Path Problem* Current and Schilling, (1994), and the *Tour Cover of a Graph* Arkin et al., (1993), in which weights are on vertices rather than on edges. Moreover, the CTP is related to the *Prize Collecting Traveling Salesman Problem* (PTSP) (see Balas 1989 and Fischetti and Toth, 1988), and to the *Selective Traveling Salesman Problem* (STSP) Laporte and Martello, (1990). Both problems have a non-negative profit p_i associated with each vertex $i \in N$ and consist of constructing a tour through vertex $i \in N$ and a subset of $V \setminus \{i\}$. The PTSP consists of minimizing the length of a tour, which has an associated total profit at least equal to a certain value p . The STSP consists of maximizing the total profit of a tour of length not exceeding a preset value r . A continuous version of our problem is the *Geometric Covering Salesman Problem* (GCSP) (see Arkin and Hassin, 1994) in which the vertex set a priori specified for the covering tour is not discrete, but a region of the plane. Applications of the GCSP include the design of robot routes in polygons under limited visibility restrictions (Ntafos 1992), and aerial forest fire detection (Kourtz 1987).

In the literature, only two heuristic algorithms and one exact method have been presented so far for solving the CTP. Both heuristic procedures, proposed in Current and Schilling, (1989), and Gendreau et al., (1995), are based upon solution procedures for the *Set Covering Problem* (SCP) and the TSP. In particular, Gendreau et al., (1995), combines the GENIUS heuristic for the TSP (see

Gendreau et al., 1992), with the PRIMAL1 set covering heuristic (see Balas and Ho, 1980). The exact method, proposed in Gendreau et al., (1995), is a branch and cut algorithm where at a generic node of the enumeration tree, a linear program containing a subset of valid constraints is solved, a search for violated constraints is made, and some of these constraints are introduced into the current program which is then reoptimized. This process is repeated until a feasible or dominated solution is reached, or until it becomes more promising to branch on a fractional variable.

The CTP can be formulated, with suitable definitions, as a Generalized Traveling Salesman Problem (GTSP) (e.g. see Fischetti et al., 1997).

The paper is structured as follows. Section 2 presents the GTSP. In Section 3 we describe a well known mathematical formulation of the CTP and a new one based on a two-commodity network flow approach. Section 4 introduces the scatter search metaheuristic technique, while Section 5 presents its adaptation to the CTP. Computational results are given in Section 6 and conclusions are reported in Section 7.

2. The Generalized Traveling Salesman Problem

The GTSP is a variant of the TSP where vertices are clustered and it is asked to find a minimum cost cycle which visits at least a vertex of each cluster.

Let $E' = \{(i, j) : i, j \in N, i < j\}$ and let (C_1, \dots, C_m) , with $C_h \subset V$, $1 \leq h \leq m$, be a collection of clusters which induces a partition on V such that $C_1 \cup \dots \cup C_m = V$. Moreover, for each $S \subseteq V$ let $\partial(S) = \{(i, j) \in E' : i \in S, j \notin S \text{ or } j \in S, i \notin S\}$. In the following we will use $\partial(j)$ as a shorthand of $\partial(\{j\})$.

$$(F1) \quad z(F1) = \min \sum_{(i,j) \in E'} c_{ij} x_{ij} \quad (3.1)$$

$$\text{s.t.} \quad \sum_{(i,j) \in \delta(k)} x_{ij} = 2y_k, \quad k \in V \quad (3.2)$$

$$\sum_{j \in C_h} y_j \geq 1, \quad h = 1, \dots, m \quad (3.3)$$

$$\sum_{(h,k) \in \delta(S)} x_{hk} \geq 2(y_i + y_j - 1), \quad \begin{cases} S \subset V, \\ 2 \leq |S| \leq n-2, \\ i \in S, j \in V \setminus S \end{cases} \quad (3.4)$$

$$x_{ij} \in \{0, 1\}, \quad (i, j) \in E' \quad (3.5)$$

$$y_j \in \{0, 1\}, \quad j \in V \quad (3.6)$$

Let x_{ij} be a $(0 - 1)$ binary variable which is equal to 1 if and only if edge $(i, j) \in E'$ is in the optimal cycle and let y_h be a $(0 - 1)$ binary variable equal to 1 if and only if vertex $i \in V$ is visited. A mathematical formulation of the GTSP is as follows Fischetti et al., (1997).

Constraints (3.2) require that the number of edges incident on a vertex k be equal to 2 if k is a visited vertex, 0 otherwise. Constraints (3.3) impose that at least one vertex of each cluster must be visited. Inequalities (3.4) are the subtour elimination constraints, while (3.5) and (3.6) are integrality constraints.

A different version of the problem, called *Equality* GTSP (E-GTSP), arises when imposing the additional constraint that exactly one node of each cluster must be visited. The E-GTSP is obtained by replacing constraints (3.3) with the following equality constraints:

$$\sum_{j \in C_h} y_j = 1, \quad h = 1, \dots, m \quad (3.7)$$

Notice that GTSP and E-GTSP are equivalent when the costs satisfy the triangle inequality: $c_{ij} \leq c_{ik} + c_{kj}, \forall i, j, k \in N$.

To improve the quality of the linear relaxation of F1, denoted LF1, several valid inequalities have been proposed, among which the following Generalized Subtour Elimination Constraints (GSEC) Fischetti et al., (1997):

$$\sum_{(i,j) \in \delta(S)} x_{ij} \geq 2, \quad \text{if } \mu(S) \neq 0, \mu(V \setminus S) \neq 0 \quad (3.8)$$

$$\sum_{(i,j) \in \delta(S)} x_{ij} \geq 2y_k, \quad \text{if } \mu(S) = 0, \mu(V \setminus S) \neq 0, k \in S \quad (3.9)$$

$$\sum_{(i,j) \in \delta(S)} x_{ij} \geq 2(y_h + y_k - 1), \quad \text{if } \mu(S) = 0, \mu(V \setminus S) = 0, h \in S, k \in V \setminus S \quad (3.10)$$

where $\mu(S) = |\{h : C_h \subseteq S\}|$.

3. Mathematical Formulations of the CTP

In this section we present two mathematical formulations of the CTP. The first one has been proposed by Gendreau et al., (1995), while the second is a new integer programming formulation based on a two-commodity approach.

3.1 The Formulation of Gendreau, Laporte and Semet (1995)

The CTP can be formulated as a GTSP as follows. It is possible to define the clusters as follows Gendreau et al., (1995):

$$C_i = \{j \in V : d_{ij} \leq D_j\}, \quad \forall i \in W \quad (3.11)$$

$$C_t = \{t\}, \quad \forall t \in T \quad (3.12)$$

Solving a GTSP on these clusters is equivalent to finding a minimum covering tour on graph G . Note however that this transformation can result in a node being replicated in more clusters. Let m be the number of clusters. Assuming $|C_j| \geq 2, \forall j \in W$, and that a degenerate solution consisting of a single node is not feasible, CTP can be formulated as follows:

$$(F2) \quad z(F2) = \text{Min} \sum_{(i,j) \in E'} c_{ij} x_{ij} \quad (3.13)$$

$$\text{s.t.} \quad \sum_{(i,j) \in \delta(k)} x_{ij} = 2y_k, \quad k \in V \quad (3.14)$$

$$\sum_{j \in C_h} y_j \geq 1, \quad h = 1, \dots, m \quad (3.15)$$

$$\sum_{(h,k) \in \delta(S)} x_{hk} \geq 2y_i, \quad \begin{cases} S \subset V, \\ 2 \leq |S| \leq n-2, \\ i \in S, T \setminus S \neq \emptyset \end{cases} \quad (3.16)$$

$$y_j = 1, \quad j \in T \quad (3.17)$$

$$y_j \in \{0, 1\}, \quad j \in V \setminus T \quad (3.18)$$

$$x_{ij} \in \{0, 1\}, \quad \{i, j\} \in V \quad (3.19)$$

In this formulation, constraints (3.14) require that exactly two edges are incident on each active vertex, while constraints (3.15) ensure that each vertex of W is covered. Inequalities (3.16) are subtour elimination constraints, while (3.17), (3.18) and (3.19) force binary integrality on the variables associated with the edges and with the vertices.

Let $\Delta = [\delta_{ij}]$ be a $|W| \times |V \setminus T|$ (0–1) matrix where entry $\delta_{ij} = 1, i \in W, j \in V \setminus T$, if and only if $d_{ij} \leq D_j$.

Sets W and V can be reduced by iteratively applying the following reduction rules:

- (i) remove from W any vertex i for which $\delta_{ij} = 1, \forall j \in V \setminus T$;
- (ii) remove from W any vertex i for which exists a vertex $j \neq i, j \in W$, such that $\delta_{ik} \leq \delta_{jk}, \forall k \in V \setminus T$;
- (iii) remove from W any vertex i for which exists a vertex $j \in T$ with $d_{ij} \leq D_j$;
- (iv) remove from $V \setminus T$ any vertex i for which $\delta_{ji} = 0, \forall j \in W$.

Let LF2 denote the LP-relaxation of formulation F2, obtained by linearly relaxing constraints (3.18) and (3.19), and with RF2 the problem obtained from LF2 by removing constraints (3.16). The optimal solution cost of RF2 is a valid lower bound to the CTP, which can be improved by means of valid inequalities, for example those proposed by Gendreau et al., (1995), which are in turn extensions of those proposed for the GTSP.

3.2 A Two-Commodity Formulation

In this section we describe a new integer programming formulation of the CTP based on the two-commodity flow formulation originally proposed by Finke et al., (1984), for the TSP. This formulation is interesting in many ways. It can be shown that its LP-relaxation satisfies a weak form of the subtour elimination constraints. The formulation can also be modified to accommodate different constraints, and it is therefore possible to extend it to different routing problems.

The two-commodity formulation has been used by Lucena, (1986), to derive new lower bounds for the *Vehicle Routing Problem* (VRP) and by Langevin et al., (1993), for solving the TSP and the Makespan Problem with Time Windows. Recently, the two-commodity formulation of the TSP has been improved by Baldacci, (1999), and has been used by Baldacci et al., (1999), to derive a new integer programming formulation of the VRP.

The idea behind the two-commodity formulation of the CTP is to associate two flow variables, x_{ij} and x_{ji} , to each edge $\{i, j\} \in E$.

Let us suppose, for instance, that the salesman starts his tour from a vertex $s \in T$. If the salesman travels from i to j , then x_{ij} represents the number of vertices that can still be visited and x_{ji} represents the number of vertices that have already been visited. Thus, the flow variables x_{ij} define two flow circuits for any feasible solution that represent a tour. One circuit is defined by the flow variables representing the number of vertices that can still be visited while the second circuit is defined by the flow variables representing the number of already visited vertices.

Figure 3.3 shows a feasible two-commodity CTP tour of the example of Figure 3.1, where the vertices of the set W have been removed for ease of

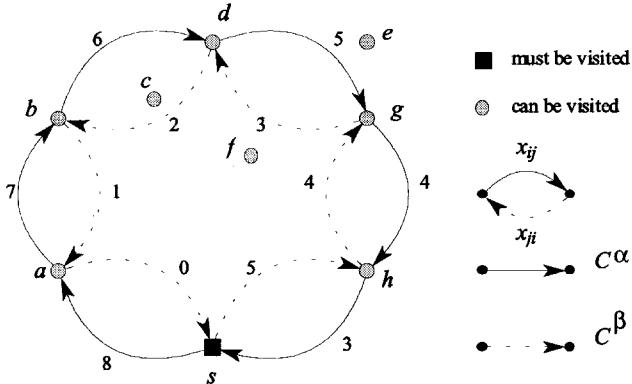


Figure 3.3. Example of Two-Commodity CTP tour

understanding. Circuit C^α is formed by the variables representing the number of vertices that can be visited: the flow $x_{sa} = 8$ indicates that $n - 1 = 8$ vertices can be visited along the tour, $x_{ab} = 7$ indicates the number of vertices that can be visited after visiting vertex a , etc. Circuit C^β is formed by the variables representing the number of vertices visited in the tour. Thus, $x_{hs} = 5$ indicates that the tour has already visited 5 vertices, $x_{hg} = 4$ represents the number of vertices visited by the tour before vertex h , etc. Note that every edge of the tour has $x_{ij} + x_{ji} = n - 1$.

Let ξ_{ij} be a $(0 - 1)$ binary variable equal to 1 if and only if edge $(i, j) \in E$ is in the solution. Let x_{ij} be the flow value of arc (i, j) , $i, j \in V$, $i \neq j$, and let y_i be a $(0 - 1)$ binary variable equal to 1 if and only if vertex $i \in V$ is visited. The new formulation F3 of the CTP is as follows.

Constraints (3.21) and (3.25) define a feasible flow for variables x_{ij} . Constraints (3.22) and (3.24) force the degree of each vertex i that must be visited (i.e. $y_i = 1$) to be equal to 2. Constraints (3.23) ensure that each vertex of W is covered and constraints (3.26) that each vertex of T is visited. Constraints (3.27) and (3.28) are the integrality constraints.

The supply-demand pattern involved, ensure that there are both paths from vertex s to any vertex in V that must be visited and back from any vertex in V that must be visited to vertex s . Hence (3.20)-(3.28) characterize exactly a CTP tour.

Formulation F3 can be written in term of variables $\{x_{ij}\}$ only, since each ξ_{ij} can be replaced by $(x_{ij} + x_{ji})/(n - 1)$ (see equations (3.24)). Hence, the number of variables of the formulation F3 is n^2 and the number of constraints is $2n + m$.

A valid lower bound to the CTP can be obtained from the LP-relaxation of formulation F3, denoted with LF3. The value of the lower bound can be improved by adding valid inequalities that are satisfied by any feasible integer solution but not necessarily verified by LF3.

$$(F3) \quad z(F3) = \text{Min} \sum_{(i,j) \in E'} c_{ij} \xi_{ij} \quad (3.20)$$

$$\text{s.t. } \sum_{j \in V} x_{ij} - \sum_{j \in V} x_{ji} = -2y_i, \quad i \in V \quad (3.21)$$

$$\sum_{j \in V} (x_{ij} + x_{ji}) = 2y_i(n-1), \quad i \in V \quad (3.22)$$

$$\sum_{j \in C_h} y_j \geq 1, \quad h = 1, \dots, m \quad (3.23)$$

$$x_{ij} + x_{ji} = (n-1)\xi_{ij}, \quad (i, j) \in E' \quad (3.24)$$

$$x_{ij} \geq 0, \quad i, j \in V, i \neq j \quad (3.25)$$

$$y_j = 1, \quad j \in T \quad (3.26)$$

$$y_j \in \{0, 1\}, \quad j \in V \setminus T \quad (3.27)$$

$$\xi_{ij} \in \{0, 1\}, \quad (i, j) \in E' \quad (3.28)$$

The valid inequalities described in Section 2 can be easily adapted for formulation F3. Moreover, other valid inequalities, the so-called *flow inequalities* (see Baldacci 1999) can be considered to improve the lower bound provided by LF3. These inequalities are derived from the observation that in any feasible integer solution, if $\xi_{ij} = 1$, $(i, j) \in E'$, $i \neq s, j \neq s$, then $x_{ij} \geq 1$ and $x_{ji} \geq 1$. Therefore, $x_{ij} \geq \xi_{ij}$ and $x_{ji} \geq \xi_{ij}$, are valid inequalities.

4. Scatter Search

Scatter Search algorithms were first proposed by Glover, (1977), Glover, (1995). The underlying idea is the use of a set R of solutions, or representative points, called *Reference Set*. Recombination operators are applied on the points in R to generate new solutions, each of which is then used as an input to a local search procedure to obtain improved solutions. Linear combinations of solutions are the favored means to recombine solutions. The best solutions among those obtained by this recombination-improvement process will be inserted into the reference set. The whole process is then repeated for a number of iterations. A very general template of Scatter Search methods is the following:

1. Generate an initial set of solutions R (*Reference Set*) by heuristic procedures designed for the problem considered.

2. Create new solutions by applying recombination operators to subsets of R .
3. Modify the solutions obtained in step 2 in order to obtain feasible solutions.
4. Add a subset of *good* solutions among those produced in step 3 to the set R .
5. Repeat steps 2-5 until a termination condition is met.

4.1 Scatter Search for Integer Programming

The general template can be presented considering a generic integer program IP defined on the variables ξ_i , $i = 1, \dots, n$. Let LP be the problem obtained from IP by relaxing the integrality constraints on variables ξ and let $\xi(0)$ be an extreme point of the convex polytope defined by the LP constraints. Each extreme point adjacent to $\xi(0)$ can be represented as $\xi(h) = \xi(0) - \theta_h \Delta_h$, $h \in NB = \{k : \xi(k) \text{ is currently a nonbasic solution}\}$, where Δ_h denotes the direction to follow and θ_h the step necessary to reach the new extreme point $\xi(h)$.

In order to detail the algorithm, it is important to define some fundamental components, which are:

(i) *Diversification procedure*

Generates a set of candidate solutions starting from a seed solution.

(ii) *Local search procedure*

Improves the quality of input solution. In case it is accepted that the input solution can be infeasible, this procedure also acts as a repair operator.

(iii) *Subset generation*

Defines subsets of solutions from the reference set which will be later recombined.

(iv) *Recombination Procedure*

Constructs new solutions from subsets of previously stored ones.

(v) *Reference set update*

Defines how current reference set solutions should be replaced by newly generated ones.

When these elements are defined, the general framework algorithm becomes the following:

Scatter Search Algorithm

Step 1. (Initialization)

Initialize the Reference Set R by means of the diversification procedure.

Step 2. (Local Search)

Use a local search procedure to improve the quality of the solutions generated at Step 1, obtaining feasible enhanced solutions which are added to the reference set.

Step 3. (Subset Generation)

Generate a collection of subsets of the reference set.

Step 4. (Recombination)

For each subset of reference set solutions, generate one or more new solutions.

Step 5. (Local Search)

Create new enhanced trial solutions by means of local search applied to each solution produced at Step 4.

Step 6. (Reference Set Update)

Update the reference set by adding all or the best solutions found in Step 5 and possibly expunging the worst ones present in the set.

Step 7. Repeat Steps 3 to 7 until a termination condition is met.

A possible specification of this framework, leading to an operational algorithm definition, could use the following subroutines.

Diversification procedure: This procedure is used to initialize the reference set. A possible method for this is to generate a sequence of solutions, in such a way that each new solution maximizes its distance from the set of already generated ones, according to some metric.

Glover and Laguna, (1997), propose the following *sequential diversification* procedure for $(0 - 1)$ vectors. It is assumed that a solution \bar{x} complements x over a set J if $\bar{x}_j = 1 - x_j, \forall j \in J$, and $\bar{x}_j = x_j, \forall j \notin J$.

Sequential Diversification Algorithm

Step 1. Arbitrarily define an initial solution x and its complement \bar{x} .*Step 2.* Let n be the number of variables. Partition the set $N = 1, \dots, n$ into P' and P'' , each of cardinality as close as possible to $n/2$. From vector x define two solutions $x^{P'}$ and $x^{P''}$ such that $x^{P'}$ complements x over P' and $x^{P''}$ complements x over P'' .*Step 3.* Define each subset generated at the last partitioning step as *key subset*. If no key subset contains more than one element, the procedure terminates.

Otherwise, partition each key subset $S \subset N$ into S' and S'' , each of cardinality close to $|S|/2$. If there exists a key subset S containing a single elements, then set $S' = S$ and $S'' = \emptyset$.

- Step 4.** Let N' be the union of all subsets S' and N'' the union of all subsets S'' . Generate solutions $x^{N'}$ and $x^{N''}$ as in Step 2 and goto Step 3.

Local search procedure: Local search procedures are greatly problem-dependent. A possible local search algorithm for the CTP is presented in Section 5.

Subset generation: This procedure generates four different types of subsets, differing in the number and quality of the elements they contain.

Subset type 1: subsets of two elements, one of which is the best solution in the reference set.

Subset type 2: subsets of three elements, containing the two best solutions of the reference set.

Subset type 3: subsets of four elements, containing the three best solutions of the reference set.

Subset type 4: subsets consisting of the best i elements of the reference set, $i = 5, \dots, |R|$.

After having constructed the subsets, the new solutions are generated combining only elements of the same subset type.

Recombination procedure: For each subset S obtained by the subset generation procedure, new solutions are generated as recombinations of elements of S . This operator can be problem dependent, however Glover, (1995), suggests, in the case of integer programming, of using a linear combination to generate new offspring, possibly generating points outside of the convex hull of the parent solution set.

Reference set update: This procedure is called after each local search step. The reference set contains the best n_{max} solutions found, ordered by decreasing quality. The new local optimum is inserted if it is of sufficient quality and if it does not duplicate an already stored solution. A hashing function is used to facilitate the ordering/retrieval operations.

The general framework of the resulting algorithm is the following.

SS1 Algorithm

Step 1. (Initialization)

Initialize the Reference Set R by means of the diversification procedure.

Step 2. (Local Search)

Carry each seed solution to the corresponding local optimum, obtaining an enhanced solution. Update R with the update procedure.

Step 3. (Subset Generation)

Generate a collection Ξ_1, \dots, Ξ_k of subsets of the reference set.

Step 4. (Recombination)

Apply to each subset Ξ_i , $i = 1, \dots, k$, the recombination procedure to obtain one or more combined solutions.

Step 5. (Local Search)

The solutions obtained in Step 4 are carried to their local optimum obtaining new enhanced trial solutions.

Step 6. (Reference Set Update)

Update the reference set with the best solutions found in Step 5.

Step 7. Repeat Steps 3 to 7 until a termination criterion is met.

4.2 Scatter Search and Cutting Planes

A second version of Scatter Search makes a more prescriptive use of considerations derived from the analysis of the problem feasibility region Glover, (1995). The general scheme is the same as in SS1, but it suggests to integrate cutting planes in this framework, using extreme points of the induced problem polytope as seed points for a rounding procedure that yields feasible integer solutions. Specifically, using the notation introduced at the beginning of this section, each point $\xi(h)$ obtained from $\xi(0)$ is a new element to use during the search process as seed for obtaining feasible solutions for IP.

Moreover, new points can be also obtained as non-convex combinations of elements of R , thus creating solutions containing information which is not represented in the points of the current reference set.

The general framework of the resulting algorithm is the following.

SS95 Algorithm

Step 1. (Initialization)

Initialize the Reference Set R with feasible solutions of problem IP.

Step 2. (Generation of New Solutions):

Let $\xi(0)$ be the optimal LP solution. Set $H = \{\xi(h) = \xi(0) - \theta_h \Delta_h : h \in NB, \theta_h > 0\}$.

Step 3. (Parent Set Definition)

Define the subsets Ξ_1, \dots, Ξ_k such that Ξ_1, \dots, Ξ_s , $s < k$, contain only elements of H , Ξ_{s+1}, \dots, Ξ_t , $t < k$, contain only elements of R and $\Xi_{s+t+1}, \dots, \Xi_k$ contain both elements of H and of R .

Step 4. (Recombination)

Construct the sets $Q(\Xi_j)$, $1 \leq j \leq k$, with points obtained from linear combination of points in Ξ_j , $1 \leq j \leq k$.

Step 5. (Local Search)

The solutions in $Q(\Xi_j)$, $1 \leq j \leq k$, are carried to their local optimum.

Step 6. (Reference Set Update)

Update R with the best solutions obtained.

Step 7. (Cutting Planes)

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.

The same subroutines introduced above should be specified also in this case.

Initial generation: Can be made in any suitable way, for example by sequential diversification.

New solutions: The most effective way to explore the whole solution space by means of linear combination of seed solutions is to use as seeds the extreme points of the convex hull of the polytope associated with the LP-relaxation of the problem to solve. In particular, an effective strategy is to start with the LP optimal solution and use as seed points extreme points which are adjacent to the LP optimal solution.

Local search: While local search is strictly problem-dependent, a related activity of interest in the case of linear combination of integer solutions is the recover of integer feasibility from fractional solutions. A procedure proposed

in Glover, (1995), to this end is the *directional rounding*, which is as follows. Let $X(0, 1)$ be a unit hypercube in the space \mathbb{R}^n , where $x \in X(0, 1)$ stands for $0 \leq x_j \leq 1$, for all $0 \leq j \leq n$, and let $V(0, 1)$ denote the vertices of the hypercube, thus $x \in V(0, 1)$ indicates $x_j \in \{0, 1\}$, for all $0 \leq j \leq n$. A directional rounding δ is a mapping from the continuous space $X(0, 1) \times X(0, 1)$ onto $V(0, 1)$ defined as follows:

$$\delta(x^*, x') = (\delta(x_1^*, x'_1), \dots, \delta(x_n^*, x'_n))$$

with

$$\delta(x_j^*, x'_j) = \begin{cases} 0 & \text{if } x'_j < x_j^* \\ 1 & \text{if } x'_j > x_j^* \\ 0 & \text{if } x'_j = x_j^* \text{ and } x'_j < 0.5 \\ 1 & \text{if } x'_j = x_j^* \text{ and } x'_j \geq 0.5 \end{cases} \quad 0 \leq x_j^*, x'_j \leq 1, x_j^*, x'_j \in \mathbb{R}^n.$$

Point x^* is called base point and point x' focal point. The directional rounding between a vector x^* and the set $X \subset V(0, 1)$ is then defined to be:

$$\delta(x^*, X) = \{\delta(x^*, x') : x' \in X\}$$

The integer solutions so obtained may be still infeasible because of constraints other than the integrality ones. In this case directional rounding is not enough and other problem-specific repair procedures are in order.

Reference set update: This procedure is the same as in SS1.

Cutting planes: Each iteration of the scatter search main loop terminates with the separation of new valid inequalities. It is thus possible to add new constraints to the current LP formulation and to identify new extreme points of the redefined polytope which act as seed solutions for the subsequent iteration.

5. Scatter Search for the CTP

We designed and implemented three Scatter Search algorithms for the CTP. The first one, SS-CTP1, follows the guidelines of algorithm SS95 and uses the cutting planes obtained by means of the GSEC inequalities to define the points $\xi(0)$, which in turns initialize each iteration of the algorithm main loop. The second algorithm, SS-CTP2, differs from SS-CTP1 in the use of a *star path* solution generation technique, which will be detailed in the presentation of the algorithm. Finally, the third algorithm SS-CTPB is designed according to the template SS1 presented in Glover, (1997), and does not use cutting planes. All algorithms share a number of problem-dependent procedures, which will be introduced in the following.

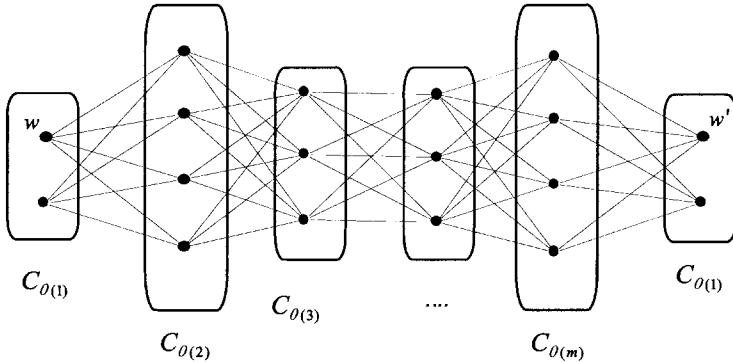


Figure 3.4. Layered Network

Generation of seed points: We used a random seed point generation, combined with a sequential diversification procedure. The working is as follows. We generate $\lceil \frac{|R|}{2} \rceil$ seed solutions, where $|R|$ is the reference set cardinality. Each is generated as a feasible CTP solution, and then modified by means of sequential diversification.

The generation of good initial feasible solution is a nontrivial task in the case of CTP. We propose the following procedure, adapted from Fischetti et al., (1997). Consider the Covering Tour Problem with formulation F2.

Let $(C_{\theta(1)}, \dots, C_{\theta(m)}, C_{\theta(m+1)})$ be a permutation of the clusters, where $C_{\theta(m+1)}$ is the duplicate of $C_{\theta(1)}$. It is possible to construct a *Layered Network* (LN) sequencing all clusters plus one, containing a dummy vertex w duplicating each $w \in C_{\theta(1)}$. The edges of the network are all edges (i, j) , $i \in C_{\theta(k)}$, $j \in C_{\theta(k+1)}$, $1 \leq k \leq m$, see Figure 3.4 for an example. Each path in LN from a vertex $w \in C_{\theta(1)}$ to the associated vertex w' identifies a feasible CTP solution. The best solution derived from the sequence $C_{\theta(1)}, \dots, C_{\theta(m+1)}$ can be obtained computing the shortest path from each vertex $w \in C_{\theta(1)}$ to the associated vertex w' . If the shortest paths are computed by means of a Dijkstra algorithm, the complexity for obtaining the best solution, given a cluster ordering, is $O(|C_{\theta(1)}|n^2)$ and it is obviously minimized by using as first the lowest cardinality cluster.

In the case, quite common in the CTP, where the clusters are not disjoint the paths are not guaranteed to be feasible solutions, as they could contain duplicates of one same node. We patched this situation by modifying the Dijkstra procedure as follows. Each vertex $i \in V$ can have multiple labels: one for each cluster C_k where it appears. Each of these labels specifies which is the predecessor of vertex i when it is considered as a member of cluster C_k , l_{iC_k} ,

and the minimum cost to reach it, f_{iC_k} . The algorithm goes through two phases: in the first one we compute a structure specifying all different labellings, in the second phase the path of interest is made explicit.

The main difference with the standard shortest path algorithm is that, when a node is identified to enter a path, first a test is made to check whether it is not already present in that path. In case it is, it is discarded and the nearest node of its same cluster is considered, then possibly the third best one and so on. More specifically, the algorithm for identifying a path from a vertex $w \in C_{\theta(1)}$ to $w' \in C_{\theta(m+1)}$ is as follows. *Active* is a vector of Booleans, such that $\text{Active}(i) = \text{False}$ if and only if node i is uncovered.

CTP Heuristic Algorithm

Step 1. Let $C_{\theta(1)}, \dots, C_{\theta(m+1)}$ be a sequence of clusters. Initialize $l_{iC_{\theta(k)}}$, the label associated with vertex i in cluster $C_{\theta(k)}$, to 0 and $f_{iC_{\theta(k)}}$, the minimum cost for reaching i in cluster $C_{\theta(k)}$, to ∞ , for each vertex $i \neq w$ and for each cluster $C_{\theta(k)}$ containing vertex i .

Set $k = 1$ and $f_{wC_{\theta(k)}} = \infty$ for each cluster $C_{\theta(k)}$ containing w . Let $M = V \setminus C_{\theta(1)}$ and $P = \{w\}$.

Step 2. Find $i \in C_{\theta(k+1)} \cap M$ and let $M = M \setminus \{i\}$.

Step 3. For each vertex $p \in P$ such that $f_{iC_{\theta(k+1)}} > f_{pC_{\theta(k)}} + c_{pi}$ set $f_{iC_{\theta(k+1)}} = f_{pC_{\theta(k)}} + c_{pi}$ and $l_{iC_{\theta(k+1)}} = p$.

If $p = w'$, i.e. a solution has been found, go to Step 5.

Step 4. If $C_{\theta(k+1)} \cap M = \emptyset$ then set $k = k + 1$ and $P = P \cup C_{\theta(k)}$. Go to Step 2.

Step 5. Set $\text{Active}(i) = \text{False}$ for each vertex $i \in V$. Set $S = \emptyset$, $r = w'$, $k = m + 1$ and $s = l_{rC_{\theta(k)}}$.

Step 6. Set $k = k - 1$, $S = S \cup \{r\}$ and $\text{Active}(r) = \text{True}$.

Step 7. If $\text{Active}(s) = \text{True}$, then $k = k - 1$, otherwise goto Step 9.

Step 8. Find the node $u \in C_{\theta(k)}$ closest to r and set $s = u$. If $k > 1$ set $r = s$, $s = l_{sC_{\theta(k)}}$ and goto Step 7, otherwise goto Step 6.

Step 9. If $k > 1$ then goto Step 6, otherwise Stop.

Local search procedure: The local search procedure is applied to improve the quality of the solutions obtained by the recombination operator. It has the double objective of getting low cost solutions and of ensuring the feasibility

of all solutions in the reference set, feasibility that is not guaranteed by the recombination procedure alone. The local search algorithm we propose is based on the observation that knowing the right sequence of clusters would make it possible to easily find a good solution. Specifically, the local search procedure consists of three phases: the first phase analyzes the solution proposed by the recombination and find its cluster sequence, the second phase finds a good solution given that sequence and the third phase applies a 2-opt or a 3-opt heuristic to the cluster sequence in order to improve the solution quality.

The three phases have been implemented as follows. Let (x^*, y^*) be the current solution, where x^* is the vector of the variables associated to the edges and y^* is the vector of the variables associated to the vertices. Let L , initially empty, be a list containing the cluster sequence and $Active(i)$ be a flag which is *False* if cluster i is uncovered in the solution, *True* otherwise. The algorithm for the identification of the cluster sequence is:

1. Let j be the vertex with highest y_j^* value. Set $Active(i) = False$ for each i .
2. For each cluster C_k containing node j set $L = L \cup C_k$ and $Active(k) = True$. Compute node j' such that the edge (j, j') has the highest $x_{j,j'}^*$ value and that j' is contained in at least one cluster C_h such that $Active(h) = False$.
3. If no j' node exists, then STOP, list L contains the desired sequence; otherwise set $j = j'$ and goto step 2.

The second phase finds a good CTP solution given a cluster sequence. It works by first producing a feasible solution by means of the CTP heuristic algorithm presented in the previous section and then it tries to improve it by eliminating nodes that cover already covered clusters. The third phase finally tries to swap the positions of two or three clusters in the sequence, then phase 1 and 2 are repeated.

Subset generation procedure: This procedure implements a linear combination of elements of the reference set. It is essential to be able to effectively select subsets of solutions to be later combined. The strategy we used in the CTP context is the same as in algorithm SS1, generating four different types of subsets. The subsets are:

Subset type 1: subset of two elements containing the best reference set solution and another randomly chosen one.

Subset type 2: subset of three elements containing the two best reference set solutions and a randomly chosen one.

Subset type 3: subset of four elements containing the three best reference set solutions and a randomly chosen one.

Subset type 4: subset of K elements, where $K = \lceil \frac{|R|}{J} \rceil$ specifies how many subsets of type 4 will be generated and J is a parameter, $5 \leq J \leq |R|$ quantifying the number of elements to be included in each subset.

Recombination procedure: The recombination operator follows the convex combination operator proposed in Glover, (1995). Each point p_i of a subset X is associated with a weight w_i related to its objective function value, $z(p_i)$. The weight w_i is determined as the ratio $\frac{1/z(p_i)}{\sum_{i \in X} 1/z(p_i)}$. After generating all weights, a new point p_n is obtained by the convex combination $p_n = \sum_{i \in X} w_i p_i$.

If generic linear combination are used instead of only convex ones, it is possible to obtain points outside of the convex hull of the points in X . This permits an improved diversification, at the price of possibly extending too much the search space.

Reference set update procedure: This is the same already described for algorithm SS1.

Cutting plane procedure: As mentioned in the previous section, cutting planes have been proposed as a means to generate promising seed points for the reference set. This in fact permits the effective search techniques implemented in the procedure as described so far to take advantage of insights into the search space, specifically as provided by the polytope of the linear relaxation of the CTP.

The cutting procedure is much similar to that of standard branch and cut methods. Given the linear relaxation of the problem, defined over a constraint set Ω , the process starts by solving a relaxation using only a subset $\Upsilon \subset \Omega$ of constraints. If the solution (x^*, y^*) satisfies all constraints in Ω then stop, otherwise identify (separate) the most violated constraints, add them to Υ and repeat the procedure.

A critical element is the separation procedure. In our application, starting from formulation F2, we relaxed constraints (16) and used generalized subtour elimination (GSEC) constraints (8), (9) and (10) in the cutting phase. The separation procedure were borrowed from Fischetti et al., (1997), where they were proposed for the GTSP.

Two procedures were used. The first, called GSEC_SEP, is an exact procedure which can identify the most violated constraints, while the second one, GSEC_H2, is heuristic and can quickly identify some violated constraints, with no guarantee on the degree of violation.

The separation algorithm we used starts from a solution (x^*, y^*) and applies GSEC_H2. If violated constraints are found, than they are added to the constraint set and scatter search continues, otherwise GSEC_SEP is applied.

5.1 Algorithm SS-CTP1

Given the elements described in the previous section, it is now possible to provide the step-by-step description of the first scatter search algorithm applied to CTP, i.e., SS-CTP1.

SS-CTP1 Algorithm

Step 1. Initialize the Reference Set R with the sequential diversification procedure Glover, (1997), and build an initial feasible solution (x', y') with the local search procedure described above.

Let $J_y = \{i : y'_i = 1\}$, X_i be the set of the k least-cost edges incident in i and having the other vertex in J_y and let $U = \bigcup_{i \in J_y} X_i$. Let $J_x = \{(i, j) : x'_{ij} = 1\} \cup U$. Finally, let $\Upsilon = \emptyset$ be the set of GSEC inequalities currently in use.

Step 2. Compute the optimal solution (x^*, y^*) of the LF2 problem defined by the constraints in Υ but using only variables $y_i, i \in J_y$, and $x_{ij}, (i, j) \in J_x$. By means of a sequential diversification procedure construct p feasible solutions and add them to R .

Step 3. Identify the r most violated GSEC constraints, to be introduced in LF2, and the negative reduced cost variables whose indices are to be added to J_x and J_y .

Step 4. Apply the subset generation and the recombination operators on the points in R .

Step 5. If (end condition) then STOP; otherwise go to Step 2.

5.2 Algorithm SS-CTP2

The second algorithm we tested is based on the idea of *Star Paths*, also proposed in Glover, (1995). It is a technique based on the directional rounding operator applied to the extreme points adjacent to the optimal LP solution identified after the cutting phase.

Let x^* be the current optimal LP solution of a generic integer problem. Let \widehat{X} be a set of extreme points adjacent to x^* . The star path procedure starts with the definition of a continuous path $P(x', x'') = \{x : x = \lambda x' + (1 - \lambda)x''\}$ connecting two points $x', x'' \in \widehat{X}$. Successively, this path is turned into a Star

Path by means of the directional rounding operator $\delta(x^*, P)$. The trajectory obtained after rounding is not continuous, and represents in fact the projection of path P on the unit hypercube $V(0, 1)$.

The procedure we used to produce the Star Path points follows the directives given in Glover, (1995). Let $\delta(\lambda)$ be a point of $\delta(x^*, P)$ identified as a function of the parameter λ . For each λ a point $x = \lambda x^* + (1 - \lambda)x'$, with $x' \in \hat{X}$ is identified. Notationally, $\delta(\lambda) = \delta(x^*, x)$ and $\delta_j(\lambda) = \delta(x_j^*, x_j)$ is the j -th component of $\delta(\lambda)$. Vector x can be alternatively identified as $x = x^* + \lambda\Delta$, where $\Delta = x' - x^*$. It is possible to define the sets $I(0) = \{j : \Delta_j = 0\}$, $I(+) = \{j : \Delta_j > 0\}$ and $I(-) = \{j : \Delta_j < 0\}$. Moreover, for each $j \in I(+)$ and for each $j \in I(-)$ let $\lambda(j) = (x_j^* - x_j')/\Delta_j$. The star path generation procedure is based on Lemma 4 in Glover, (1995), which states that the elements of $\delta(\lambda)$, given λ , are:

$$\begin{aligned}\delta_j(\lambda) &= \delta(x_j^*, x'), j \in I(0) \\ \delta_j(\lambda) &= \begin{cases} 0 & \text{if } \lambda < \lambda(j) \text{ and } j \in I(+) \\ 1 & \text{if } \lambda \geq \lambda(j) \text{ and } j \in I(+) \end{cases} \\ \delta_j(\lambda) &= \begin{cases} 1 & \text{if } \lambda < \lambda(j) \text{ and } j \in I(-) \\ 0 & \text{if } \lambda \geq \lambda(j) \text{ and } j \in I(-) \end{cases}\end{aligned}$$

Given this, the star path generation procedure is the following.

Star Path Generation Procedure

- Step 1.* Let x^* and x' be the two extreme points of the star path. Let $\theta(1), \dots, \theta(u)$ be a permutation of the indices in $I(+) \cup I(-)$ such that $\lambda(\theta(1)) \leq \lambda(\theta(2)) \leq \dots \leq \lambda(\theta(u))$, where $u = |I(+) \cup I(-)|$.
- Step 2.* Set $\lambda = \lambda_{start} < \lambda(\theta(1))$ and generate a solution $x^o = \delta(\lambda)$ as dictated by Lemma 4.1
- Step 3.* Generate point \hat{x}^o by setting $\hat{x}_p^o = 1 - x_p^o$ and leaving $\hat{x}_j^o = x_j^o \forall j \neq p$.
- Step 4.* Set $h = h + 1$. If $h > u$ or $\lambda(\theta(h)) > \lambda_{end}$, with $\lambda_{start} < \lambda_{end} \leq \lambda(\theta(u))$, then STOP, else set $p = \theta(h)$ and go to Step 1.

This algorithm is a subroutine of the second scatter search algorithm we tested on the CTP, which is as follow.

SS-CTP2 Algorithm

Step 1. Initialize the Reference Set R with the sequential diversification procedure Glover, (1997), and build an initial feasible solution (\bar{x}, \bar{y}') with the local search procedure described above.

Let $J_y = \{i : y'_i = 1\}$, X_i be the set of the k least-cost edges incident in i and having the other vertex in J_y and let $U = \bigcup_{i \in J_y} X_i$. Let $J_x = \{(i, j) : x'_{ij} = 1\} \cup U$. Finally, let $\Upsilon = \emptyset$ be the set of GSEC inequalities currently in use.

Step 2. Compute the optimal solution (x^*, y^*) of the LF2 problem defined by the constraints in Υ but using only the variables in J_x and J_y . Let \widehat{X} be the set containing extreme points adjacent to (x^*, y^*) . Construct one or more Star Paths using the points in \widehat{X} . Use the points in the Star Paths to generate seed solutions, to be made feasible and improved by means of local search, and add them to the reference set.

Step 3. Identify the r most violated GSEC constraints, to be introduced in LF2, and the negative reduced cost variables whose indices are to be added to J_x and J_y .

Step 4. Apply the subset generation and the recombination operators on the points in R .

Step 5. If (end condition) then STOP; otherwise go to Step 2.

6. Computational Results

In this section we present the computational results of the algorithms SS-CTPB, the basic Scatter Search algorithm presented in Glover, (1997), SS-CTP1 and SS-CTP2 presented in the previous section.

The algorithms were coded in Fortran 77 and run on a Pentium III 750 Mhz machine. We used CPLEX 6.5, as the LP-solver.

The CTP test instances were generated by means of the procedure proposed in Gendreau et al., (1995), as follows. The vertex set was obtained by generating $|V| + |W|$ points in the $[0, 100] \times [0, 100]$ square, according to a uniform distribution. The sets T and V were defined by taking the first $|T|$ and $|V| - |T|$ points, respectively, and W was defined as the set of the remaining points. Tests were generated for $n = 50, 75, 100$, $|T| = 1, \lceil .25n \rceil, \lceil .50n \rceil, \lceil .75n \rceil$ and $|W| = n, 2n, 4n, 5n$. For each combination of the parameters, five instances were generated.

The costs $\{c_{ij}\}$ were computed as integer values equal to $\lfloor e_{ij} + .5 \rfloor$, where e_{ij} is the Euclidean distance between points i and j (see Reinelt 1991). The threshold values D_j , $j = 1, \dots, n$, were all set equal to d , where d was determined

as $d = \max(\max_{k \in V \setminus T} \min_{l \in W} \{d_{lk}\}, \max_{l \in W} \{d_{l\pi(l)}\})$, where $\pi(l)$ is the index of the vertex of $V \setminus T$ that is the second closest to l . This ensures that each vertex of $V \setminus T$ covers at least one vertex of W and each vertex of W is covered by at least two vertices of $V \setminus T$.

The computational results we present refer to the three versions of scatter search applied to CTP introduced in the previous section, which are SS-CTPB (scatter search with no cutting planes), SS-CTP1 (scatter search with cutting planes) and SS-CTP2 (scatter search with cutting planes and star paths). Each algorithm has a number of parameters, their union gives the following set:

k	: number of arcs to introduce in set J_x ;
p	: number of heuristic solutions to be constructed by means of the cutting-based approach;
r	: number of violated constraints to be introduced at each iteration;
q	: number of times the recombination operator is applied on the points in the reference set R ;
n_{max}	: maximum number of solutions in the reference set;
t_{max}	: maximum CPU time.

We carried out a number of runs to define an efficient parameter settings for each algorithm. We used a *coeteris paribus* approach, based on sets of three or four values for each parameter. The resulting setting is the following.

The number of solutions n_{max} was set to $\frac{2}{3}|V|$ for the problems with $|V| = 100$, and to 40 for the other ones.

The number of subsets generated included all subsets of types 1, 2 and 3, while subsets of type 4 included only 5 elements.

The number of heuristic solutions p was set to $\lceil \frac{n_{max}}{5} \rceil$. The same value was also used as the number of extreme points utilized to generate the star paths.

At each iteration of the algorithm, we inserted in the current linear relaxation 40% of the constraints identified by the separation algorithm.

Finally, t_{max} was set to 1500 CPU seconds for all problems.

Table 3.1 shows the details of the problem reductions. In Table 3.2 we compare the results obtained by algorithm SS-CTP1 and SS-CTPB. The comparison was made on a subset of 80 difficult CTP instances selected among the 240 instances generated. Table 3.3 reports the results obtained by algorithm SS-CTP1 and SS-CTP2 compared against the results published in Gendreau et al., (1995). Finally, Table 3.4, shows the results obtained by algorithm SS-CTP1 using the two mathematical formulations of the CTP presented in Section 3. For this comparison we selected 18 difficult CTP instances among the instances generated.

Under the heading *Dimension*, Tables 3.1 to 3.4 show the following columns:

$ V $: cardinality of the set V ;
$ T $: cardinality of the set T ;
$ W $: cardinality of the set W .

Table 3.1 shows the following columns:

- $|V'|$: average number, over the five instances, of the cardinality of the set V after the problem reductions;
 I : average number, over the five instances, of the vertices belonging to more than one cluster;
 C : average number, over the five instances, of clusters generated.

Tables 3.2 and 3.3 show, under the SS-CTP1 and SS-CTPB headings, the following columns:

- RIS : number of problems solved to optimality among the five instances;
GAP : average gap computed over the five instances between the final upper and lower bounds (0 if all the five instances were solved to optimality);
CPU : average cpu time in seconds needed to obtain the best solution.

Tables 3.3 shows, under the Gendreau et al. heading, the following columns:

- RIS : number of problems solved to optimality among the five instances;
GAP : average gap computed over the five instances between the upper bound provided by the heuristic and the optimal solution value;

Table 3.4 shows the following columns:

- LBOPT : best lower bound obtained either by formulation LF2 or LF3 after the addition of all valid inequalities;
LBO : LP-relaxation value of formulation LF3 and LF2;
GAP : gap between the upper bound and the lower bound LBOPT;
CPU : cpu time in seconds needed to obtain the best solution.

It is evident from Table 3.1 that the number of mandatory nodes in T has a big impact on the effectiveness of the reduction procedures. In fact, in all three cases of $|V| = 50$, $|V| = 75$ and $|V| = 100$, very few problem groups with $|T| \neq 1$ have a significant number of vertices belonging to more than one cluster, thus increasing the number of decision alternatives to consider. Moreover, the increase of the cardinality of T entails a reduction of V that makes the two sets usually coincident in almost all problem instances for which $|T| = 0.75 * |V|$. The problems with high T cardinality are therefore actually deprived of their covering component, leaving just the tour definition problem. This is not true for small T cardinalities, for which the value of I is consistently different from 0.

Table 3.2. Comparison of the algorithms SS-CTP1 and SS-CTPB

Dimension			SS-CTP1		SS-CTPB	
$ V $	$ T $	$ W $	GAP	CPU	GAP	CPU
75	1	75	0.3	348.4	3.9	7.5
75	1	150	0.0	634.1	3.7	14.3
75	1	300	3.1	826.6	3.4	11.9
75	1	375	0.2	774.5	2.9	6.3
100	1	100	9.2	909.1	10.9	27.6
100	1	200	10.5	894.4	12.7	41.1
100	1	300	16.8	876.2	36.6	34.1
100	1	500	16.5	875.2	25.5	43.2
Avg			7.1	767.3	12.45	23.2

Table 3.1. Problem reduction

$ V $	$ T $	$ W $	$ V' $	I	C
50	1	50	46.0	34.4	19.4
50	1	100	46.4	38.0	24.4
50	1	150	47.6	41.4	28.6
50	1	200	38.0	24.6	23.6
50	13	50	28.6	0.8	16.0
50	13	100	27.4	4.6	17.6
50	13	150	23.0	3.2	16.4
50	13	200	38.0	20.4	25.4
50	25	50	25.0	0.0	25.0
50	25	100	26.2	0.0	25.2
50	25	150	26.2	0.0	25.2
50	25	200	26.2	0.0	25.2
50	38	50	38.0	0.0	38.0
50	38	100	38.0	0.0	38.0
50	38	150	42.6	5.4	40.4
50	38	200	38.0	0.0	38.0
75	1	75	69.8	55.8	36.2
75	1	150	72.8	67.4	43.2
75	1	300	74.2	63.8	56.0
75	1	375	73.6	69.2	59.0
75	19	75	39.8	5.0	24.0
75	19	150	32.4	3.4	21.8
75	19	300	37.0	3.8	23.6
75	19	375	38.6	3.6	23.0
75	38	75	40.6	0.8	35.4
75	38	150	40.4	0.0	38.4
75	38	300	40.4	0.0	38.4
75	38	375	40.4	0.0	38.4
75	56	75	56.0	0.0	56.0
75	56	150	56.0	0.0	56.0
75	56	300	56.0	0.0	56.0
75	56	375	56.0	0.0	56.0
100	1	100	97.8	82.0	48.8
100	1	200	99.8	95.8	79.4
100	1	300	99.8	95.8	93.6
100	1	500	100.0	95.6	106.8
100	25	100	42.6	5.8	29.0
100	25	200	47.4	8.6	30.2
100	25	300	53.0	10.6	35.8
100	25	500	54.0	13.0	37.2
100	50	100	51.0	0.0	50.2
100	50	200	50.0	0.0	50.0
100	50	300	50.0	0.0	50.0
100	50	500	51.0	0.0	50.2
100	75	100	75.0	0.0	75.0
100	75	200	75.0	0.0	75.0
100	75	300	75.0	0.0	75.0
100	75	500	75.0	0.0	75.0
Avg			51.6	17.8	42.7

Table 3.2 shows a comparison of the basic scatter search approach, as proposed in Glover, (1997), and implemented in SS-CTPB, with the cutting planes based one of Glover, (1995), implemented in S-CTP1. The comparison was made only on a subset of hard instances, in order not to dilute the differences with a number of results on easy problems, for which both approaches are effective. On all tested problems the two approaches follow a consistent pattern: SS-CTP1 is slower but more effective. The high CPU time is mainly due to the LP solver, which is given to solve a number of progressively more complex problems, but the indications which can be derived from the redefinition of the problem polytope prove to be an effective search guidance.

Table 3.3. Comparison of SS-CTP against Gendreau et al.

Dimension			SS-CTP1			SS-CTP2			Gendreau et al.	
V	T	W	RIS	GAP	CPU	RIS	GAP	CPU	RIS	GAP
50	1	50	5	0.0	22.9	5	0.0	28.3	5	1.3
50	1	100	4	0.0	85.5	4	0.0	93.5	4	0.7
50	1	150	5	0.0	48.1	5	0.0	51.2	2	0.8
50	1	200	5	0.0	24.1	5	0.0	32.7	4	0.3
50	13	50	5	0.0	1.6	5	0.0	20.6	3	0.3
50	13	100	4	0.0	1.9	4	0.0	15.4	4	0.6
50	13	150	5	0.0	1.2	5	0.0	3.0	3	0.1
50	13	200	5	0.0	11.7	5	0.0	11.9	4	0.5
50	25	50	5	0.5	1.6	5	0.5	1.6	2	0.3
50	25	100	5	0.5	1.9	5	0.5	4.2	2	0.6
50	25	150	5	0.2	1.6	5	0.2	3.0	2	0.4
50	25	200	5	0.2	1.6	5	0.2	4.9	3	0.4
50	38	50	5	0.3	4.7	5	0.3	7.9	2	1.3
50	38	100	5	0.5	4.7	5	0.5	8.2	2	1.6
50	38	150	5	0.2	8.6	5	0.2	9.3	2	1.4
50	38	200	5	0.2	4.9	5	0.2	6.5	2	1.5
75	1	75	4	0.3	348.4	4	0.3	427.8	3	0.8
75	1	150	5	0.0	634.1	5	0.0	674.5	4	0.3
75	1	300	1	3.1	826.6	1	3.1	829.9	5	1.5
75	1	375	3	0.2	774.5	3	0.2	799.1	1	2.0
75	19	75	2	0.5	6.3	2	0.5	20.6	1	0.3
75	19	150	3	0.3	2.3	3	0.3	3.7	1	0.1
75	19	300	3	0.6	4.7	3	0.6	5.6	1	0.1
75	19	375	4	0.3	5.1	4	0.3	8.9	2	0.5
75	38	75	1	0.5	5.8	1	0.5	6.5	2	2.1
75	38	150	2	0.3	4.9	2	0.3	6.1	2	1.0
75	38	300	2	0.3	5.4	2	0.3	7.7	3	2.1
75	38	375	1	0.3	5.1	1	0.3	7.0	3	1.6
75	56	75	1	0.9	21.7	1	0.9	23.8	0	2.5
75	56	150	1	0.9	22.0	1	0.9	28.3	0	3.3
75	56	300	1	0.9	22.0	1	0.9	28.5	0	3.5
75	56	375	1	0.9	22.2	1	0.9	27.1	1	3.1
100	1	100	0	9.2	909.1	0	9.2	934.8	2	1.3
100	1	200	0	10.5	894.4	0	10.5	932.7	1	2.0
100	1	300	0	16.8	876.2	0	16.8	884.3	0	2.3
100	1	500	0	16.5	875.2	0	16.5	882.9	1	3.3
100	25	100	3	0.6	11.9	3	0.6	12.4	1	0.6
100	25	200	2	0.7	25.9	2	0.7	31.5	1	0.0
100	25	300	1	0.8	13.8	1	0.8	15.2	0	0.5
100	25	500	1	1.3	28.0	1	1.3	30.1	0	0.3
100	50	100	0	0.6	75.0	0	0.6	93.7	1	2.5
100	50	200	0	0.6	75.0	0	0.6	82.9	0	2.9
100	50	300	0	0.6	48.6	0	0.6	56.3	0	2.7
100	50	500	0	0.6	73.8	0	0.6	91.6	0	2.7
100	75	100	0	1.2	353.7	0	1.2	375.5	0	2.5
100	75	200	0	1.2	358.4	0	1.2	376.4	0	2.6
100	75	300	0	1.2	356.1	0	1.2	397.7	0	3.1
100	75	500	0	1.2	360.3	0	1.2	371.5	0	3.1
Sum			120		120			82		
Avg			1.6	172.4		1.6	182.9		1.4	

Table 3.3 permits us to compare the effectiveness of the scatter search approach with the tailored approach presented in Gendreau et al., (1995). We report only the results of SS-CTP1 and SS-CTP2 because, as it appears from Table 3.2, SS-CTPB is dominated by these two methods. Two different trends can be identified among the results presented in the table. The first, in accordance with the insight obtained from Table 3.1, discloses that larger $|T|$ values produce easier problems. This can be verified from the results obtained for problems with the same $|V|$. The second observation is that problems with higher $|V|$ values are harder, in fact almost all problems with $|V| = 50$ could be solved to optimality (i.e., at the end of the run we had an identical value for upper and lower bound), while this is true for only a few problems with $|V| = 100$. A further observation is that SS-CTP2 could never improve the values of SS-CTP1, despite the increased computational effort due to the star path generation. It appears therefore that in this case, the generation of star paths is not worth the resources allocated to it. Finally, we want to point out that the comparison with Gendreau et al.'s heuristic is incomplete, as these authors do not report CPU times for their heuristic: we must however emphasize that their heuristic is much faster than scatter search, since in several cases they have been able to solve to optimality a problem in less time than we used in our scatter search.

Table 3.4 compares the results obtained by SS-CTP1 making use of the lower bound derived from F2 and of the two-commodity based one of F3. The two columns labelled LB0, reporting the bound value with no cuts added, show that the two commodities bound is better than Gendreau et al.'s, as it consistently produced higher values. We must however notice that the bound quality is in general very poor, and that SS-CTP1 obtained better results when using the worse LF2 bound. This is because cutting planes and separation procedures have been studied specifically for F2, and little effort has been devoted for doing the same for F3. Better valid inequalities will surely improve the results obtainable by SS-CTP1 using LF3.

Table 3.4. Comparison of algorithm SS-CTPB and SS-CTP1 using formulations (F^3) and (F^2)

Dimension	V	T	W	Formulation F_3			Formulation F_2				
				LB OPT	LB0	SS-CTP1	GAP	CPU	LB0		
75	1	75	596.4	158.4	598	0.3	1082.7	141.5	606	1.6	934.1
75	1	75	664.0	166.2	677	2.0	907.5	148.3	664	0.0	358.9
75	1	75	573.0	139.9	582	1.6	957.0	129.5	573	0.0	234.1
75	1	150	522.0	115.1	522	0.0	995.3	98.6	522	0.0	111.9
75	1	150	715.0	220.0	729	2.0	940.2	200.8	715	0.0	589.7
75	1	150	549.0	137.4	567	3.3	955.6	130.8	549	0.0	576.9
75	1	300	540.0	125.7	546	1.1	1015.9	122.5	540	0.0	512.4
75	1	300	719.0	227.6	730	1.4	890.4	203.5	740	2.8	856.1
75	1	300	553.2	154.5	578	4.5	841.4	146.0	578	4.5	887.9
100	1	100	320.3	82.5	345	7.7	882.9	79.4	345	7.7	847.7
100	1	100	333.8	93.0	353	5.8	1236.2	89.0	353	5.8	987.6
100	1	100	305.6	82.7	346	13.2	843.0	70.5	340	11.2	887.9
100	1	200	298.8	73.9	337	12.8	1294.6	69.0	337	12.8	965.2
100	1	200	311.0	90.8	352	13.2	879.9	82.6	343	10.3	851.9
100	1	200	319.0	109.3	342	7.2	1168.9	97.5	345	8.1	899.1
100	1	300	277.9	86.1	342	23.2	936.7	81.6	351	26.5	851.9
100	1	300	284.9	87.3	329	15.4	851.9	85.0	332	16.5	899.1
100	1	300	319.0	113.3	352	10.1	855.1	106.5	362	13.3	890.7
Avg					6.9	974.2			6.7	730.2	

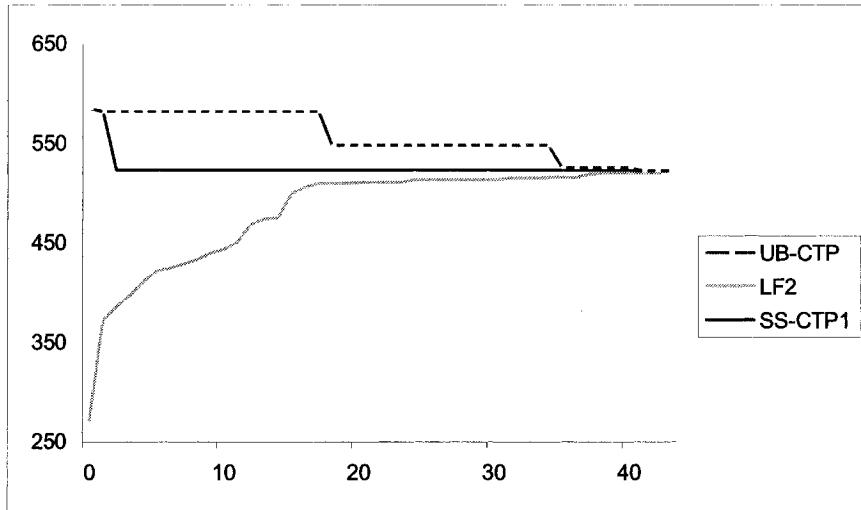


Figure 3.5. Execution Trace

Finally, Figure 3.5 presents the trace of a run on a sample instance (75-1-200). We report for each iteration the LF2 solution value, the SS-CTP1 solution and the best solution obtained by rounding the optimal LF2 solution (UB-CTP). Two observations are in order. The first testifies the effectiveness of generating reference sets and recombining solutions (in fact SS-CTP1 is always below UB-CTP). The second observation is the possibility inherent to SS-CTP1 to detect the optimality of a solution: when LF2 becomes equal to SS-CTP1 (as in Figure 3.5), then the current best solution can be defined to be optimal. This is a possibility still uncommon in current metaheuristic approaches.

7. Conclusions

This paper presents a study on the possibility of applying the scatter search approach to the covering tour problem. Despite its reported practical relevance, the CTP has received little attention from the metaheuristic community, while significant results have been obtained by more mathematically oriented methods. We identified scatter search, with special reference to the method proposed in Glover, (1995), as a highly promising means to encompass polyhedral results in a metaheuristic framework.

Following Glover's two more normative publications on scatter search, Glover, (1995), and Glover, (1997), we designed three scatter search algorithms for the

CTP: one which makes no use of cutting planes, one which uses them and the third one which uses both cutting planes and star paths.

Further contributions of this work include a new mathematical formulation for the CTP, based on a 2-commodity approach as proposed in Baldacci, (1999), and Baldacci et al., (1999), which is the basis for obtaining a new lower bound for the problem, and new simple but original problem reduction rules.

We tested all contributions on a problem testset previously proposed in Gen-dreau et al., (1995). The problem reductions proved to be more effective for problems with larger numbers of mandatory nodes in the solutions, the 2-commodity bound proved to be tighter than the LP relaxation used in previous studies. More work needs to be done on this new model, however, in order to improve the effectiveness of the valid inequalities which can be added to the basic LP-relaxation.

As for the scatter search approaches, we find that both cutting plane-based algorithms outperform the primal-only approach, and that star paths add no contribution to the generate-and-recombine SS-CTP1 algorithm. Again however, we must point out that to the best of our knowledge this is the first time the two approaches have been combined, thus more work needs to be done before claiming definitive results about the superiority of one technique over another.

References

- Arkin, E.M., M.M. Halldorsson and R. Hassin (1993) "Approximating the Tree and Tour Covers of a Graph," *Information Processing Letters*, 47:275–282.
- Arkin, E.M. and R. Hassin (1994) "Approximation Algorithms for the Geometric Covering Salesman Problem," *Discrete Appl. Math.*, 55:197–218.
- Balas, E. (1989) "The Prize Collecting Traveling Salesman Problem," *Networks*, 19:621–636.
- Balas, E. and A. Ho (1980) "Set Covering Algorithms Using Cutting Planes, Heuristic, and Subgradient Optimisation: A Computational Study," *Mathematical Programming Study*, 12:37–60.
- Baldacci, R. (1999) "Algorithms for Location and Routing Problems in Distribution Systems," Ph.D. Thesis, Management Science Dept., Imperial College, London.
- Baldacci, R., A. Mingozzi and E. Hadjiconstantinou (1999) "An Exact Algorithm for the Capacitated Vehicle Routing Problem based on a Two-Commodity Network Flow Formulation," Working Paper, Department of Mathematics, University of Bologna.
- CPLEX Optimization Inc. (1996) Using the Cplex Callable Library and Cplex Mixed Integer Library. 930 Tahoe Blvd 802-297, Incline Viallge, NV89451, U.S.A.

- Current, J.R. and D.A. Schilling (1989) "The Covering Salesman Problem," *Transp. Sci.*, 23:208–213.
- Current, J.R. and D.A. Schilling (1994) "The Median Tour and Maximal Covering Problems," *European Journal of Operational Research*, 73:114–126.
- Finke, G., A. Claus and E. Gunn (1984) "A Two-Commodity Network Flow Approach to the Traveling Salesman Problem," *Congress. Numerantium*, 41:167–178.
- Fischetti, M., J.J. Salazar and P. Toth (1997) "A Branch-and-Cut Algorithm for the Symmetric Generalized Traveling Salesman Problem," *Operations Research*, 45:378–394.
- Fischetti, M. and P. Toth (1988) An Additive Approach for the Optimal Solution of the Prize Collecting Traveling Salesman Problem. In *Vehicle Routing: Methods and Studies*, B.L. Golden and A.A. Assad (eds.), North-Holland, Amsterdam, 319–343.
- Gendreau, M., A. Hertz and G. Laporte (1992) "New Intersection and Postoptimization Procedures for the Traveling Salesman Problem," *Operations Research*, 40:1086–1094.
- Gendreau, M., G. Laporte and F. Semet (1995) "The Covering Tour Problem," *Operations Research*, 45:568–576.
- Glover, F. (1977) "Heuristics for Integer Programming using Surrogate Constraints," *Decision Sciences*, 8:156–166.
- Glover, F. (1995) "Scatter Search and Star Paths: Beyond the Genetic Metaphor," *OR Spektrum*, 17:125–137.
- Glover, F. (1997) A Template for Scattersearch and Path Relinking. Lecture Notes in Computer Science, J.K. Hao, E. Lutton, E. Ronald, M. Schoenauer, D. Snyers (Eds.).
- Glover, F. and M. Laguna (1997) *Tabu Search*, Kluwer Academic Publishers.
- Kourtz, P. (1987) The Need for Improved Forest Fire Detection, *The Forestry Chronicle*, 272–277.
- Labbe, M. and G. Laporte (1986) "Maximizing User Convenience and Postal Service Efficiency in Post Box Location," *Belgian J. Opns. res. Statist. and Computer Sci.*, 26:21–35.
- Langevin, A., M. Desrochers, J. Desrosiers, S. Gelinas and F. Soumis (1993) "A Two-Commodity Flow Formulation for the Traveling Salesman and the Makespan Problems with Time Windows," *Networks*, 23:631–640.
- Laporte, G. and S. Martello (1990) "The Selective Traveling Salesman Problem," *Discrete Appl. Math.*, 26:193–207.
- Lucena, A. (1986) "Exact Solution Approaches for the Vehicle Routing Problem," Ph.D. Thesis, Management Science Dept., Imperial College, London.
- Ntafos, S. (1992) "Watchman Routes under Limited Visibility," *Computational Geometry*, 1:149–209.

- Oppong, J.R. and M.J. Hodgson (1994) "Spatial Accessibility to Health Care Facilities in Suhum District, Ghana," *Professional Geographer*, 46:199–209.
- Reinelt, G. (1991) "Tsplib - a Traveling Salesman Problem Library," *ORSA Journal on Computing*, 3:376–384.
- ReVelle, C.S. and G. Laporte (1993) "New Directions in Plant Location," *Studies in Locational Analysis*, 5:31–58.

Chapter 4

SOLUTION OF THE SONET RING ASSIGNMENT PROBLEM WITH CAPACITY CONSTRAINTS

Roberto Aringhieri and Mauro Dell'Amico

*DISMI - University of Modena and Reggio Emilia, Viale Allegri 13, 42100 Reggio Emilia, Italy,
{aringhieri.roberto;dellamico}@unimore.it*

Abstract Synchronous Optical Network (SONET) in North America and Synchronous Digital Hierarchy (SDH) in Europe and Japan are the current transmission and multiplexing standards for high speed signals within the carrier infrastructure. The typical topology of a SONET network is a collection of rings connecting all the customer sites. We deal with a design problem in which each customer has to be assigned to exactly one ring and these rings have to be connected through a single federal ring. A capacity constraint on each ring is also imposed. The problem is to find a feasible assignment of the customers minimizing the total number of rings used. A Tabu Search method is proposed to solve the problem. The key elements are the use of a variable objective function and the strategic use of two neighborhoods. We have also implemented other techniques such as Path Relinking, exploring Tabu Search and a Scatter Search. Extensive computational experiments have been done using two sets of benchmark instances. The performances of the proposed algorithms have also been compared with those of three multistart algorithms involving greedy methods previously proposed for the problem, and of the CPLEX solver. The computational experiments show the effectiveness of the proposed Tabu Search.

Keywords: Metaheuristics, SONET, Graph Partitioning

1. Introduction

Synchronous Optical Network (SONET) in North America and Synchronous Digital Hierarchy (SDH) in Europe and Japan are the current transmission and multiplexing standards for high speed signals within the carrier infrastructure.

A typical topology of a SONET network is a collection of *local rings* or simply *rings* directly connecting a subset of customer sites or nodes. Each node sends, receives and relays messages through a device called add-drop-

multiplexer (ADM). In bidirectional rings the traffic between two nodes can be sent clockwise or counterclockwise, and the volume of traffic is limited by the ring capacity, which is equal in both directions. The capacity of the bidirectional ring has to accommodate the sum of bandwidth requests between all pairs of nodes connected by the ring. Finally, all rings are connected together by a special ring called *federal ring* through an expensive device, the digital cross connect (DXC), which joins each ring to the federal ring.

With this topology, the traffic between two customers may use up to three rings. A different design choice imposes that the traffic between two nodes is always transmitted using only one ring. In this case, a customer may be assigned to several rings. A design problem requiring this topology has been considered in Laguna, 1994, where a mathematical model and Tabu Search algorithm are presented.

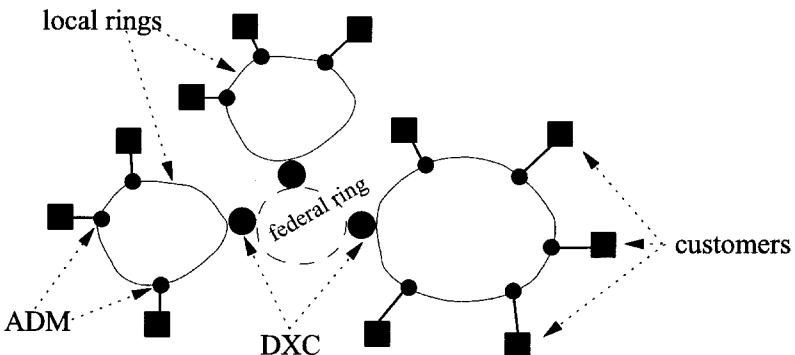


Figure 4.1. A SONET network

In this paper we deal with a basic design problem using the first topology and requiring that: i) each customer has to be assigned to exactly one local ring; and ii) the maximum capacity of each ring is bound by a common value. We want to minimize the number of local rings, that is the cost of DXCs installed. This problem is called SONET Ring Assignment Problem (SRAP) with capacity constraints. An example of SONET network is given in Figure 4.1.

Fixing the number k of rings required, we derive a parametric version of the problem called k -SRAP. Goldschmidt, Laugier and Olinick (Goldschmidt et al. 2003) have shown that SRAP is \mathcal{NP} -complete by reducing the 3-Regular Graph Bisection Problem to a special case of SRAP and proposed three randomized greedy algorithms to solve it.

Mathematical formulations for SRAP are presented in Section 2. In Section 3 we briefly describe the greedy algorithms proposed in Goldschmidt et al. 2003. In Section 4 we introduce a basic Tabu Search (BTS) for solving the problem.

BTS is then improved implementing several intensification and diversification strategies. The strategic use of two neighborhoods leads to a Tabu Search with Strategic Oscillation (TSSO) which is reported in Section 5. Path Relinking, eXploring Tabu Search and Scatter Search approaches are reported in Section 6. Extensive computational results are reported in Section 7, and some conclusions are presented in Section 8.

2. Mathematical Formulations

We introduce the following notation: let n be the number of nodes, B the common capacity bound, d_{uv} the symmetric traffic demand between the pair of nodes u and v ($u, v = 1, \dots, n, u \neq v$) and $W_u = \sum_{v \neq u} d_{uv}$ the total traffic on node u ($u = 1, \dots, n$). Let us define the binary variables: $p_{uvr} = 1$ ($u, v = 1, \dots, n, u \neq v, r = 1, \dots, n$) if nodes u and v belong both to ring r , $p_{uvr} = 0$ otherwise; $x_{ur} = 1$ ($u = 1, \dots, n, r = 1, \dots, n$) if node i is assigned to ring r , $x_{ur} = 0$ otherwise; $y_r = 1$ ($r = 1, \dots, n$) if ring r is used, $y_r = 0$ otherwise. A model for SRAP derived from that presented in Goldschmidt et al. 2003, is the following:

$$\text{SRAP : min } \sum_{r=1}^n y_r$$

s.t. $\sum_{u=1}^n x_{ur} W_u - \sum_{u=1}^{n-1} \sum_{v=u+1}^n p_{uvr} d_{uv} \leq B, r = 1, \dots, n \quad (4.1)$

$$\frac{1}{2} \sum_{u=1}^n W_u - \sum_{r=1}^n \sum_{u=1}^{n-1} \sum_{v=u+1}^n p_{uvr} d_{uv} \leq B \quad (4.2)$$

$$\sum_{r=1}^n x_{ur} = 1, \quad u = 1, \dots, n \quad (4.3)$$

$$p_{uvr} \leq x_{ur}, \quad r = 1, \dots, n, \quad \{u, v = 1, \dots, n | u < v\} \quad (4.4)$$

$$p_{uvr} \leq x_{vr}, \quad r = 1, \dots, n, \quad \{u, v = 1, \dots, n | u < v\} \quad (4.5)$$

$$x_{ur} \leq y_r, \quad u, r = 1, \dots, n, \quad (4.6)$$

$$x_{ur}, p_{uvr} \in \{0, 1\}, \quad r = 1, \dots, n, \quad u, v = 1, \dots, n$$

Constraints (4.1) and (4.2) assure, respectively, that the bounds on the total traffic through each ring and through the federal ring are satisfied. Constraints (4.3) assure that each node is assigned to exactly one ring. Constraints (4.4)-(4.6) impose the congruence of the sets of variables. The objective function is to minimize the number of rings.

It is worth noting that this formulation is not useful for solving the problem with a commercial and general purpose code (e.g. CPLEX), due to the enormous

computing times. A model with only constraints (4.1)-(4.5) and no objective function can be used for checking the feasibility of an instance. We adopted this method in Section 7.

A graph theory model for SRAP can be defined as follows. Let $G = (V, E)$ be the graph representing the SONET network (V is the set of customer's nodes) and d_{ij} the cost of edge $(i, j) \in E$. The problem of finding a solution for SRAP corresponds to finding a partition V_1, V_2, \dots, V_k of V into disjoint sets such that k is minimized and

$$\sum_{u \in V_i} \sum_{v \in V, v \neq u} d_{uv} \leq B, \quad i = 1, \dots, k \quad (4.7)$$

$$\sum_{i=1}^{k-1} \sum_{j=i+1}^k \sum_{u \in V_i} \sum_{v \in V_j} d_{uv} \leq B \quad (4.8)$$

where (4.7) and (4.8) assure, respectively, that the bounds on the total traffic through each ring and through the federal ring are satisfied.

3. Previous Work

Goldschmidt, Laugier and Olinick (Goldschmidt et al. 2003) have proposed three different greedy heuristics for solving SRAP: the *edge-based*, the *cut-based* and the *node-based* heuristics. Both the edge-based and the cut-based heuristics start their computation assigning each node to a different ring, and, at each iteration two different rings are merged if the resulting ring is feasible. In the edge-based heuristic, the edges are ordered by non decreasing weight and the pair of rings connected by the maximum weight edge is chosen. In the cut-based heuristic the pair of rings sharing the maximum traffic is chosen. Both the edge-based and the cut-based heuristics start their computation assigning each node to a different ring.

Finally, the node-based heuristic receives a tentative number of rings k as input, then assigns to each of the k rings a node randomly chosen. At each iteration the heuristic selects the ring with the largest unused capacity, say r , then, the unassigned node with largest traffic toward ring r is chosen and added to the ring. The node-based heuristic is repeated ten times and every time a feasible solution is computed the value of k is decreased by one.

These heuristics are applied by the authors to a set of benchmark instances. A feasible k is computed by running the edge-based and cut-based heuristics and given as input to the node-based heuristic. Note that both the edge-based and cut-based heuristics may have different behavior if different tie break rules are used. Therefore the authors propose to repeat these procedures ten times and to choose randomly how to break ties. When the heuristic solution is feasible

with a value greater than the simple lower bound

$$k_{lb} = \left\lceil \frac{\sum_u \sum_{v \neq u} d_{uv}}{2B} \right\rceil, \quad (4.9)$$

a tentative to prove the optimality of the heuristic solution is performed by means of CPLEX.

4. Basic Tabu Search

We propose a solution algorithm based on a Tabu Search (TS) scheme which explores the space of the possible partitions of graph G looking for a partition associated with a feasible solution of SRAP with a minimum number of rings.

In order to simplify the presentation we introduce the following notation. Let r be a ring index. The *traffic values* with respect to ring r are:

$$\begin{aligned} I_r &= \sum_{u,v \in r, u \neq v} d_{uv} && \text{(total internal traffic)} \\ E_r &= \sum_{u \in r, v \notin r} d_{uv} && \text{(total external traffic)} \\ T_r &= I_r + E_r && \text{(total traffic on the ring } r\text{)} \end{aligned}$$

Note that a ring r is feasible if $T_r \leq B$. Given a node u define:

$$\begin{aligned} W_{u,r} &= \sum_{v \in r, v \neq u} d_{uv} && \text{(total traffic toward node } u \text{ from the ring } r\text{)} \\ W_u &= \sum_{v \neq u} d_{uv} && \text{(total traffic toward node } u\text{)} \end{aligned}$$

Observe that

$$W_u = \sum_r W_{u,r}.$$

Finally define:

$$\begin{aligned} F &= \frac{\sum_r E_r}{2} && \text{(total traffic through the federal ring)} \\ BN &= \max(F, \max_r T_r) && \text{(network bottleneck)} \end{aligned} \quad (4.10)$$

4.1 Neighborhood N_1 and Traffic Value Update

The basic Tabu Search (BTS) uses a *node improvement* neighborhood, say N_1 , which, starting from a given solution (feasible or unfeasible), moves in turn each node from its ring to a different one or to a new ring, such that the resulting ring (existing or new) is feasible.

In Figure 4.2 an example of move generated by N_1 is depicted: the node u is moved from the ring s to the ring t ; the dotted region represents the federal ring and r is a generic ring such that $r \neq s, t$. We observe that moving u from

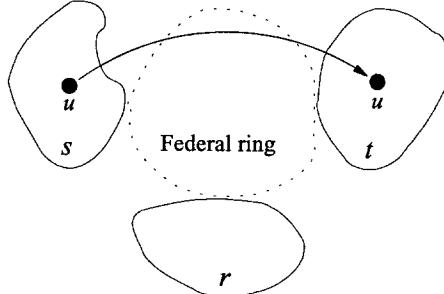


Figure 4.2. Neighborhood N_1 : u is moved from s to t

s to t modifies the following traffic values: T_s , T_t , F and possibly BN .

In Figure 4.3 we depict the traffic values before and after moving node u from ring s to ring t .

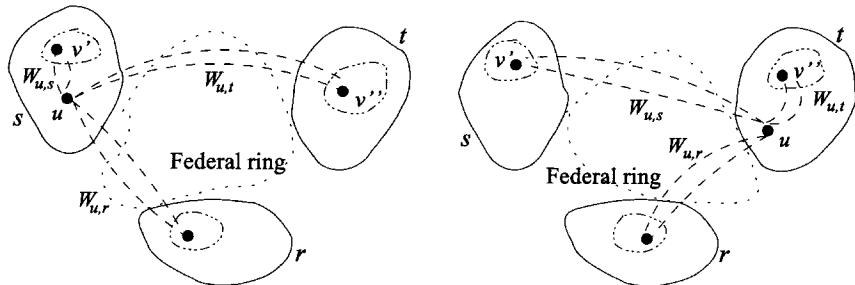


Figure 4.3. Neighborhood N_1 : traffic values

Note that traffic T_s decreases by

$$W_{u,t} + \sum_{r \neq s, t} W_{u,r} = W_u - W_{u,s}.$$

Analogously traffic T_t increases by

$$W_{u,s} + \sum_{r \neq s, t} W_{u,r} = W_u - W_{u,t}.$$

Further note that the traffic F on the federal ring F varies by $W_{u,s} - W_{u,t}$.

Therefore the modified traffic values after moving u from s to t are:

$$\hat{T}_s = T_s - W_{u,s} + W_{u,s} \quad (4.11)$$

$$\hat{T}_t = T_t + W_{u,t} - W_{u,t} \quad (4.12)$$

$$\hat{F} = F - W_{u,t} + W_{u,s} \quad (4.13)$$

where the hat symbol denotes the modified values.

Using the updated values (4.11)-(4.13) we can compute the new bottleneck and evaluate the new solution.

After selecting the best move in N_1 , it is necessary to update the data structures. Considering again Figure 4.3, we observe that $W_{v',s}$ is decreased by $d_{uv'}$ while $W_{v'',t}$ is increased by $d_{uv''}$. Instead traffic $W_{v,r}$ with $v \neq u$ and $r \neq s, t$ does not change, hence

$$\widehat{W}_{v,r} = \begin{cases} W_{v,r} - d_{uv} & r = s, \quad v \in s \\ W_{v,r} + d_{uv} & r = t, \quad v \in t \\ W_{v,r} & \text{otherwise} \end{cases} \quad (4.14)$$

4.2 Tabu Lists

We have used two tabu lists to avoid visiting, for a certain period, already visited solutions.

The first list called *node-list* stores the nodes visited in the last iterations. A node contained in node-list is not taken into account when exploring the neighborhood.

The second list called *node-from-list* stores the moved nodes within the identifier of the original ring from which the node has been removed. During the exploration of the neighborhood we avoid inserting a node stored in this list into its original ring.

We maintain the length l_1 of node-list smaller than the length l_2 of node-from-list in order to fix a node for l_1 iterations (node-list) and then to forbid its return to the original ring for l_2 other iterations (node-from-list).

The list length varies during the search (see e.g. Dell'Amico and Trubian, 1998) between $\frac{1}{2}$ and $\frac{3}{2}$ of a given *tabu-tenure* value. We decrease l_1 and l_2 to intensify the search within promising regions, whilst we increase them to speed up the leaving from not promising regions. We define an *improving phase* a set of Δip consecutive iterations which lower the objective function. On the contrary, a *worsening phase* is a set of Δwp consecutive iterations such that the objective function value does not improve. The initial values of l_1 and l_2 are respectively equal to *tabu-tenure*₁ and *tabu-tenure*₂ and vary as follows:

$$l_i = \max(l_i - 1, \frac{1}{2} \text{tabu-tenure}_i), \quad i = 1, 2 \quad (4.15)$$

after an improving phase and

$$l_i = \min(l_i + 1, \frac{3}{2} \text{tabu-tenure}_i), \quad i = 1, 2 \quad (4.16)$$

after a worsening phase. On the basis of a set of preliminary experiments, we set the parameter values as reported in Table 4.1.

Table 4.1. Tabu List: values of parameters controlling the length of tabu lists

values	l_1	l_2
tabu-tenure	5	10
Δip	5	5
Δwp	3	3

4.3 Objective Functions

In order to drive the search toward feasible and good solutions we have to take into account both the number of rings k and the value of the bottleneck BN . We have defined the following tentative objective functions:

$$\begin{aligned} z_1 &= k + \max\{0, BN - B\}, \\ z_2 &= z_1 + \begin{cases} k T_c & \text{if a new ring } c \text{ has been created} \\ 0 & \text{otherwise} \end{cases}, \\ z_3 &= kB + BN. \end{aligned}$$

Function z_1 leads the search toward solutions with a small number of rings while penalizing unfeasible solutions. Function z_2 modifies z_1 by strongly penalizing moves creating a new ring. Finally, function z_3 leads the search toward solutions with a minimum value of k , and among these toward solutions with minimum bottleneck value..

We observed that the expected solution improvement depends on the feasibility status of the current solution. For example, if the current solution is feasible and a move creates a new feasible solution, then we are interested in minimizing k . On the contrary, if the current solution is unfeasible and a move creates a new unfeasible solution, then we would like to minimize BN . Therefore we have studied a further multiobjective function which drives the search taking into account these different cases. Denoting with \widetilde{BN} the bottleneck

values associated with unfeasible solutions, we define:

$$z_4 = \begin{cases} k B + BN & \text{if the move starts from a feasible solution} \\ & \text{and generates a feasible one} \\ (k+1)\widetilde{BN} & \text{if the move starts from a feasible solution} \\ & \text{and generates an unfeasible one} \\ k B & \text{if the move starts from an unfeasible solution} \\ & \text{and generates a feasible one} \\ n \widetilde{BN} & \text{if the move starts from an unfeasible solution} \\ & \text{and generates an unfeasible one} \end{cases}$$

Since for a feasible solution $k B + BN \leq (k+1)B$, $(k+1)B \leq (k+1)\widetilde{BN}$ and $k B \leq n \widetilde{BN}$, then z_4 drives the search from unfeasible solutions to feasible ones. Note that z_4 is a sort of extension of z_3 .

4.4 Procedure BTS

After reading the input instance and initializing the main data structures, the Basic Tabu Search BTS starts a main cycle which is repeated until *MaxTime* seconds have elapsed.

At each iteration, the best move *Move* is selected by the function **BestMove**. If *Move* $\neq \emptyset$, the current solution *Sol* is modified (**Update**(*Move*, *Sol*)) and the best solution *Best* is updated if $z(\text{Sol}) < z(\text{Best})$.

The update phase concerns both the parameters controlling the length of tabu lists and the traffic values: the parameters are updated with respect to the objective function improvement ($z(\text{Sol}) < \bar{z}$) while the traffic values are updated according to the equations (4.11)-(4.14) by procedure **UpdateTrafficValues**.

The tabu lists are updated by the **UpdateTabuList** procedure: if *Move* $\neq \emptyset$, *Move* is inserted in the node-list and node-from-list, otherwise a dummy move is inserted. Then, if $ip = \Delta ip$ or $wp = \Delta wp$, the length of the tabu lists are updated as in (4.15) and (4.16).

The pseudo-code of the BTS algorithm is as follows.

```
BTS( Instance, MaxTime ) {
    GetData( Instance );
    Init( Move, Sol, Best );
    Compute( Wu,r, Wu, Tr, F, BN );
    Elapsed := 0; Starting := Now();
    while ( Elapsed  $\leq$  MaxTime ) {
        Move := BestMove( Sol, N1 )
        if ( Move  $\neq \emptyset$  ) {
             $\bar{z}$  := z(Sol);
```

```

 $Sol := \text{Update}( Move, Sol );$ 
 $\text{if } ( z(Sol) < z(Best) ) \text{ } Best:=Sol;$ 
 $\text{if } ( z(Sol) < \bar{z} ) \{ ip := ip + 1; wp := 0; \}$ 
 $\text{else } \{ wp := wp + 1; ip := 0; \}$ 
 $\text{UpdateTrafficValues}( Move, W_{u,r}, T_r, F, BN );$ 
 $\};$ 
 $\text{UpdateTabuList}( Move, ip, wp );$ 
 $Elapsed := \text{Now}( ) - Starting;$ 
 $\};$ 
 $\text{return}( Best );$ 
 $\}.$ 

```

At the end of the computation the best solution computed is returned as output of the procedure.

4.5 Complexity of Neighborhood Evaluation

The neighborhood evaluation is the most time consuming component of the whole algorithm. We observe that in the worst case each of the n nodes can be moved into $n - 1$ rings. Since the computation of the best move requires the assessment of all the possible moves, the neighborhood evaluation of N needs to generate $O(n^2)$ moves. We also observe that all the objective functions proposed in §4.3 require the computation of BN . As described in §4.1, moving u from s to t can increase or decrease the BN value.

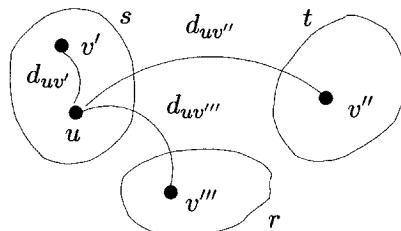


Figure 4.4. Neighborhood N_1 : variation of BN value

We illustrate this case in Figure 4.5 where the node u , which is connected only with $v' \in s$, $v'' \in t$ and $v''' \in r$, is moved from s to t . Suppose that: $F = 80$, $T_t = 85$, $T_r = 100$, $T_s = BN = 110$, $d_{uv'} = 15$, $d_{uv''} = 5$ and

$d_{uv'''} = 7$. Applying the equations (4.11)-(4.13), we obtain

$$\hat{T}_s = T_s - (d_{uv''} + d_{uv'''}) = 98$$

$$\hat{T}_t = T_t + (d_{uv''} + d_{uv'''}) = 97$$

$$\hat{F} = F + (d_{uv'} - d_{uv''}) = 90$$

and T_r is not modified. The new value of BN is now equal to $T_r = 100$, hence it is decreased. On the other side, if $T_t = 100$ the new value of BN increases to 112.

As shown in the above example, the computation of BN involves not only \hat{T}_t and F , but also all the T_r values with $r \neq s, t$, hence an $O(n)$ time is required to compute BN after each node movement and the neighborhood evaluation is done in $O(n^3)$ in the worst case.

This complexity can be reduced to $O(n^2 \log n)$ maintaining a heap data structure to order the T_r values. Some special cases (e.g. when $BN = T_t$) can be also considered to reduce the average computation time. However, in the practical application (see Section 7), the number of rings is small and no special implementation is required.

5. Tabu Search with Strategic Oscillation

The computational experiments performed (see Section 7, Table 4.4) show that procedure BTS is not very effective to solve SRAP hence we have tried to improve it by means of a simple variable neighborhood strategy consisting in a further neighborhood N_2 and a switching rule to substitute N_1 to N_2 and vice versa.

Employing only the neighborhood N_1 , BTS often terminates its computation returning a solution containing k large rings and a single ring, say r , containing few nodes. In these cases, we expect that a local optimal solution with k rings can be obtained spreading the nodes belonging to r over the remaining k rings. However BTS is not able to find this solution since moving all but one nodes belonging to r increases the bottleneck while it maintains the same number of rings.

The introduction of the *empty minimum cardinality ring* neighborhood, say N_2 , is aimed at dealing with this kind of solution: a node belonging to r is compulsively moved to one of the remaining rings without taking into account the feasibility of the new solution.

The switching of N_1 to N_2 is controlled by two conditions:

condition C_1 : a number i_{ni} of consecutive not improving iterations has been performed;

condition C_2 : the current solution is feasible.

At each iteration we check condition C_1 , C_2 or both (see Section 7 for implementation details) and, if necessary, we transform the current solution into a new one through neighborhood N_2 (**NewSol()**).

We insist with neighborhood N_2 until the minimum cardinality ring has a single node. At this point, N_2 is applied once again and we return to N_1 for the next iterations.

This is a particular implementation of the strategic oscillation technique in which we have only two neighborhoods and the second one is applied for a fixed number of iterations, namely the cardinality of the minimum cardinality ring.

A possible pseudo-code of our TS with Strategic Oscillation (TSSO) algorithm is as follows.

```

TSSO( Instance, MaxTime ) {
    GetData( Instance );
    Init( Move, Sol, Best );
    Compute( Wu,r, Wu, Tr, F, BN );
    Elapsed := 0; Starting := Now( );
    switch := FALSE;
    while ( Elapsed  $\leq$  MaxTime ) {
        Move := BestMove( Sol, N1 )
        if ( Move  $\neq$   $\emptyset$  ) {
             $\bar{z}$  := z(Sol);
            Sol := Move( Move, Sol );
            if ( z(Sol) < z(Best) ) Best:=Sol;
            if ( z(Sol) <  $\bar{z}$  ) {
                ip := ip + 1; wp := 0; ini := 0;
            } else {
                wp := wp + 1; ip := 0; ini := ini + 1;
            }
            UpdateTrafficValues( Move, Tu,r, Tr, F, BN );
        };
        UpdateTabuList( Move, ip, wp, ini );
        if ( Check( C1, C2 ) )
            Sol := NewSol( Sol, N2 );
        Elapsed := Now( ) - Starting;
    };
    return( Best );
}.

```

Note that procedure **NewSol()** consists of a number of moves with neighborhood N_2 which spread the nodes belonging to the minimum cardinality ring to the remaining rings.

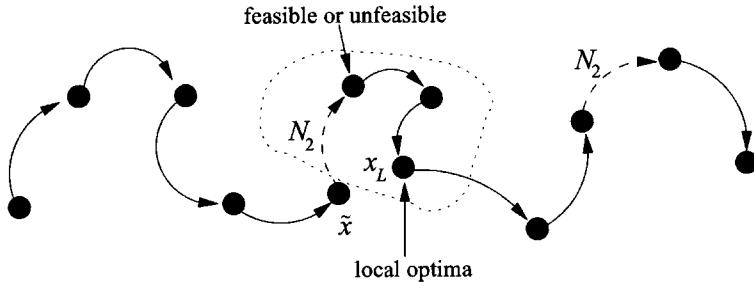


Figure 4.5. Switching N_1 and N_2 : exploration of the solution space

A typical exploration of the solution space obtained by switching N_1 and N_2 is depicted in Figure 5 where a black circle is a solution, a solid arc is a set of moves generated by N_1 and a dashed arc is a set of moves generated by N_2 . After some moves, the search reach \tilde{x} and the neighborhoods are switched; a new solution is obtained by spreading the nodes belonging to r over the remaining k rings.

The technique of switching N_1 and N_2 is similar to the *strategic oscillation* described in Glover and Laguna, 1997, which operates by orienting moves in relation to a *critical level*. A critical level usually identifies a solution such that the normal search would not be able to improve. For example, a desired or undesired form of a graph structure. When the critical level is reached, the strategic oscillation changes the rules of selecting moves in such a way that the critical level is crossed, that is the solution should be improved. In Figure 5, the critical level can be represented by the dotted region around the local optima x_L .

Finally, we note that all the possible moves generated by N_2 are $O(n^2)$, hence also the evaluation of N_2 has computational complexity $O(n^3)$.

6. Other Intensification and Diversification Strategies

Path Relinking (PR) (Glover, 1999), eXploring Tabu Search (XTS) (Dell'Amico et al. 1999) and Scatter Search (SS) (Glover, 1997, Glover, 1999, Glover et al. 2000) are strategies aimed to intensify the search into promising regions or to diversify it generating new restarting solutions. They are successfully applied to several optimization problems. We have implemented all these techniques to compare their results to those obtained by TSSO.

We use the main ingredients of PR and XTS to differentiate the search within our tabu search when necessary. More precisely, we use the same framework of TSSO by replacing procedure **NewSol** with a method for generating a diverse solution based on PR or XTS.

We implemented SS along the guidelines of Laguna, 2001. SS can be intended as a general methodology in which the intensification and diversification phases are driven by a set of solutions. These solutions are then combined to generate further solutions. Our main concern is to exploit the techniques embedded in SS to improve the quality of the solution computed by BTS.

This Section is organized in three subsections describing the proposed algorithms: PR (§6.1), XTS (§6.2) and SS (§6.3).

6.1 Path Relinking

A *relinking path* is a set of moves connecting two different solutions. It can be defined as the *more direct route* or the *fewest number of moves* to connect two different solutions. Clearly, a relinking path is also a sequence of solutions.

An example is reported in Figure 4.6: the original path is shown by the solid line while the relinking one is shown by the dashed line.

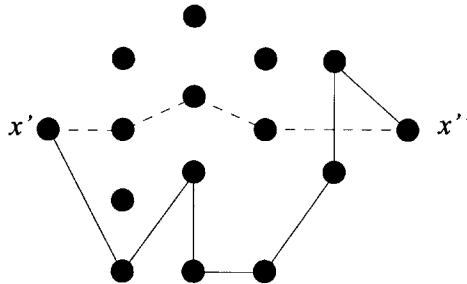


Figure 4.6. PR: example from Glover et al. 2000

The basic idea of PR is to generate a new solution by choosing one of those along the relinking path.

We consider two different assignments x' and x'' of nodes to rings. The hamming distance

$$H(x', x'') = \sum_u \sum_r \frac{|x'_{ur} - x''_{ur}|}{2}. \quad (4.17)$$

returns the minimum number of moves required to get x'' from x' . Let M_{PR} be this set of moves. Looking at Figure 4.6, M_{PR} contains all the four moves denoted by the dashed line.

We propose two PR implementations. The first one, say PR1, generates the set M_{PR1} relinking the current solution to the best one. Then, $\lfloor M_{PR1}/2 \rfloor$ moves have been selected from M_{PR1} in such a way that the objective function is minimized.

The second PR implementation proposed, say PR2, try to augment the diversification by considering more paths at the same time. Instead of relinking the current solution to the best one, we maintain an *elite solution set*, say Δ , containing a fixed number of best solutions computed during the search and we generate all the paths from the current solution to the solutions in Δ . This implementation has been proposed in Laguna et al. 1999.

An example of PR2 is depicted in Figure 4.7 where two different relinking paths are generated by two solutions belonging to Δ .

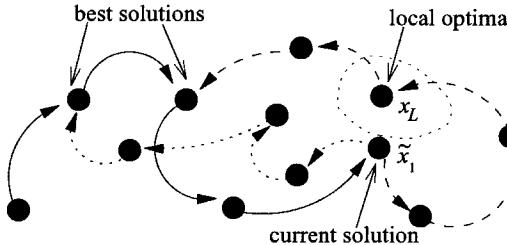


Figure 4.7. PR2: relinking paths

6.2 eXploring Tabu Search

The N_1 evaluation can generate many solutions which have quite similar objective function values. The basic idea of XTS is to adopt a *long term memory structure* to record them. Then, any of these solutions can be used to restart the search.

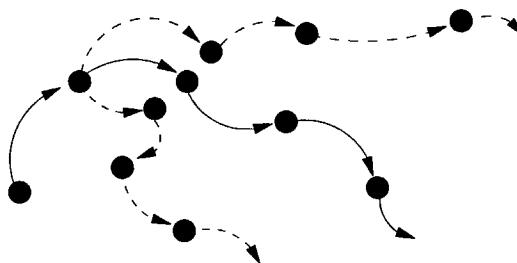


Figure 4.8. XTS: restarting and exploration with two stored solutions

An example of XTS is given in Figure 4.8: the dashed lines represent the solution space explored restarting the search from two different recorded solutions.

As already done in Dell'Amico and Trubian, 1998, our XTS implementation stores the second best not-tabu solution computed during each N_1 evaluation.

These solutions are collected in a fixed length list called *second-list* (*SL*) with a copy of the tabu lists and the search parameters in order to restart the search with the same conditions encountered when the solution was inserted in the list *SL*.

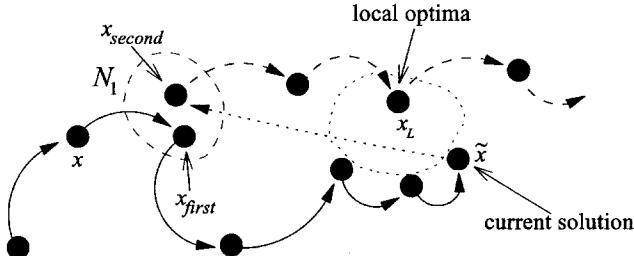


Figure 4.9. XTS: expected behavior of the search

Figure 4.8 reports an example of restarting phase employing XTS. Exploring N_1 from x we reach x_{first} and x_{second} : solution x_{first} is selected whereas solution x_{second} is stored in *SL*. When we reach \tilde{x} , condition C_1 and C_2 tell us to diversify the search so we restart the search from x_{second} .

6.3 Scatter Search

SS is a general methodology in which the intensification and diversification phases are driven by a set of solutions called *Reference Set* which is composed of a subset of *high quality* solutions (*HQ*) and by a subset of *diverse* solutions (*DV*).

Let:

$$D_u(x, x^*) = \begin{cases} 0 & \text{if } u \text{ belongs in the same ring both in } x \text{ and } x^* \\ 1 & \text{otherwise} \end{cases}.$$

The function measuring the *diversity* of x with respect to the best solution x^* is

$$D(x, x^*) = \sum_{u=1}^n D_u(x, x^*). \quad (4.18)$$

Our implementation traces the general scheme proposed in Laguna, 2001. We report only the methods which are specific for SRAP:

- *Diversification Generation Method.* We iteratively generate pairs of random solutions with $h = 2, \dots, k_{lb} + 10$ rings. In the first solution, we randomly assign each node to one of the h rings. The second solution is built from the first one by moving each node to a randomly chosen

different ring (when $h = 2$ a node is moved to the other ring with 0.5 probability).

- *Solution Combination Method.* We combine the solutions belonging to the Reference Set to obtain a new, possibly better, solution by means of the following score function:

$$s_{u,r} = \frac{\sum_{j \in S} z(x^j) x_{ur}^j}{\sum_{j \in S} z(x^j)} \quad (4.19)$$

where $z(x^j)$ denotes the objective value of the j -th solution x^j belonging to a subset S of the Reference Set. Then, the new solution \tilde{x} is equal to $\bar{x}_{u,r} = 1$ if $s_{u,r} = \max_r s_{u,r}$, $\bar{x}_{u,r} = 0$ otherwise.

- *Improvement Method.* We tested two different techniques to enhance the quality of the solutions generated: the first is a pure Local Search (LS) based on neighborhood N_1 ; the second one is our procedure BTS.

The objective functions used with SS are z_1 and a modified version of z_4 :

$$\hat{z}_4 = \begin{cases} k B + BN & \text{if the solution is feasible} \\ n BN & \text{otherwise.} \end{cases}$$

Note that the original z_4 cannot be used with SS since it is based on the concept of move, but it is used within the improvement method.

Each new solution \tilde{x} is inserted in HQ if its objective function value is better than that of the worst solution in HQ . On the other side, a new solution is inserted in DV if

$$\min_{x \in HQ} D(\tilde{x}, x) > \min_{y \in D} \left\{ \min_{x \in HQ} D(y, x) \right\}.$$

At the beginning, procedure **Init** initializes the reference set with the feasible or unfeasible solutions computed by the diversification generation method. During each iteration, the solution combination method (**CombineSol**) generates some new solutions which could be inserted in the reference set (**InsertRefSet**) after have been improved by **ImproveSol** procedure. The iteration terminates when none of the new combined solutions is inserted. Before starting a new iteration, SS tries to insert in the reference set some new solutions generated by the diversification generation method (**UpdateRefSet**). A possible pseudo-code of our SS is as follows.

```
SS( Instance, MaxIters ) {
    GetData( Instance );
    ...
```

```

Init( RefSet, HQ, DV );
Elapsed := 0; Starting := Now( ); Iters := 0;
for ( Iters 1 to MaxIter ) do {
    Inserted := TRUE;
    while ( Inserted ) {
        Inserted := FALSE;
        for ( h = 2 to 5 ) do
            for each ( h-upla  $x_1, \dots, x_h \in HQ \cup DV$  ) do {
                NewSol := CombineSol(  $x_1, \dots, x_h$  );
                ImproveSol( NewSol );
                Inserted := InsertRefSet( NewSol );
            };
        };
        if ( Iters  $\leq$  MaxIter )
            UpdateRefSet( RefSet, HQ, DV );
    };
    return( HQ );
}.

```

7. Computational Results

The algorithms described in the previous Sections were implemented in ANSI C language and tested on a Linux PC Pentium III/600 with 524 Megabytes of main memory. Computational experiments have been made using two sets of benchmark instances, referred to as *B1* and *B2*.

7.1 Benchmark Instances *B1*

The set *B1* has been generated by Goldschmidt et al. 2003, and used to test their greedy algorithms. It contains 160 instances separated in two sets of instances:

- 80 *geometric* instances representing *natural cluster*, that is the fact that customers try to communicate more with their close neighbors than with their distant ones;
- 80 *random* instances.

The traffic demand between two nodes is determined by a discrete, uniform random variable indicating the equivalent number of T1 lines (a T1 line has a capacity of 1.544 Mbs) required for the estimated traffic. Each set contains both *high* and *low* demand graphs. The 40 high demand graphs have the ring capacity *B* set to 622 Mbs and demand generated in [11, 17] while the 40 low demand instances have *B* = 155 Mbs and demand generated in [3, 7]. Finally, the dimension of the instances is 15, 25, 30 and 50 nodes (10 instances each).

The reported results concern only the 118 instances proved to be feasible by optimally solving SRAP using CPLEX (the remaining 42 instances are unfeasible). The feasible instances have been solved by CPLEX within an average computing time of 20 minutes (with a maximum of 23 hours) whereas the unfeasibility of an instance has been proven within 57 hours, on average.

We have also implemented the greedy randomized procedures of Goldschmidt et al. 2003, and run each procedure with 10^5 restarts. With these algorithms 110 out of 118 instances are solved. Note that the three greedy algorithms with 10^5 restarts run for about 1 minute to solve an instance with 15 nodes and more than 1 hour for an instance with 50 nodes.

The maximum running time has been set to 3 seconds for all algorithms, except SS which terminates after 5 iterations for the version with LS and 1 iteration for the version with BTS.

In Table 4.2 we report the average computation time in milliseconds (avg. ms) and the number of instances optimally solved by TSSO (#). Procedure TSSO was tested using both conditions C_1 and C_2 , one of them or none. Parameter i_{ni} in C_1 was set to 10. Using the multiobjective function z_4 , all the feasible

Table 4.2. TSSO: avg. computation time in milliseconds and # of instances optimally solved

C_1, C_2	z_1		z_2		z_3		z_4	
	avg. ms	#	avg. ms	#	avg. ms	#	avg. ms	#
Y, Y	58.0	116	67.6	112	433.8	8	96.4	118
Y, N	33.4	99	37.4	96	—	6	41.7	100
N, Y	54.1	116	53.3	113	—	1	60.4	118
N, N	98.6	98	110.1	98	393.3	88	286.1	90

instances are solved setting both conditions C_1 and C_2 or only C_2 . Using only C_2 TSSO has the best average computation time.

Table 4.3 highlights the results concerning the instances not solved with the greedy algorithms. We report the simple lower bound value k_b (defined by (4.9)) and the optimal solution value k^* (obtained with CPLEX). The results show the effectiveness of z_1 and z_4 to drive the search while z_2 fails to find a feasible solution for two instances and one is not optimal; z_3 fails to solve all the instances proposed.

The results reported in Table 4.4 highlights the effectiveness of the diversification method based on neighborhood N_2 to improve the quality of the solutions computed by BTS: BTS solves the 84,5% and the 76,3% of instances solved by TSSO with only condition C_2 when the objective function used is z_1 and z_4 , respectively.

Table 4.5 reports the results obtained running PR1, PR2 and XTS. The parameters of these procedures have been defined through preliminary experiments

Table 4.3. TSSO: instances not solved by the greedy algorithms

instance	n	m	k_{lb}	k^*	z_1		z_2		z_4	
					k	ms	k	ms	k	ms
RH.15.3	15	53	2	3	3	0	-	-	3	0
RH.30.10	30	64	3	3	3	280	4	10	3	60
RH.50.3	50	89	4	4	4	1240	4	1950	4	280
RH.50.4	50	89	3	4	4	240	4	330	4	250
RH.50.7	50	90	4	5	5	270	5	320	5	260
RL.25.1	25	51	3	4	4	30	-	-	4	30
RL.30.3	30	58	3	4	4	20	4	10	4	10
RL.30.6	30	56	3	4	4	20	4	20	4	20

Table 4.4. Comparison of the performances of BTS and TSSO

	z_1		z_2		z_3		z_4	
	avg. ms	#	avg. ms	#	avg. ms	#	avg. ms	#
BTS	98.6	98	110.1	98	393.3	88	286.1	90
TSSO	54.1	116	53.3	113	-	1	60.4	118

as follows. Only condition C_1 is tested, $i_{ni} = 50, 75$ and 150 for PR1, PR2 and XTS, respectively. The cardinality of the elite set of PR2 was fixed to 15 and the length of the second-list was set to n . Algorithm PR2 dominates the other methods, but the performances of the three algorithms are similar. Also note that, in contrast with the behaviour observed for TSSO and BTS, method PR and XTS has significantly better performances with z_1 rather than z_4 and without checking condition C_2 . We observe also that PR1, PR2 and XTS solves

Table 4.5. Performances of PR1, PR2 and XTS.

z	PR1		PR2		XTS	
	avg. ms	#	avg. ms	#	avg. ms	#
z_1	94.2	102	114.4	106	152.9	103
z_2	84.8	99	127.7	104	191.5	98
z_3	412.5	97	537.9	102	368.2	88
z_4	388.0	97	245.5	97	273.8	87

less instances than the greedy algorithms.

Table 4.6 reports the gaps, in terms of rings, for the instances not optimally solved by PR1, PR2 and XTS employing objective function z_1 . It is interesting to observe that the maximum gap for PR2 is one, whereas PR1 and XTS have gaps up to 5. As expected, PR2 exhibits superior performance.

Table 4.6. Gaps for instances not optimally solved by PR1, PR2 and XTS using z_1

Gap (# of rings)	PR1	PR2	XTS
$k^* + 1$	11	12	9
$k^* + 2$	2	0	3
$k^* + 3$	3	0	3
Total	16	12	15

Table 4.7 reports the results obtained by SS using LS (SS-LS) or BTS (SS-BTS) as improvement method.

Table 4.7. SS: results of SS using as improvement methods LS or BTS.

z	SS-LS		SS-BTS	
	avg. ms	#	avg. ms	#
z_1	566.1	87	1468.2	116
z_4	1100.4	84	8593.1	113

For procedure SS-LS we set the number of high quality solutions and the number of diverse solutions to 15 and we gave a limit of 5 iterations. For SS-BTS, the cardinality of the two sets was fixed to 20, one iteration was performed and a limit of 20 milliseconds was given to each run of BTS.

Using SS-LS we obtain very poor performances with a significant computing effort. Procedure SS-BTS has better performances than the pure BTS, but with very high computing times. This confirms the effectiveness of the intensification and diversification strategies embedded in SS.

7.2 Benchmark Instances $B2$

We have generated further benchmark instances in order to provide more computational results for our algorithms. The main idea is to generate *harder* instances with respect to those belonging to $B1$. We define as hard a feasible instance such that the greedy algorithms are not able to find the optimal solution, within a reasonable computing time.

We have generated 230 instances by taking the 42 unfeasible instances of $B1$ and randomly eliminating traffic demands until we reduce the total traffic of an amount greater than or equal to the difference between the value of the bottleneck ring in the (unfeasible) solution computed by TSSO and the bound B . We insert an instance in $B2$ if 10^3 restarts of the greedy algorithm determine a solution worse than the best one computed by CPLEX.

The distribution of the new instances among geometric and random graphs, high or low demand and the various values of n , are reported in tables 4.8a and 4.8b. Note that all the 230 instances are feasible. We tried to solve these instances giving the greedy algorithm a limit of 10^5 restarts, but we obtained only 46 feasible solutions.

Table 4.8a. Set $B2$: Distribution of geometric and random instances in $B2$

	High	Low	Total
Geometric	70	70	140
Random	20	70	90
Total	90	140	230

Table 4.8b. Set $B2$: Distribution of the instances by cardinality

	15	25	30	50	Total
n	50	40	50	90	230

Table 4.9 reports the results obtained by TSSO algorithm with objective functions z_1 and z_4 and the same parameter values used for the instances in $B1$.

Table 4.9. TSSO: results for instances in $B2$

z	TSSO					
	C_1 and C_2			only C_2		
	avg. ms	#	BTS	avg. ms	#	BTS
z_1	137.5	230	12	117.3	230	15
z_2	121.8	190	13	112.5	190	15
z_3	409.4	16	-	-	0	-
z_4	135.8	225	14	110.7	225	17

We report the average computation time in milliseconds, the number of feasible solutions obtained and the number of solutions with fewer rings than those computed by BTS.

Although TSSO with z_4 solves all the instances in $B1$, it does not find a feasible solution for 5 instances in $B2$. However, TSSO with z_1 always finds a good feasible solution for all the instances in $B2$. We note that the average computation time is increased with respect to that of Table 4.4 but this is mainly due to the larger number of instances with 50 nodes.

Table 4.10 reports the results obtained by PR1, PR2 and XTS on set $B2$ using the same parameter adopted for solving the instances in $B1$ again. The results confirm the effectiveness of PR2 with respect to PR1 and XTS.

Table 4.11 reports the performances of SS obtained adopting the same parameter values used to solve the instances in $B1$. The results confirm the effectiveness of SS with BTS as improvement method.

Table 4.10. Performances of PR1, PR2 and XTS.

z	PR1			PR2			XTS		
	avg. ms	#	BTS	avg. ms	#	BTS	avg. ms	#	BTS
z_1	129.4	228	14	198.7	230	17	224.3	227	2
z_2	127.1	195	20	174.9	193	23	220.9	198	13
z_3	180.3	222	10	225.7	223	27	241.4	220	8
z_4	138.8	229	22	201.9	230	26	200.7	226	13

Table 4.11. SS: performances of SS on set B2

z	SS-LS			SS-BTS		
	avg. ms	#	BTS	avg. ms	#	BTS
z_1	1022.9	118	0	2638.7	229	22
z_4	2432.6	165	2	3602.1	229	32

8. Conclusions

We have considered a basic problem arising in the designing of SONET networks, which can be formulated as a graph partitioning problem with capacity constraints. We first introduced a Basic Tabu Search, then we improved it by including diversification strategies based on Strategic Oscillation, Path Relinking and eXploring Tabu Search. A pure Scatter Search algorithm has been also implemented. The six resulting procedures have been tested through extensive computational experiments using benchmark instances both from the literature and new ones. As competitors we considered the only methods available in the literature, namely three greedy algorithms, which have been executed with 10^5 restarts. The computational experiments show the superiority of the Tabu Search method enhanced with a strategic oscillation. Moreover comparisons of the different diversification methods are outlined.

Acknowledgements

This research was supported by Ministero dell'Istruzione, dell'Università e della Ricerca (MIUR), Italy and by Consiglio Nazionale delle Ricerche (CNR), Italy.

References

- Dell'Amico, M., A. Lodi and F. Maffioli (1999) "Solution of the Cumulative Assignment Problem with a Well-Structured Tabu Search Method," *Journal of Heuristics*, 5(2):123–143.

- Dell'Amico, M. and M. Trubian (1998) "Solution of Large Weighted Equicut Problems," *European J. Oper. Res.*, 106(2-3):500–521.
- Glover, F (1997) A Template for Scatter Search and Path Relinking. In J. Hao, E. Lutton, E. Ronald, M. Schoenauer and D. Snyers, Editors, *Lecture Notes in Computer Science*, 1363:13–54.
- Glover, F (1999) Scatter Search and Path Relinking. In D. Corne, M. Dorigo and F. Glover, Editors, *New Ideas in Optimization*, 297–316. McGraw Hill.
- Glover, F. and M. Laguna (1997) *Tabu Search*. Kluwer Academic Publishers, Boston.
- Glover, F., M. Laguna and R. Martí (2000) "Fundamentals of Scatter Search and Path Relinking," *Control and Cybernetics*, 39(3):653–684.
- Goldschmidt, O., A. Laugier and E.V. Olinick (2003) "Sonet/SDH Ring Assignment with Capacity Constraints," *Discr. Appl. Math.*, 129:99–128.
- Laguna, M (2001) Scatter Search. In P.M. Pardalos and M.G.C. Resende, Editors, *Handbook of Applied Optimization*. Oxford Academic Press.
- Laguna, M., R. Martí and V. Campos (1999) "Intensification and Diversification with Elite Tabu Search Solutions for the Linear Ordering Problem," *Computers Oper. Res.*, 26:1217–1230.
- Laguna, M. (1994) "Clustering for the Design of SONET Rings in Interoffice Telecommunications," *Management Science*, 40:1533–1541.

Chapter 5

A VERY FAST TABU SEARCH ALGORITHM FOR JOB SHOP PROBLEM

Józef Grabowski¹, and Mieczysław Wodecki²

¹*Wrocław University of Technology, Institute of Engineering Cybernetics, Janiszewskiego 11-17,
50-372 Wrocław, Poland, grabow@ict.pwr.wroc.pl;*

²*University of Wrocław, Institute of Computer Science, Przesmyckiego 20, 51-151 Wrocław,
Poland, mwd@ii.uni.wroc.pl*

Abstract This paper deals with the classic job-shop scheduling problem with makespan criterion. Some new properties of the problem associated with blocks are presented and discussed. These properties allow us to propose a new, very fast local search procedure based on a tabu search approach. The central concepts are lower bounds for evaluations of the moves, and perturbations that guide the search to the more promising areas of solution space, where "good solutions" can be found. Computational experiments are given and compared with the results yielded by the best algorithms discussed in the literature. These results show that the algorithm proposed solves the job-shop instances with high accuracy in a very short time. The presented properties and ideas can be applied in many local search procedures.

Keywords: Job-Shop Scheduling, Makespan, Heuristics, Tabu Search

1. Introduction

The paper deals with the job-shop problem, which can be briefly presented as follows. There is a set of jobs and a set of machines. Each job consists of a number of operations, which are to be processed in a given order, each on a specified machine for a fixed duration. The processing of an operation can not be interrupted. Each machine can process at most one operation at a time. We want to find the schedule (the assignment of operations to time intervals on machines) that minimizes the *makespan*.

The job-shop scheduling problem, although relatively easily stated, is NP-hard, and is considered one of the hardest problems in the area of combinatorial optimization. This is illustrated by the fact that a classical benchmark problem

(FT10) of 10 jobs and 10 machines, proposed by Fisher and Thompson (1963), remained unsolved (to optimality) for more than a quarter of a century. Many various methods have been proposed, ranging from simple and fast dispatching rules to sophisticated branch-and bound algorithms. For the literature on job-shop scheduling, see Carlier and Pinson (1989), Morton and Pentico (1993), Nowicki and Smutnicki (1996b), Vaessens, Aarts and Lenstra (1996), Aarts and Lenstra (1997), Balas and Vazacopoulos (1998), and Pezzella and Merelli (2000), and their references. In this paper, we present new properties and techniques which allows us to solve the large-size job-shop instances with high accuracy in a relatively short time.

The paper is organized as follows. In Section 2, the notations and basic definitions are introduced. Section 3 presents the new properties of the problem, moves and neighbourhood structure, methods to evaluate the moves, search strategy, dynamic tabu list, perturbations, and algorithm based on a tabu search approach. The central concepts are lower bounds for evaluations of the moves, and perturbations used during the performance of the algorithm. Computational results are shown in Section 4 and compared with those taken from the literature. Section 5 gives our conclusions and remarks.

2. Problem Formulation and Preliminaries

The job-shop problem can be formally defined as follows, using the notation by Nowicki and Smutnicki (1996b). There are: a set of jobs $J = \{1, 2, \dots, n\}$, a set of machines $M = \{1, 2, \dots, m\}$, and a set of operations $O = \{1, 2, \dots, o\}$. Set O decomposes into subsets (chains) corresponding to the jobs. Each job j consists of a sequence of o_j operations indexed consecutively by $(l_{j-1} + 1, \dots, l_{j-1} + o_j)$, which are to be processed in order, where $l_j = \sum_{i=1}^j o_i$, is the total number of operations of the first j jobs, $j = 1, 2, \dots, n$, ($l_0 = 0$), and $o = \sum_{i=1}^n o_i$. Operation x is to be processed on machine $\mu_x \in M$ during processing time p_x , $x \in O$. The set of operations O can be decomposed into subsets $M_k = \{x \in O | \mu_x = k\}$, each containing the operations to be processed on machine k , and $m_k = |M_k|$, $k \in M$. Let permutation π_k define the processing order of operations from the set M_k on machine k , and let Π_k be the set of all permutations on M_k . The processing order of all operations on machines is determined by m -tuple $\pi = (\pi_1, \pi_2, \dots, \pi_m)$, where $\pi \in \Pi_1 \times \Pi_2 \times \dots \times \Pi_m$.

It is useful to present the job-shop problem by using a graph. For the given processing order π , we create the graph $G(\pi) = (N, R \cup E(\pi))$ with a set of nodes N and a set of arcs $R \cup E(\pi)$, where:

- $N = O \cup \{s, c\}$, where s and c are two fictitious operations representing dummy “start” and “completion” operations, respectively. The weight of node $x \in N$ is given by the processing time p_x , ($p_s = p_c = 0$).

- $$R = \bigcup_{j=1}^n \left[\bigcup_{i=1}^{o_j-1} \{(l_{j-1} + i, l_{j-1} + i + 1)\} \cup \{(s, l_{j-1} + 1)\} \right. \\ \left. \cup \{(l_{j-1} + o_j, c)\}\right].$$

Thus, R contains arcs connecting consecutive operations of the same job, as well as arcs from node s to the first operation of each job and from the last operation of each job to node c .

- $$E(\pi) = \bigcup_{k=1}^m \bigcup_{i=1}^{m_k-1} \{(\pi_k(i), \pi_k(i+1))\}.$$

Thus, arcs in $E(\pi)$ connect operations to be processed by the same machine.

Arcs from set R represent the processing order of operations in jobs, whereas arcs from set $E(\pi)$ represent the processing order of operations on machines. The processing order π is feasible if and only if graph $G(\pi)$ does not contain a cycle.

Let $C(x, y)$ and $L(x, y)$ denote the longest (critical) path and length of this path, respectively, from node x to y in $G(\pi)$. It is well-known that makespan $C_{max}(\pi)$ for π is equal to length $L(s, c)$ of critical path $C(s, c)$ in $G(\pi)$. Now, we can rephrase the job-shop problem as that of finding a feasible processing order $\pi \in \Pi$ that minimizes $C_{max}(\pi)$ in the resulting graph.

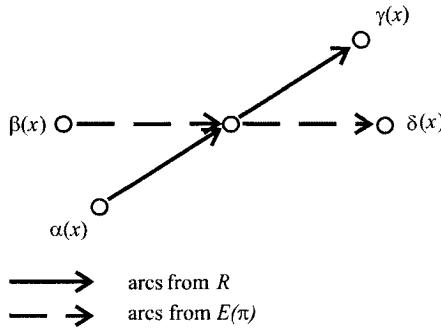


Figure 5.1. Operation predecessors and successors.

We use a notation similar to the paper of Balas and Vazacopoulos (1998). For any operation $x \in O$, we will denote by $\alpha(x)$ and $\gamma(x)$ the job-predecessor and job-successor (if it exists), respectively, of x , i.e. $(\alpha(x), x)$ and $(x, \gamma(x))$ are arcs from R . Further, for the given processing order π , and for any operation $x \in O$, we will denote by $\beta(x)$ and $\delta(x)$ the machine-predecessor and machine-successor (if it exists), respectively, of x , i.e. the operation that precedes x , and

succeeds x , respectively, on the machine processing operation x . In other words, $(\beta(x), x)$ and $(x, \delta(x))$ are arcs from $E(\pi)$, see Figure 5.1.

Denote the critical path in $G(\pi)$ by $C(s, c) = (s, u_1, u_2, \dots, u_w, c)$, where $u_i \in O$, $1 \leq i \leq w$, and w is the number of nodes (except fictitious s and c) in this path. The critical path $C(s, c)$ depends on π , but for simplicity in notation we will not express it explicitly. The critical path is decomposed into subsequences B_1, B_2, \dots, B_r called *blocks* in π on $C(s, c)$ (Grabowski, 1979; Grabowski, Nowicki, and Smutnicki, 1988), where

- 1 $B_k = (u_{f_k}, u_{f_k+1}, \dots, u_{l_k-1}, u_{l_k})$, $1 \leq f_k \leq l_k \leq w$, $k = 1, 2, \dots, r$.
- 2 B_k contains operations processed on the same machine,
 $k = 1, 2, \dots, r$.
- 3 two consecutive blocks contain operations processed on different machines.

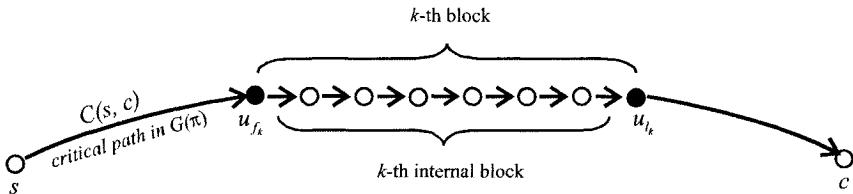


Figure 5.2. Block on critical path.

In other words, the block is a maximal subsequence of $C(s, c)$ and contains successive operations from the critical path processed consecutively on the same machine. In the further considerations, we will be interested only in *non-empty* block, i.e. such that $|B_k| > 1$, or alternatively $f_k < l_k$. Operations u_{f_k} and u_{l_k} in B_k are called the *first* and *last* ones, respectively. The k -th block, exclusive of the first and last operations, is called the k -th *internal block*, see Figure 5.2.

A block has advantageous so-called *elimination properties*, introduced originally in the form of the following theorem (Grabowski, 1979; Grabowski, Nowicki, and Smutnicki, 1988).

THEOREM 5.1 . Let $G(\pi)$ be an acyclic graph with blocks B_k , $k = 1, 2, \dots, r$. If acyclic graph $G(\omega)$ has been obtained from $G(\pi)$ through the modifications of π so that $C_{max}(\omega) < C_{max}(\pi)$, then in $G(\omega)$

- (i) at least one operation $x \in B_k$ precedes job u_{f_k} , for some $k \in \{1, 2, \dots, r\}$,
or
- (ii) at least one operation $x \in B_k$ succeeds job u_{l_k} , for some $k \in \{1, 2, \dots, r\}$.

3. Tabu Search Algorithm (TS)

Currently, TS is one of the most effective methods using local search techniques to find near-optimal solutions to combinatorial optimization problems, see Glover (1989, 1990). The basic idea in our context involves starting from an initial basic job processing order and searching through its neighbourhood, for the processing order with the lowest makespan (in our case, the processing order with the lowest lower bound on the makespan). The search then repeats using the chosen neighbor as a new basic processing order.

The neighbourhood of a basic processing order is generated by the moves. A move changes the location of some operations in the basic processing order. In order to avoid cycling, becoming trapped at a local optimum, or continuing the search in a too narrow region, the mechanisms of a tabu list and a perturbation are utilized. The tabu list records the performed moves, for a chosen span of time, treating them as forbidden for possible future moves, i.e. it determines forbidden processing orders in the currently analyzed neighbourhood. The list content is refreshed each time a new basic processing order is found: the oldest elements are deleted and new ones added. The search stops when a given number of iterations has been reached without improvement of the best current makespan, the algorithm has performed a given total number of iterations, time has run out, the neighbourhood is empty, or a processing order with a satisfying makespan has been found, etc. In practice, the design of the particular components of a search algorithm is considered an art. The construction of the components influences the algorithm performance, speed of convergence, running time, etc.

3.1 Moves and Neighbourhood

In the literature are many types of moves based on interchanges of operations (or jobs) on a machine. The intuition following from Theorem 5.1 suggests that the “insertion” type move is the most proper for the problem considered. In general, the insertion move operates on a sequence of operations by removing an operation from its position in a sequence and inserting it in to another position in the same sequence. More precisely, let $v = (x, y)$ be a pair of operations on a machine, $x, y \in O$, $x \neq y$. With respect to graph $G(\pi)$, the pair $v = (x, y)$ defines a move that consists in removing operation x from its original position and inserting it in the position immediately after (or before) operation y if operation y succeeds (or precedes) operation x in $G(\pi)$. This move v generates a new graph $G(\pi_v)$ from $G(\pi)$. All graphs $G(\pi_v)$, which can be obtained by performing moves from a given move set U , create the neighbourhood $N(U, \pi) = \{G(\pi_v) \mid v \in U\}$ of graph $G(\pi)$.

The proper definition of the move and selection of U , i.e. the neighbourhood $N(U, \pi)$, is very important in constructing an effective algorithm. The set U

should be neither too “big” nor too “small”. The large set requires a great computational effort for the search of $N(U, \pi)$ at a given iteration of the algorithm, whereas the small one needs a large number of iterations for finding a “good” solution.

For the job-shop problem there are several definitions of moves based on the interchanges of adjacent and non-adjacent pairs of operations on a machine.

The interchange moves of the adjacent pairs have been used earlier by Balas (1969), while the non-adjacent ones by Grabowski (1979), Grabowski and Janiak, (1987), and Grabowski, Nowicki and Smutnicki (1988). The latter moves were widely employed for the flow-shop problem by Grabowski (1980, 1982), and Grabowski, Skubalska and Smutnicki (1983), and for the one-machine scheduling by Carlier (1982), Grabowski, Nowicki and Zdrzalka (1986), Adams, Balas and Zawack (1988), and Zdrzalka and Grabowski (1989). All these moves were applied in branch-and-bound procedures.

Recently, in the heuristics for the job-shop problem, the adjacent moves have been employed by Matsuo, Suh and Sullivan (1988), Laarhoven, Aarts and Lenstra (1992), and Nowicki and Smutnicki (1996b), while the non-adjacent ones by DellAmico and Trubian (1993), Balas and Vazacopoulos (1998), and Pezzella and Merelli (2000). Besides, the latter moves were used by Nowicki and Smutnicki (1996a), Smutnicki (1998), and Grabowski and Pempera (2001) in the tabu search algorithms for the flow-shop problem.

The second component of the local search algorithms is a selection (construction) of “effective” neighbourhood $N(U, \pi)$. Amongst many types of neighbourhoods considered (and connected with the chosen definition of the move), two appear to be very interesting.

The first is that proposed by Nowicki and Smutnicki (1996b). In point of the computational results, it seems that their neighbourhood used in the tabu search procedure with built-in block properties (and based on interchanges of some adjacent pairs only) is “optimal”. However, we believe this neighbourhood is “too small”, that is, their TS needs too many iterations. Despite our criticism, the computational results obtained by Nowicki and Smutnicki (1996b) are excellent. Their spectacular success encourages further explorations in that area.

The second neighbourhood (based on interchanges of non-adjacent pairs), presented by Balas and Vazacopoulos (1998), is employed in their local search (tree search) algorithm, denoted GLS. This algorithm is based on the branch-and-bound procedure with an enumeration tree whose size is bounded in a guided manner, so that GLS can be treated as an approximation algorithm. It is clear that the largest size of the neighbourhood is at the root of the tree, and while searching, the size decreases with increasing levels of the tree. Additionally, GLS consists of several such procedures (each of them starting with various initial solutions). As a consequence, it is difficult to compare the neighbour-

hood size of GLS with those given in the literature. However, with regard to the neighbourhood at the root of the tree, GLS investigates a considerably larger neighbourhood than the heuristics based on interchanges of adjacent pairs of operations. Computational results obtained by GLS confirm an advantage over other heuristics for the job-shop problem. In our algorithm TS, the neighbourhood is larger than that at the root of the tree in GLS, and is based on the block approach. Besides, in order to reduce calculations for the search of neighbourhood, we propose to use a lower bound on the makespan instead of calculating the makespan explicitly, as a basis for choosing the best move.

For any block B_k in $G(\pi)$ (acyclic), let us consider the set of moves $W_k(\pi)$ which can be performed inside this block, i.e. on operations $u_{f_k+1}, \dots, u_{l_k-1}$, $k = 1, 2, \dots, r$. Precisely, each $W_k(\pi)$ is defined by the formula

$$W_k(\pi) = \{(x, y) \mid x, y \in \{u_{f_k+1}, \dots, u_{l_k-1}\}, x \neq y\}.$$

All these moves create the set $W(\pi) = \bigcup_{k=1}^r W_k(\pi)$.

Immediately from Theorem 5.1 we obtain the following Corollary which provides the basis for elimination (Grabowski, 1979; Grabowski, Nowicki, and Smutnicki, 1988).

COROLLARY 1 . *If acyclic graph $G(\pi_v)$ has been generated from acyclic graph $G(\pi)$ by a move $v \in W(\pi)$, then $C_{max}(\pi_v) \geq C_{max}(\pi)$.*

This Corollary states that the moves from the set $W(\pi)$ defined above are not interesting, taking into account the possibility of an immediate improvement of the makespan after making a move.

Next, we will give a detailed description of the moves and neighbourhood structure considered in this paper. Let us consider the sequence of operations on critical path $C(s, c)$ in $G(\pi)$ and blocks B_1, B_2, \dots, B_r determined for $C(s, c)$. For each fixed operation x belonging to the critical path $C(s, c)$, we consider at most one move to the right and at most one to the left. Moves are associated with blocks. Let us take the block $B_k = \{u_{f_k}, u_{f_k+1}, \dots, u_{l_k-1}, u_{l_k}\}$, $k = 1, 2, \dots, r$. Then, we define the sets of *candidates* (Grabowski, 1979; Grabowski, Nowicki, and Smutnicki, 1988).

$$\begin{aligned} E_{ka} &= \{u_{f_k}, u_{f_k+1}, \dots, u_{l_k-1}\} = B_k - \{u_{l_k}\}, \\ E_{kb} &= \{u_{f_k+1}, \dots, u_{l_k-1}, u_{l_k}\} = B_k - \{u_{f_k}\}. \end{aligned}$$

Each set E_{ka} (or E_{kb}) contains the operations in the k -th block of $G(\pi)$ that are candidates for being moved to a position *after* (or *before*) all other operations in the k -th block. More precisely, we move operation x , $x \in E_{ka}$, to the right in to the position immediately after operation u_{l_k} , and this move takes the form $v = (x, u_{l_k})$, so Corollary 1 can not be applied to this move, i.e. $v \notin W_k$. By symmetry, operation x , $x \in E_{kb}$, is moved to the left in the position immediately before operation u_{f_k} , and this move takes the form $v = (x, u_{f_k})$, so $v \notin W_k$. Note that after performing a move $v = (x, u_{l_k})$, $x \in E_{ka}$ (or $v = (x, u_{f_k})$, $x \in E_{kb}$), operation x , in $G(\pi_v)$, is to be processed as the last (or first) operation of the k -th block of $G(\pi)$. It is easy to observe that in order to obtain the graph $G(\pi_v)$ by performing a move $v = (x, u_{l_k})$, $x \in E_{ka}$ (or $v = (x, u_{f_k})$, $x \in E_{kb}$), we should remove the arcs $(\beta(x), x)$, $(x, \delta(x))$ and $(u_{l_k}, \delta(u_{l_k}))$ from $G(\pi)$ and add to $G(\pi)$ the arcs $(\beta(x), \delta(x))$, (u_{l_k}, x) and $(x, \delta(u_{l_k}))$ (or remove the arcs $(\beta(x), x)$, $(x, \delta(x))$ and $(\delta(u_{f_k}), u_{f_k})$, and add the arcs $(\beta(x), \delta(x))$, (x, u_{f_k}) and $(\beta(u_{f_k}), x)$). For illustration, performing the move $v = (x, u_{l_k})$ is shown in Figure 5.3.

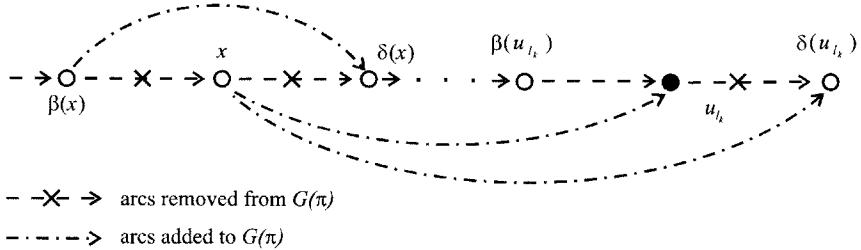


Figure 5.3. Move performance.

According to the description given, for any block B_k in $G(\pi)$, $k = 1, 2, \dots, r$, we define the following set of moves to the right

$$ZR_k(\pi) = \{(x, u_{l_k}) | x \in E_{ka}\}$$

and the set of moves to the left

$$ZL_k(\pi) = \{(x, u_{f_k}) | x \in E_{kb}\}.$$

Set $ZR_k(\pi)$ contains all moves of operations of E_{ka} to the right after the last operation u_{l_k} of the k -th block. Similarly, set $ZL_k(\pi)$ contains all moves of operations of E_{kb} to the left before the first operation u_{f_k} of the k -th block. Of

course, Corollary 1 does not hold for moves from the sets $ZR_k(\pi)$ and $ZL_k(\pi)$. For illustration, the moves performed to the right and left are shown in Figure 5.4.

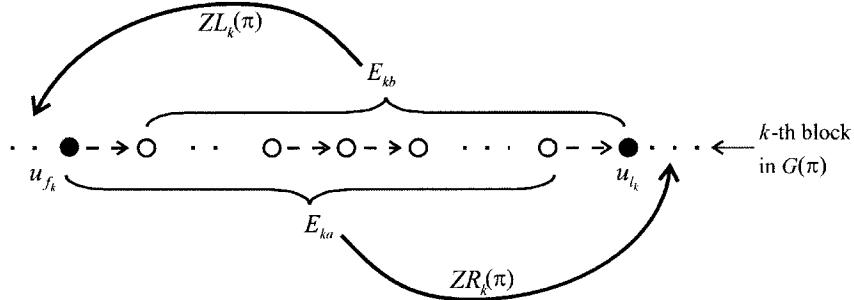


Figure 5.4. Operation movements.

Note that if $|B_k| = 2$, for some $k \in \{1, 2, \dots, r\}$, then $E_{ka} = \{u_{f_k}\}$, $E_{kb} = \{u_{l_k}\}$, and $ZR_k(\pi) = ZL_k(\pi)$, and one of these sets can be eliminated. If we assume that $E_{ka} = \{u_{l_{k-1}}\}$ in $ZR_k(\pi)$ and $E_{kb} = \{u_{f_{k+1}}\}$ in $ZL_k(\pi)$ then $ZR_k(\pi) \cup ZL_k(\pi)$ is similar to that presented by Nowicki and Smutnicki (1996b), denoted as $V_k(\pi)$.

As a consequence of the above considerations, in TS, we should take the set of moves

$$M(\pi) = \bigcup_{k=1}^r (ZR_k(\pi) \cup ZL_k(\pi))$$

and the resulting neighbourhood $N(M(\pi), \pi)$.

A set of moves similar to $M(\pi)$ has been proposed by Grabowski (1980, 1982) and Grabowski, Skubalska and Smutnicki (1983) for the flow-shop problem. However, for the job-shop problem, the neighbourhood $N(M(\pi), \pi)$ contains processing orders which can be infeasible. It should be noticed that if a move $v = (x, u_{l_k}) \in ZR_k(\pi)$, (or $v = (x, u_{f_k}) \in ZL_k(\pi)$) contains an adjacent pair of operations, i.e. $x = u_{l_{k-1}} \in E_{ka}$, (or $x = u_{f_{k+1}} \in E_{kb}$), then the resulting graph $G(\pi_v)$ is acyclic (Balas, 1969; Laarhoven, Aarts, and Lenstra, 1992).

In sequel, we consider conditions under which performing a move $v = (x, u_{l_k}) \in ZR_k(\pi)$, $x \neq u_{l_{k-1}}$, (or $v = (x, u_{f_k}) \in ZL_k(\pi)$, $x \neq u_{f_{k+1}}$) in an acyclic $G(\pi)$, generates the acyclic graph $G(\pi_v)$.

THEOREM 5.2 *For each acyclic $G(\pi)$, if $G(\pi_v)$ has been generated by a move $v = (x, u_{l_k}) \in ZR_k(\pi)$, $x \neq u_{l_{k-1}}$, $k = 1, 2, \dots, r$, and if in $G(\pi)$*

$$L(u_{l_k}, c) + \min(p_{\alpha(u_{l_k})}, p_{u_{l_{k-1}}}) + p_{\gamma(x)} > L(\gamma(x), c), \quad (5.1)$$

and $\gamma(x) \neq \alpha(u_{l_k})$, then $G(\pi_v)$ is acyclic.

Proof (by contradiction).

For simplicity, the index k will be dropped. For a move $v = (x, u_l)$, $x \in E_a$, $x \neq u_{l-1}$, we suppose that there is created a cycle C in $G(\pi_v)$. It is obvious that C contains some arcs that are added to graph $G(\pi)$ (see Figure 5.5).

If $(\beta(x), \delta(x)) \in C$, then $G(\pi)$ contains a path from $\delta(x)$ to $\beta(x)$, which contradicts the assumption that $G(\pi)$ is acyclic. Therefore, C can contain $(x, \delta(u_l))$ or (u_l, x) . If C contains both these arcs, then there is a path in $G(\pi)$ from $\delta(u_l)$ to u_l , contrary to the assumption that $G(\pi)$ is acyclic. Hence, C contains either $(x, \delta(u_l))$, or (u_l, x) . If $(x, \delta(u_l)) \in C$, then there is a path in $G(\pi)$ from $\delta(u_l)$ to x , again contrary to the assumption. Finally, if $(u_l, x) \in C$, then C contains

- a) a path $d_1(x, u_l) = ((x, \gamma(x), d(\gamma(x), \alpha(u_l)), (\alpha(u_l), u_l))$, or
- b) a path $d_2(x, u_l) = ((x, \gamma(x), d(\gamma(x), u_{l-1}), (u_{l-1}, u_l))$,

(a) In this case if C contains path $d_1(x, u_l)$, then this path is in $G(\pi)$ and, since $\gamma(x) \neq \alpha(u_l)$, we obtain

$$L(\gamma(x), c) \geq L(u_l, c) + p_{\alpha(u_l)} + p_{\gamma(x)}. \quad (5.1a)$$

(b) But if C contains path $d_2(x, u_l)$, then this path is in $G(\pi)$, and now we obtain

$$L(\gamma(x), c) \geq L(u_l, c) + p_{u_{l-1}} + p_{\gamma(x)}. \quad (5.1b)$$

Together (5.1a) and (5.1b) imply

$$L(\gamma(x), c) \geq L(u_l, c) + \min(p_{\alpha(u_l)}, p_{u_{l-1}}) + p_{\gamma(x)},$$

which contradicts the assumption 5.1.

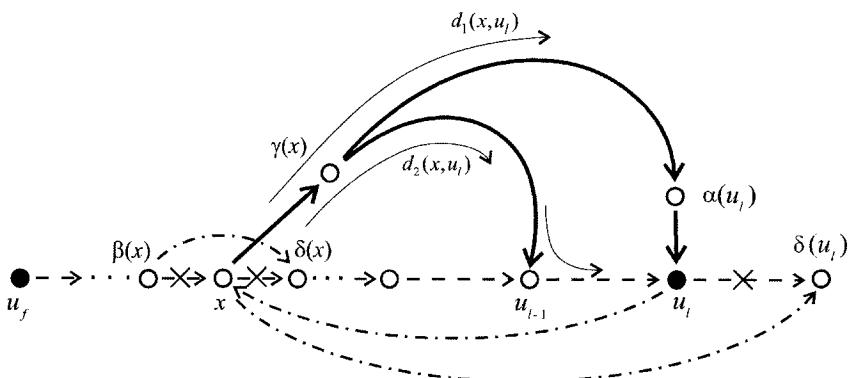


Figure 5.5. Paths $d_1(x, u_l)$ and $d_2(x, u_l)$ in $G(\pi)$.

The considerations in the proof of Theorem 5.2 suggest the following property.

PROPERTY 1 *For each acyclic $G(\pi)$, if $G(\pi_v)$ has been generated by a move $v = (x, u_{l_k}) \in ZR_k(\pi)$, $k = 1, 2, \dots, r$ and if x has no job-successor $\gamma(x)$, then $G(\pi_v)$ is acyclic.*

By symmetry, we have

THEOREM 5.3 *For each acyclic $G(\pi)$, if $G(\pi_v)$ has been generated by a move $v = (x, u_{f_k}) \in ZL_k(\pi)$, $x \neq u_{f_k+1}$, $k = 1, 2, \dots, r$, and if in $G(\pi)$*

$$L(s, u_{f_k}) + \min(p_{\gamma(u_{f_k})}, p_{u_{f_k}+1}) + p_{\alpha(x)} > L(s, \alpha(x)), \quad (5.2)$$

and $\alpha(x) \neq \gamma(u_{f_k})$, then $G(\pi_v)$ is acyclic.

The proof of Theorem 5.3 can be obtained by using similar considerations to Theorem 5.2, but with the set of moves $ZL_k(\pi)$.

By analogy, we have

PROPERTY 2 *For each acyclic $G(\pi)$, if $G(\pi_v)$ has been generated by a move $v = (x, u_{f_k}) \in ZL_k(\pi)$, $k = 1, 2, \dots, r$, and if x has no job-predecessor $\alpha(x)$, then $G(\pi_v)$ is acyclic.*

Note that the conditions 5.1 and 5.2 are both less restrictive than those given by Balas and Vazacopoulos (1998) for procedure GLS, so that our neighbourhood is larger than that proposed by Balas and Vazacopoulos (1998), but it is slightly smaller than that of DellAmico and Trubian (1993).

Let

$$\begin{aligned} ZR_k^*(\pi) &= \{v \in ZR_k(\pi) | v \text{ satisfies 5.1 and } \gamma(x) \neq \alpha(u_{l_k}), \text{ or } x = u_{l_k-1}\}, \\ ZL_k^*(\pi) &= \{v \in ZL_k(\pi) | v \text{ satisfies 5.2 and } \alpha(x) \neq \gamma(u_{f_k}), \text{ or } x = u_{f_k+1}\}, \end{aligned}$$

be the sets of the moves from $ZR_k(\pi)$ and $ZL_k(\pi)$, the performance of which generates acyclic $G(\pi_v)$ from acyclic $G(\pi)$. Finally, in our TS, we will employ the set of moves

$$M^*(\pi) = \bigcup_{k=1}^r (ZR_k^*(\pi) \cup ZL_k^*(\pi)),$$

which creates the neighbourhood $N(M^*(\pi), \pi)$.

As a consequence of the above considerations, let

$$\begin{aligned} E_{ka}^* &= \{x \in E_{ka} | (x, u_{l_k}) \in ZR_k^*(\pi)\}, \\ E_{kb}^* &= \{x \in E_{kb} | (x, u_{f_k}) \in ZL_k^*(\pi)\}, \end{aligned}$$

be the sets of operations whose movement generates acyclic $G(\pi_v)$ from acyclic $G(\pi)$.

In order to decrease the total computational effort for the search, we propose calculation of a lower bound on the makespans instead of computing the makespans explicitly for use in selecting the best solution, though doing so can increase the number of iterations in TS. The makespan resulting from performing a move can be calculated by using the standard Bellman's algorithm in $O(o)$ time, however, doing this for every solution becomes too expensive, so that we propose a less costly lower bound. In fact, this lower bound is used for evaluating and selecting the "best" move.

Next, we present a method to indicate a move to be performed, i.e. an operation which should be moved after the last operation u_k (or before the first operation u_{f_k}) of the k -th block. According to the search strategy in TS, we want to choose a move v which will generate graph $G(\pi_v)$ with the smallest possible makespan $L_v(s, c)$. To evaluate all moves from the sets $ZR_k^*(\pi)$ and $ZL_k^*(\pi)$, (i.e. all operations from E_{ka}^* and E_{kb}^*) we introduce the formula

$$\Delta_{ka}(x) = \max(L_1^a, L_2^a, L_3^a, L_4^a, L_5^a), \quad x \in E_{ka}^*,$$

where:

$$\begin{aligned} L_1^a &= -p_x, \\ L_2^a &= L(\gamma(x), c) - L(u_{l_k}, c) + p_{u_{l_k}}, \\ L_3^a &= L(s, \alpha(\delta(x))) - L(s, x), \\ L_4^a &= L_2^a + L_3^a + p_x, \\ L_5^a &= L(\gamma(\beta(x)), c) - L(x, c), \quad x \neq u_{f_k}. \end{aligned}$$

And

$$\Delta_{kb}(x) = \max(L_1^b, L_2^b, L_3^b, L_4^b, L_5^b), \quad x \in E_{kb}^*,$$

where:

$$\begin{aligned} L_1^b &= -p_x, \\ L_2^b &= L(s, \alpha(x)) - L(s, u_{f_k}) + p_{u_{f_k}}, \\ L_3^b &= L(\gamma(\beta(x)), c) - L(x, c), \\ L_4^b &= L_2^b + L_3^b + p_x, \\ L_5^b &= L(s, \alpha(\delta(x))) - L(s, x), \quad x \neq u_{l_k}. \end{aligned}$$

The complexity of $\Delta_{k\lambda}(x)$, $\lambda \in \{a, b\}$, having the components $L(s, i)$ and $L(i, c)$, $i \in N$, is $O(1)$. Note that these components are obtained during the calculation of the makespan $L(s, c)$ in $G(\pi)$. Here, if there does not exist $\gamma(x)$ (or $\alpha(x)$) for some x , then $\gamma(x) = c$ and $L(\gamma(x), c) = 0$ (or $\alpha(x) = s$ and $L(s, \alpha(x)) = 0$). Further, if there does not exist $\alpha(\delta(x))$ (or $\gamma(\beta(x))$) for some x , then $\alpha(\delta(x)) = s$ and $L(s, \alpha(\delta(x))) = 0$ (or $\gamma(\beta(x)) = c$ and $L(\gamma(\beta(x)), c) = 0$). Note that if $x = u_{f_k}$ (or $x = u_{l_k}$), then L_5^a (or L_5^b)

is not used during the calculation of $\Delta_{ka}(x)$ (or $\Delta_{kb}(x)$). The usefulness of these values $\Delta_{\lambda k}(x)$, $\lambda \in \{a, b\}$, for the choice of a move is illustrated by the following Theorems.

THEOREM 5.4 *For each acyclic $G(\pi)$, if $G(\pi_v)$ has been generated by a move $v = (x, u_l) \in ZR_k^*(\pi)$, $k = 1, 2, \dots, r$, then*

$$L_v(s, c) \geq L(s, c) + \Delta_{ka}(x)$$

where $L_v(s, c)$ is the length of a critical path in $G(\pi_v)$.

Proof.

For simplicity, the index k will be dropped, then we have

$$\begin{aligned} L(s, c) + \Delta_a(x) &= L(s, c) + \max(L_1^a, L_2^a, L_3^a, L_4^a, L_5^a) \\ &= L(s, c) + \max(-p_x, L(\gamma(x), c) - L(u_l, c) + p_{u_l}, L(s, \alpha(\delta(x))) \\ &\quad - L(s, x), L(\gamma(x), c) - L(u_l, c) + p_{u_l} + L(s, \alpha(\delta(x))) - L(s, x) \\ &\quad + p_x, L(\gamma(\beta(x)), c) - L(x, c)). \end{aligned}$$

Since $G(\pi)$ is acyclic, then there exists a critical path. And for each node $i \in N$ which belongs to the critical path, we have

$$C(s, c) = (C(s, i), C(i, c)),$$

and

$$L(s, c) = L(s, i) + L(i, c) - p_i. \quad (5.3)$$

Further, since the nodes $\beta(x)$, x , $\delta(x)$ and u_l belong to the critical path $C(s, c)$, then, using 5.3, we get

$$\begin{aligned} L(s, c) + \Delta_a(x) &= \max(L(s, c) - p_x, L(s, c) + L(\gamma(x), c) \\ &\quad - L(u_l, c) + p_{u_l}, L(s, c) + L(s, \alpha(\delta(x))) - L(s, x), L(s, c) + L(\gamma(x), c) \\ &\quad - L(u_l, c) + p_{u_l} + L(s, \alpha(\delta(x))) - L(s, x) + p_x, L(s, c) - L(x, c) \\ &\quad + L(\gamma(\beta(x)), c)) = \max(L(s, x) + L(x, c) - 2p_x, L(s, u_l) + L(u_l, c) \\ &\quad + p_{u_l} + L(\gamma(x), c) - L(u_l, c) + p_{u_l}, L(s, x) + L(\delta(x), c) \\ &\quad + L(s, \alpha(\delta(x))) - L(s, x), L(s, x) + L(\delta(x), u_l) + L(u_l, c) - p_{u_l} \\ &\quad + L(\gamma(x), c) - L(u_l, c) + p_{u_l} + L(s, \alpha(\delta(x))) - L(s, x) \\ &\quad + p_x, L(s, \beta(x)) + L(x, c) + L(\gamma(\beta(x)), c) - L(x, c)) \\ &= \max(L(s, x) + L(x, c) - 2p_x, L(s, u_l) + L(\gamma(x), c), L(\delta(x), c) \\ &\quad + L(s, \alpha(\delta(x))), L(s, \alpha(\delta(x))) + L(\delta(x), u_l) + p_x \\ &\quad + L(\gamma(x), c), L(s, \beta(x)) + L(\gamma(\beta(x)), c)). \end{aligned} \quad (5.4)$$

Now, let us consider certain paths $d_1(s, c)$, $d_2(s, c)$, $d_3(s, c)$, $d_4(s, c)$ and $d_5(s, c)$ from s to c in $G(\pi_v)$ generated by the move $v = (x, u_l) \in ZR_k^*(\pi)$ (see Figure 5.6),

$$\begin{aligned} d_1(s, c) &= (C(s, \beta(x)), (\beta(x), \delta(x)), C(\delta(x), c)), \\ d_2(s, c) &= (C(s, \beta(x)), (\beta(x), \delta(x)), C(\delta(x), u_l), (u_l, x), (x, \gamma(x)), \\ &\quad C(\gamma(x), c)), \\ d_3(s, c) &= (C(s, \alpha(\delta(x))), (\alpha(\delta(x)), \delta(x)), C(\delta(x), c)), \\ d_4(s, c) &= (C(s, \alpha(\delta(x))), (\alpha(\delta(x)), \delta(x)), C(\delta(x), u_l), (u_l, x), \\ &\quad (x, \gamma(x)), C(\gamma(x), c)), \\ d_5(s, c) &= (C(s, \beta(x)), (\beta(x), \gamma(\beta(x))), C(\gamma(\beta(x)), c)). \end{aligned}$$

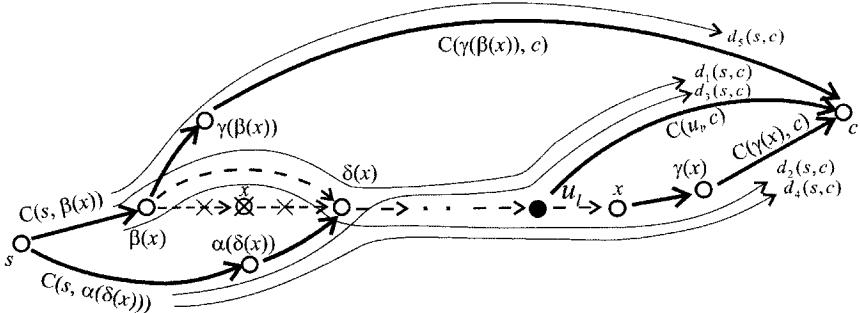


Figure 5.6. Paths $d_1(s, c)$, $d_2(s, c)$, $d_3(s, c)$, $d_4(s, c)$ and $d_5(s, c)$ in $G(\pi_v)$.

The lengths of these paths are

$$\begin{aligned} l_1(s, c) &= L(s, \beta(x)) + L(\delta(x), c) = L(s, x) - p_x + L(x, c) - p_x \\ &= L(s, x) + L(x, c) - 2p_x, \\ l_2(s, c) &= L(s, \beta(x)) + L(\delta(x), u_l) + p_x + L(\gamma(x), c) = L(s, u_l) \\ &\quad + L(\gamma(x), c), \\ l_3(s, c) &= L(s, \alpha(\delta(x))) + L(\delta(x), c), \\ l_4(s, c) &= L(s, \alpha(\delta(x))) + L(\delta(x), u_l) + p_x + L(\gamma(x), c), \\ l_5(s, c) &= L(s, \beta(x)) + L(\gamma(\beta(x)), c). \end{aligned}$$

Since $G(\pi_v)$ is acyclic, then there exists a critical path $C_v(s, c)$, the length of which can not be shorter than the length of any path from s to c in $G(\pi_v)$. Therefore, we have

$$\begin{aligned} L_v(s, c) &\geq l_1(s, c), \quad L_v(s, c) \geq l_2(s, c), \quad L_v(s, c) \geq l_3(s, c), \\ L_v(s, c) &\geq l_4(s, c), \quad L_v(s, c) \geq l_5(s, c). \end{aligned}$$

Hence, using 5.4, we get

$$\begin{aligned}
 L_v(s, c) &\geq \max(l_1(s, c), l_2(s, c), l_3(s, c), l_4(s, c), l_5(s, c)) \\
 &= \max(L(s, x) + L(x, c) - 2p_x, L(s, u_l) \\
 &\quad + L(\gamma(x), c), L(\delta(x), c) + L(s, \alpha(\delta(x))), L(s, \alpha(\delta(x)))) \\
 &\quad + L(\delta(x), u_l) + p_x + L(\gamma(x), c), L(s, \beta(x)) + L(\gamma(\beta(x)), c)) \\
 &= L(s, c) + \Delta_a(x).
 \end{aligned}$$

An analogous result holds for moves from the set $ZL_k^*(\pi)$.

THEOREM 5.5 *For each acyclic $G(\pi)$, if $G(\pi_v)$ has been generated by a move $v = (x, u_{f_k}) \in ZL_k^*(\pi)$, $k = 1, 2, \dots, r$, then*

$$L_v(s, c) \geq L(s, c) + \Delta_{kb}(x),$$

where $L_v(s, c)$ is the length of the critical path in $G(\pi_v)$.

Proof. Parallels that of Theorem 5.4.

Hence, by moving operation $x \in E_{ka}^*$ (or $x \in E_{kb}^*$) after operation u_{l_k} (or before operation u_{f_k}) in $G(\pi)$, a lower bound on value $L_v(s, c)$ of acyclic graph $G(\pi_v)$ is $L(s, c) + \Delta_{ka}(x)$ (or $L(s, c) + \Delta_{kb}(x)$). Thus, the values $\Delta_{k\lambda}(x)$, $\lambda \in \{a, b\}$, can be used to decide which operation should be moved, i.e. the operation should have the smallest value of $\Delta_{k\lambda}(x)$. The smallest value of $\Delta_{k\lambda}(x)$ corresponds to the “best” move $v = (x, u_{l_k}) \in ZR_k^*(\pi)$, (or $v = (x, u_{f_k}) \in ZL_k^*(\pi)$) if $\Delta_{ka}(x)$ (or $\Delta_{kb}(x)$) reaches this value. From Theorems 5.4 and 5.5 it follows that if $\Delta_{k\lambda}(x) > 0$, then in the resulting graph $G(\pi_v)$ we have $L_v(s, c) > L(s, c)$.

Generally, in our TS, for the given graph $G(\pi)$, we calculate the critical path $C(s, c)$ (if there is more than one critical path, any one of them can be used), and the length of this path $C_{max}(\pi)$ ($= L(s, c)$). We then identify the blocks B_1, B_2, \dots, B_r , create the set of moves $M^*(\pi)$, compute the values $\Delta_{k\lambda}(x)$, $x \in E_{k\lambda}^*$, $\lambda \in \{a, b\}$, $k = 1, 2, \dots, r$, choose the “best” move v (corresponding to the smallest value of $\Delta_{k\lambda}(x)$) from set $M^*(\pi)$ and create the graph $G(\pi_v)$ by removing some arcs from $G(\pi)$ and adding other ones to $G(\pi)$ (see beginning of this section). Next, the search process of TS is repeated for the resulting graph $G(\pi_v)$ until *Maxiter* of iterations is reached. Of course, according to the philosophy of TS, there are some exceptions while choosing the “best” move:

- A. If the chosen move has a status tabu (see next section for details), the move is not allowed.

- B. If MaxretP ($\text{MaxretP} < \text{Maxiter}$) of the consecutive non-improving iterations pass in TS, then, instead of a single (“best”) move, we choose several ones to be performed simultaneously (see section **Perturbations** for details).

Exception (B) gives assistance in addition to the tabu list to avoid being trapped at a local optimum.

3.2 Tabu List and Tabu Status of Move

In our algorithm we use the tabu list defined as a finite list (set) T with dynamic length $\text{Length } T$ containing ordered pairs of operations. The list is initiated by introducing $\text{Length } T$ empty elements. If a move $v = (x, u_{t_k}) \in ZR_k^*(\pi)$, (or move $v = (x, u_{f_k}) \in ZL_k^*(\pi)$) is performed on graph $G(\pi)$ generating graph $G(\pi_v)$, then the pair of operations $(\delta(x), x)$ (or pair $(x, \beta(x))$), representing a precedence constraint, is added to T . Each time before adding a new pair to T , we must delete the oldest one.

With respect to graph $G(\pi)$, a move $(x, u_{t_k}) \in ZR_k^*(\pi)$, (or a move $(x, u_{f_k}) \in ZL_k^*(\pi)$) has the *tabu* status (it is forbidden) if $A(x) \cap B_k \neq \emptyset$ (or $B(x) \cap B_k \neq \emptyset$), where:

$$\begin{aligned} A(x) &= \{y \in O \mid (x, y) \in T\}, \\ B(x) &= \{y \in O \mid (y, x) \in T\}. \end{aligned}$$

Set $A(x)$ (or set $B(x)$) indicates which operations are to be processed *after* (or *before*) operation x with respect to the current content of the tabu list T .

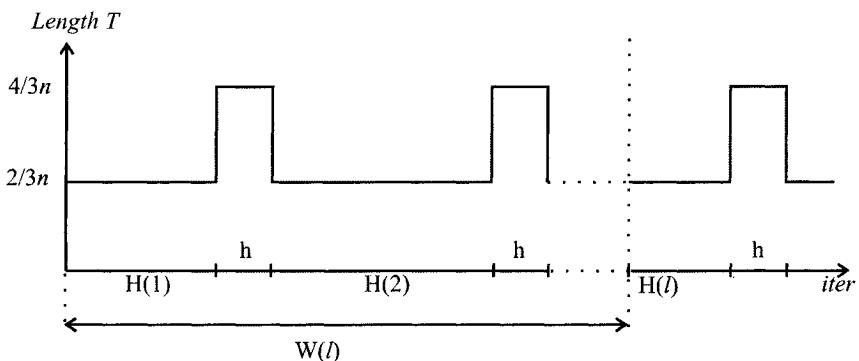


Figure 5.7. Dynamic tabu list.

As mentioned above, our algorithm uses a tabu list with dynamic length. This length is changed, as the current iteration number iter of TS increases. The length change is used as a “pick” intended to carry the search to another

area of the solution space. $LengthT$ is a cyclic function shown in Figure 5.7 and defined by the expression

$$LengthT = \begin{cases} \left\lceil \frac{2}{3}n \right\rceil, & \text{if } W(l) < iter \leq W(l) + H(l), \\ \left\lceil \frac{4}{3}n \right\rceil, & \text{if } W(l) + H(l) < iter \leq W(l) + H(l) + h, \end{cases}$$

where: $l = 1, 2, \dots$ is the number of the cycle, $W(l) = \sum_{s=1}^l H(s-1) + (l-1)*h$, (here $H(0) = 0$), and h is the width of the pick equal to n . Interval $H(l)$ is the parameter which is not constant, but it depends on the structure of graph $G(\pi)$ currently considered. More precisely, let $G(\pi)$ be the graph obtained at the beginning of the interval $H(l)$, i.e. in $W(l) + 1$ iteration (see expression on $LengthT$). Then the next pick is begun when $H(l) = 2 \times |C|$ iterations pass in TS, where $|C|$ is the number of nodes in the critical path of $G(\pi)$. The one exception is for the first cycle when we take $H(1) = 3 \times |C|$.

If $LengthT$ decreases then a suitable number of the oldest elements of tabu list T is deleted and the search process is continued.

3.3 Search Strategy

We employ a specific searching strategy which yields very good computational results. A move $v = (x, u_{l_k}) \in M^*(\pi)$ (or $v = (x, u_{f_k}) \in M^*(\pi)$) is unforbidden (UF), if it does not have the tabu status. For a given graph $G(\pi)$, the neighbourhood is searched in the following manner. First, the sets of unforbidden moves are defined

$$UR_k = \{v \in ZR_k^*(\pi) \mid \text{move } v \text{ is UF}\},$$

$$UL_k = \{v \in ZL_k^*(\pi) \mid \text{move } v \text{ is UF}\}.$$

For the k -th block, the “best” moves $v_{R(k)} \in UR_k$ and $v_{L(k)} \in UL_k$ are chosen (respectively):

$$\begin{aligned} DELTA(v_{R(k)}) &= \min_{v=(x,u_{l_k}) \in UR_k} \Delta_{ka}(x), \quad k = 1, 2, \dots, r, \\ DELTA(v_{L(k)}) &= \min_{v=(x,u_{f_k}) \in UL_k} \Delta_{kb}(x), \quad k = 1, 2, \dots, r. \end{aligned}$$

Next, the following sets of moves are created

$$RB = \{v_{R(k)} \mid k = 1, 2, \dots, r\},$$

$$LB = \{v_{L(k)} \mid k = 1, 2, \dots, r\},$$

and

$$BB = RB \cup LB = \{v_1, v_2, \dots, v_{2r}\}.$$

Note that the move $v_k \in BB$ belongs either to RB or to LB . The move v to be performed is selected amongst those in BB with the lowest value of $DELTA(v)$, i.e. $DELTA(v) = \min_{v_k \in BB} DELTA(v_k)$, and which gives the lowest bound on value $C_{max}(\pi_v)$, that is $C_{max}(\pi) + DELTA(v)$ (see Theorems 5.4 and 5.5). If the move v is selected, then the resulting graph $G(\pi_v)$ is created, and a pair of operations corresponding to the move v is added to the tabu list T (see section **Tabu list and tabu status of move** for details). If set BB is empty, then the oldest element of tabu list T is deleted, and the search is repeated until non-empty set BB is found.

3.4 Perturbations

The main handicap of a local search procedure is its myopic nature: it looks only one single move ahead, and any move can lead to a “bad” solution where the search becomes trapped in a local optimum that may be substantially “worse” than the global optimum, even in the tabu search approach where a tabu list is used. In this paper, we use a certain perturbation technique in addition to the tabu list for overcoming this drawback of traditional local search algorithms.

The generic key idea of a perturbation is to consider a search which allows us several moves to be made simultaneously in a single iteration and carry the search to the more promising areas of solution space.

In our algorithm, the set of promising moves can be found as follows

$$BB^{(-)} = \{v_k \in BB \mid DELTA(v_k) < 0\} = \{v_1, v_2, \dots, v_z\}, \quad z \leq 2r.$$

The intuition following from Theorems 5.4 and 5.5 suggests that each move $v \in BB^{(-)}$ can provide a graph $G(\pi_v)$ that is “better” than $G(\pi)$. Therefore, as a perturbation, we decided to perform simultaneously all moves from $BB^{(-)}$ in $G(\pi)$, obtaining the resulting graph, denoted $G(\bar{\pi})$, where $\bar{\pi} = (v_1, v_2, \dots, v_z)$. While performing simultaneously all moves from $BB^{(-)}$, the different moves of $BB^{(-)}$ operate in different blocks of $G(\pi)$. Therefore, graph $G(\bar{\pi})$ is acyclic (it follows from the proofs of Theorems 5.2 and 5.3).

Note that if $|BB^{(-)}| = 1$, then the perturbation is equivalent to the selection from BB the single (“best”) move to be performed, thus, in this case, it is not treated as a perturbation. Furthermore, if set $BB^{(-)}$ is empty then the perturbation can not be performed. Therefore, in both cases, the search process is continued (according to the description given in section **Search strategy**) until the graph with $|BB^{(-)}| > 1$ is obtained, and then the perturbation can be made.

If a perturbation is performed, then a pair of operations corresponding to the move v with the smallest value of $DELTA(v)$ is added to tabu list T (see section **Tabu list and tabu status of move** for details).

A perturbation is used when at least *MaxretP consecutive non-improving iterations* pass in the algorithm. More precisely, if graph $G(\bar{\pi})$ is obtained after

performing a perturbation, then the next one is made when $MaxretP$ of the iterations will pass in TS. In other words, the perturbation is made periodically, where $MaxretP$ is the number of the iterations between the neighbouring ones.

3.5 Algorithm TSGW

In the algorithm, the asterisk (*) refers to the best values found, the zero superscript (o) refers to initial values, and its lack denotes the current values. The algorithm starts from a given initial graph $G(\pi^o)$ (π^o can be found by any algorithm). The algorithm stops when $Maxiter$ iterations have been performed.

INITIALISATION.

Set $G(\pi) := G(\pi^o)$, $C^* := C_{max}(\pi^o)$, $\pi^* := \pi^o$, $T := \emptyset$, $iter := 0$, $retp := 0$.

SEARCHING.

Set $iter := iter + 1$, modify (if it is appropriate) $LengthT$ of the tabu list according to the method described earlier, and for graph $G(\pi)$ create a set of representatives BB .

SELECTION.

If $BB = \emptyset$, then remove the oldest element of the tabu list and go to SEARCHING.

Find the “best” move $v \in BB$, i.e.

$$\text{DELT}A(v) = \min_{v_k \in BB} \text{DELT}A(v_k),$$

create the graph $G(\pi_v)$, calculate $C_{max}(\pi_v)$, and modify the tabu list according to the method described earlier. If $C_{max}(\pi_v) < C^*$, then save the best values $C^* := C_{max}(\pi_v)$, and $\pi^* := \pi_v$. If $C_{max}(\pi_v) \geq C_{max}(\pi)$, then set $retp := retp + 1$, otherwise set $retp := 0$.

Next set $G(\pi) := G(\pi_v)$.

STOP CRITERIA

If $iter \geq Maxiter$ then STOP.

If $retp < MaxretP$ then go to SEARCHING.

PERTURBATION

For graph $G(\pi)$ create the sets BB and $BB^{(-)}$. If $BB = \emptyset$, then remove the oldest element of the tabu list and go to SEARCHING. Perform the perturbation according to the method described earlier generating graph $G(\pi_{\bar{v}})$, and calculate $C_{max}(\pi_{\bar{v}})$. If $C_{max}(\pi_{\bar{v}}) < C^*$, then save the best

values $C^* := C_{max}(\pi_{\bar{v}})$, $\pi^* := \pi_{\bar{v}}$ and set $retp := 0$. If $|BB^{(-)}| \leq 1$ and $C_{max}(\pi_{\bar{v}}) \geq C_{max}(\pi)$, then set $retp := retp + 1$. If $|BB^{(-)}| \leq 1$ and $C_{max}(\pi_{\bar{v}}) < C_{max}(\pi)$, then set $retp := 0$. If $|BB^{(-)}| > 1$, then set $retp := 0$. Modify the tabu list according to the method described earlier. Next set $G(\pi) := G(\pi_{\bar{v}})$, and go to SEARCHING.

Algorithm TSGW has one tuning parameter $MaxretP$ which is to be chosen experimentally.

4. Computational Results

Algorithm TSGW was coded in C++, run on a personal computer Pentium 333 MHz, and tested on benchmark problems taken from the literature. The results obtained by our algorithm were then compared with results from the literature.

So far, the best approximation algorithms for the job-shop problem with the makespan criterion were proposed in papers by Matsuo, Suh and Sullivan (1988), Laarhoven, Aarts and Lenstra (1992), DellAmico and Trubian (1993), Nowicki and Smutnicki (1996b), Balas and Vazacopoulos (1998), and Pezzella and Merelli (2000). Pezzella and Merelli reported that their algorithm, denoted as TSSB, provides better results than the ones proposed by other authors. Therefore we compare our algorithm TSGW with TSSB, which is also based on the tabu search approach.

Algorithm TSGW, similarly as TSSB, was tested on 133 commonly used problem instances of various sizes and difficulty levels taken from the OR-Library.

(a) Five instances denoted as ORB1–ORB5 with $n \times m = 10 \times 10$ due to Applegate and Cook (1991), three instances FT6, FT10, FT20 with $n \times m = 6 \times 6, 10 \times 10, 5 \times 20$ due to Fisher and Thompson (1963), and five instances ABZ5–ABZ9 with $n \times m = 10 \times 10, 20 \times 15$ due to Adams, Balas and Zawack (1988).

(b) Forty instances of eight different sizes LA01–LA40 with $n \times m = 10 \times 5, 15 \times 5, 20 \times 5, 10 \times 10, 15 \times 10, 20 \times 10, 30 \times 10, 15 \times 15$ due to Lawrence (1984). The optimal solution of the instance LA29 is thus far unknown.

(c) Eighty instances of eight different sizes TA1–TA80 with $n \times m = 15 \times 15, 20 \times 15, 20 \times 20, 30 \times 15, 30 \times 20, 50 \times 15, 50 \times 20, 100 \times 20$ due to Taillard (1993). For this class, the optimal solution is known only 32 out of 80 instances.

The effectiveness of our algorithm was analysed in both terms of CPU time and solution quality. There are some complications involving the speed of computers used in the tests. Algorithm TSGW was run on Pentium 333 MHz, whereas TSSB was run on Pentium 133 MHz. Regarding the speed of the performance, it is becoming very difficult to compare the CPU times of algorithms tested on different computers. An attempt is made to compare the CPU times

for different algorithms using conversion factors for different machines given in a report by Dongarra (2004). Although, the benchmark results reported in Dongarra tests can be used to give a rough estimate on the relative performance of different computers, these results refer to floating-point operations and therefore may not be representative when computations are essentially with integers, as in the case of our algorithms. Besides, the architecture, configurations, cache, main memory and compilers also affect the CPU times. Therefore, in order to avoid discussion about the conversion factors and speed of computers used in the tests, we enclosed for each compared algorithm the original name of computer on which it has been tested, as well as the original running time.

Algorithm TSGW needs an initial solution, which can be found by any heuristic method. In our tests, we use the procedure INSA which is based on an insertion technique, see Nowicki and Smutnicki (1996b). The computational complexity of this heuristic is $O(n^3m^2)$.

At the initial stage, TSGW was run several times, for small-size instances in order to find the proper value of tuning parameter $MaxretP$. This was chosen experimentally as a result of the compromise between the running time and solution quality and we set $MaxretP = 3$.

For each test instance, we collected the following values:

C^A – the makespan found by the algorithm $A \in \{\text{TSGW}, \text{TSSB}\}$.

$Time$ – CPU in seconds.

Then two measures of the algorithms quality were calculated

$PRD(A) = 100(C^A - LB)/LB$ – the value (average) of the percentage relative difference between makespan C^A and the best known lower bound LB (or the optimum value OPT , if it is known).

$CPU(A)$ – the computer time (average) of algorithm A .

For TSSB, there are some problems concerning the interpretation of the results in CPU times for the instances of class (c). In the paper of Pezzella and Merelli (2000), it is reported that for each instance, TSSB performs $Maxiter$ iterations equal to $100n$. So that, the average CPU for the instances with size $n \times m = 20 \times 20$ should be shorter than for the ones with $n \times m = 100 \times 20$, whereas in Table 6 of the paper we have found that for the former instances, the CPU is, in approximation, 150 times longer than for the latter ones. Similar problems are in Table 3 of the paper for the instances of class (b).

Therefore, we conclude that for the instances of classes (b) and (c), $Maxiter$ is not equal to $100n$, but it is different for different instances. Instead, the analysis of the results in Table 5.1 for class (a) suggests that these inconveniences are avoided. Hence, we have assumed that the CPU times of TSSB obtained for both classes (b) and (c) are those for which the C_{max} values (or PRD values)

presented in the paper of Pezzella and Merelli (2000) are reached. And, since these values are reported for each instance, it is possible to detect *Maxiter* and/or CPU time to be correspondent to the C_{max} value produced by TSSB, for an individual instance.

As a consequence of the above, while testing our algorithm, for each instance of classes (b) and (c), we detect the CPU time at which TSGW has reached the C_{max} value not greater than that obtained by TSSB. Then it was possible to compare the CPU times of the algorithms.

In Table 5.1, we present the results obtained for the test problems of class (a) ORB1–ORB5, FT6, FT10, FT20, and ABZ5–ABZ9. For these instances, TSGW was tested for *Maxiter* equal to $300n$.

Table 5.1. Detailed results for the problem instances of class (a)

Problem	$n \times m$	OPT or (LB-UB)	TSGW			TSSB			
			<i>Maxiter</i> = $300 * n$			CPU to opt (or to best)	<i>Maxiter</i> = $100 * n$		
			C_{max}	PRD	CPU		C_{max}	PRD	CPU
ORB1	10×10	1059	1059	0.00	0.9	0.6	1064	0.47	82
ORB2	10×10	888	888	0.00	0.9	0.6	890	0.23	75
ORB3	10×10	1005	1005	0.00	1.1	0.7	1013	0.80	87
ORB4	10×10	1005	1005	0.00	0.8	0.6	1013	0.80	75
ORB5	10×10	887	887	0.00	0.9	0.2	887	0.00	81
FT6	6×6	55	55	0.00	0.1	0.0	55	0.00	-
FT10	10×10	930	930	0.00	1.2	0.2	930	0.00	80
FT20	20×5	1165	1165	0.00	2.3	0.7	1165	0.00	115
ABZ5	10×10	1234	1236	0.16	1.1	(0.2)	1234	0.00	75
ABZ6	10×10	943	943	0.00	1.0	0.2	943	0.00	80
ABZ7	20×15	656	656	0.00	14.8	3.8	666	1.52	200
ABZ8	20×15	(647-669)	671	3.71	14.6	(5.7)	678	5.12	205
ABZ9	20×15	(661-679)	682	3.18	14.9	(3.9)	693	4.84	195
all				0.54				1.06	

CPU represents the CPU time:

TSGW on Pentium 333MHz,

TSSB on Pentium 133MHz (Pezzella and Merelli 2000)

Our algorithm finds an optimal solution to ten out of thirteen problems in relatively very short times. For very famous FT10 with $n \times m = 10 \times 10$, it finds an optimal solution in 0.2 second. Nevertheless, for ABZ5, we could not find any optimal solution reported in the literature, equal to 1234. Besides, note that for *Maxiter* equal to $300n$, TSGW needs a very small amount of CPU times. The longest CPU time of TSGW is equal to 14.9 seconds (on computer Pentium 333), whereas TSSB needs 205 seconds (on Pentium 133) for *Maxiter* equal to $100n$. Finally, note that in the terms of PRD values, TSGW produces significantly better results than TSSB.

Table 5.2. Detailed results for the problem instances of class (b)

LA	OPT or (LB-UB)				TSGW		TSSB		LA	OPT or (LB-UB)				TSGW		TSSB	
	C_{\max}	PRD	CPU	C_{\max}	PRD	C_{\max}	PRD	C_{\max}	PRD	C_{\max}	PRD	CPU	C_{\max}	PRD	C_{\max}	PRD	
10×5										15×10							
1	666	666	0.00	0.0	666	0.00	21	1046	1046	0.00	3.4	1046	0.00	26	1218	1218	0.00
2	655	655	0.00	0.0	655	0.00	22	927	927	0.00	2.7	927	0.00	27	1235	1235	0.00
3	597	597	0.00	0.2	597	0.00	23	1032	1032	0.00	0.2	1032	0.00	28	1216	1216	0.00
4	590	590	0.00	0.0	590	0.00	24	935	936	0.10	0.9	938	0.32	29	1142-1153	1160	1.57
5	593	593	0.00	0.0	593	0.00	25	977	978	0.10	3.7	979	0.20	30	1355	1355	0.00
15×5										20×10							
6	926	926	0.00	0.0	926	0.00	26	1218	1218	0.00	1.0	1218	0.00	31	1784	1784	0.00
7	890	890	0.00	0.0	890	0.00	27	1235	1235	0.00	3.9	1235	0.00	32	1850	1850	0.00
8	863	863	0.00	0.0	863	0.00	28	1216	1216	0.00	4.4	1216	0.00	33	1719	1719	0.00
9	951	951	0.00	0.0	951	0.00	29	1142-1153	1160	1.57	0.9	1168	2.28	34	1721	1721	0.00
10	958	958	0.00	0.0	958	0.00	30	1355	1355	0.00	0.2	1355	0.00	35	1888	1888	0.00
20×5										30×10							
11	1222	1222	0.00	0.0	1222	0.00	31	1784	1784	0.00	0.0	1784	0.00	36	1268	1268	0.00
12	1039	1039	0.00	0.0	1039	0.00	32	1850	1850	0.00	0.0	1850	0.00	37	1397	1411	1.00
13	1150	1150	0.00	0.0	1150	0.00	33	1719	1719	0.00	0.0	1719	0.00	38	1196	1198	0.17
14	1292	1292	0.00	0.0	1292	0.00	34	1721	1721	0.00	0.0	1721	0.00	39	1233	1233	0.00
15	1207	1207	0.00	0.0	1207	0.00	35	1888	1888	0.00	0.1	1888	0.00	40	1222	1225	0.25
10×10										15×15							
16	945	945	0.00	0.6	945	0.00	36	1268	1268	0.00	0.1	1268	0.00	37	1397	1411	2.4
17	784	784	0.00	0.0	784	0.00	38	1196	1198	0.17	2.4	1201	0.42	39	1233	1233	0.00
18	848	848	0.00	3.2	848	0.00	40	1222	1225	0.25	4.5	1240	0.57	40	902	902	0.5
19	842	842	0.00	2.1	842	0.00	all								0.08	0.14	
20	902	902	0.00	0.5	902	0.00											

CPU represents the CPU time on Pentium 333MHz.

Table 5.3. Average results for the instance groups of class (b)

Problem	$n \times m$	TSGW		TSSB	
		PRD (aver.)	CPU (aver.)	PRD (aver.)	CPU (aver.)
LA01-05	10 × 5	0.00	0.1	0.00	9.8
LA06-10	15 × 5	0.00	0.0	0.00	-
LA11-15	20 × 5	0.00	0.0	0.00	-
LA16-20	10 × 10	0.00	1.3	0.00	61.5
LA21-25	15 × 10	0.04	2.2	0.10	115
LA26-30	20 × 10	0.31	2.2	0.46	105
LA31-35	30 × 10	0.00	0.0	0.00	-
LA36-40	15 × 15	0.28	2.6	0.58	141
all		0.08		0.14	

CPU represents the CPU time:

TSGW on Pentium 333MHz,

TSSB on Pentium 133MHz (Pezzella and Merelli 2000)

Table 5.4. Detailed results for the problem instances of class (c)

TA	OPT or (LB-UB)			TSGW		TSSB		TA	OPT or (LB-UB)			TSGW		TSSB	
	C_{\max}	PRD	riptsize	CPU	C_{\max}	PRD			C_{\max}	PRD	CPU	C_{\max}	PRD		
15×15								30×20							
1	1231	1239	0.649		7.9	1241	0.812	41	1859	2023	2033	9.359	3.9	2045	10.005
2	1244	1244	0.000		7.2	1244	0.000	42	1867	1961	1976	5.839	3.2	1979	5.999
3	1218	1218	0.000		4.7	1222	0.328	43	1809	1879	1898	4.920	8.2	1898	4.920
4	1175	1175	0.000		6.6	1175	0.000	44	1927	1998	2031	5.397	44.1	2036	5.656
5	1224	1228	0.327		9.6	1229	0.408	45	1997	2005	2021	1.202	7.1	2021	1.202
6	1238	1238	0.000		9.4	1245	0.565	46	1940	2029	2046	5.464	6.6	2047	5.515
7	1227	1227	0.000		4.5	1228	0.081	47	1789	1913	1937	8.272	4.4	1938	8.329
8	1217	1218	0.082		9.1	1220	0.246	48	1912	1971	1986	3.870	9.2	1996	4.393
9	1274	1287	1.020		9.7	1291	1.334	49	1915	1984	2007	4.804	6.9	2013	5.117
10	1241	1249	0.645		7.1	1250	0.725	50	1807	1937	1971	9.076	5.7	1975	9.297
20×15								50×15							
11	1321-1364	1370	3.709		7.5	1371	3.785	51	2760	2760	0.000	1.7	2760	0.000	
12	1321-1367	1376	4.164		3.9	1379	4.391	52	2756	2756	0.000	3.2	2756	0.000	
13	1271-1350	1355	6.609		4.7	1362	7.160	53	2717	2717	0.000	5.7	2717	0.000	
14	1345	1345	0.000		7.6	1345	0.000	54	2839	2839	0.000	3.1	2839	0.000	
15	1293-1342	1355	4.795		10.1	1360	5.182	55	2679	2681	0.075	5.3	2684	0.187	
16	1300-1362	1369	5.307		11.9	1370	5.385	56	2781	2781	0.000	7.6	2781	0.000	
17	1458-1464	1477	1.303		4.7	1481	1.578	57	2943	2943	0.000	3.8	2943	0.000	
18	1369-1396	1418	3.579		6.9	1426	4.164	58	2885	2885	0.000	2.8	2885	0.000	
19	1276-1341	1350	5.800		9.1	1351	5.878	59	2655	2655	0.000	4.1	2655	0.000	
20	1316-1353	1361	3.419		4.8	1366	3.799	60	2723	2723	0.000	3.6	2723	0.000	
20×20								50×20							
21	1539-1645	1658	7.739		12.0	1659	7.797	61	2868	2868	0.000	3.9	2868	0.000	
22	1511-1601	1620	7.213		13.9	1623	7.412	62	2869	2872	2937	2.370	3.4	2942	2.544
23	1472-1558	1567	6.454		18.8	1573	6.861	63	2755	2755	0.000	5.8	2755	0.000	
24	1602-1651	1656	3.371		11.9	1659	3.558	64	2702	2702	0.000	10.7	2702	0.000	
25	1504-1597	1604	6.649		14.0	1606	6.782	65	2725	2725	0.000	2.7	2725	0.000	
26	1539-1651	1666	8.252		16.7	1666	8.252	66	2845	2845	0.000	4.6	2845	0.000	
27	1616-1687	1693	4.765		22.1	1697	5.012	67	2825	2861	1.274	46.1	2865	1.416	
28	1591-1615	1622	1.948		32.2	1622	1.948	68	2784	2784	0.000	7.3	2784	0.000	
29	1514-1625	1635	7.992		57.0	1635	7.992	69	3071	3071	0.000	4.8	3071	0.000	
30	1473-1585	1602	8.758		5.9	1614	9.572	70	2995	2995	0.000	2.7	2995	0.000	
30×15								100×20							
31	1764	1769	0.283		11.1	1771	0.397	71	5464	5464	0.000	4.8	5464	0.000	
32	1774-1803	1836	3.495		17.3	1840	3.720	72	5181	5181	0.000	3.0	5181	0.000	
33	1778-1796	1831	2.981		25.2	1833	3.093	73	5568	5568	0.000	3.6	5568	0.000	
34	1828-1832	1842	0.766		46.9	1846	0.985	74	5339	5339	0.000	4.3	5339	0.000	
35	2007	2007	0.000		7.9	2007	0.000	75	5392	5392	0.000	5.8	5392	0.000	
36	1819	1820	0.055		14.7	1825	0.330	76	5342	5342	0.000	7.1	5342	0.000	
37	1771-1784	1808	2.089		23.3	1813	2.372	77	5436	5436	0.000	3.0	5436	0.000	
38	1673-1677	1694	1.255		17.6	1697	1.435	78	5394	5394	0.000	2.8	5394	0.000	
39	1795	1812	0.947		19.2	1815	1.114	79	5358	5358	0.000	3.5	5358	0.000	
40	1631-1686	1724	5.702		17.2	1725	5.763	80	5183	5183	0.000	6.5	5183	0.000	
								all							
								2.30							
								2.43							

CPU represents the CPU time on Pentium 333MHz.

Tables 5.2 and 5.3 report the computational results for the Lawrence's test problems (LA01–LA40) of class (b). Table 5.2 shows the detailed results obtained for each instance tested. Instances LA01–LA15 and LA31–LA35 are “easy” because the number of jobs is several times larger than the number of machines. They were solved to optimality by TSGW in less than 0.4 seconds. The more difficult instances LA16–LA30 and LA36–LA40 were solved in less than 4.5 seconds.

Table 5.3 lists the average results for each size (group) $n \times m$ of the instances. For all groups, CPU times of TSGW are very small (on the average). And so, for group with the largest instances LA36–LA40, TSGW needs 2.6 seconds (on Pentium 333), whereas TSSB needs 141 seconds (on Pentium 133). While, for group with the smallest instances LA01–LA05, the respective CPU times are 0.1 and 9.8 seconds. Besides, note that in the terms of PRD values, TSGW produces substantially better results than TSSB.

Tables 5.4 and 5.5 present the results on 80 test problems of class (c) proposed by Taillard (TA01–TA80). It is reported that for 32 out of 80 instances optimal solutions are not known.

Table 5.4 lists detailed results for TA01–TA80. Instances TA51–TA80 are “easy” because the number of jobs is several times larger than the number of machines. Most of them (i.e. 28 out of 30) were solved to optimality by TSGW in less than 10 seconds. For more difficult instances TA31–TA40 and TA41–TA50, the best C_{max} values of TSSB were produced by TSGW in less than 50 seconds. While, for the most difficult instances TA21–TA30 the values were produced in less than 60 seconds. Most of them (i.e. 8 out of 10) were obtained in less than 25 seconds. The longest CPU is reached for TA29 and is equal to 57 seconds.

Table 5.5. Average results for the instance groups of class (c)

Problem	$n \times m$	TSGW		TSSB	
		PRD (aver.)	CPU (aver.)	PRD (aver.)	CPU (aver.)
TA01-10	15 × 15	0.27	7.6	0.45	2175
TA11-20	20 × 15	3.87	7.1	4.13	2526
TA21-30	20 × 20	6.31	20.4	6.52	34910
TA31-40	30 × 15	1.75	20.1	1.92	14133
TA41-50	30 × 20	5.82	9.9	6.04	11512
TA51-60	50 × 15	0.01	4.1	0.02	421
TA61-70	50 × 20	0.36	9.2	0.39	6342
TA71-80	100 × 20	0.00	4.4	0.00	231
all		2.30		2.43	

CPU represents the CPU time:

TSGW on Pentium 333MHz,

TSSB on Pentium 133MHz (Pezzella and Merelli 2000)

Finally, Table 5.5 shows the average results for each size (group) $n \times m$ of instances. For all groups, CPU times of TSGW are extremely small (on the average). And so, for group with the smallest instances TA01–TA10, our algorithm needs 7.6 seconds (on Pentium 333), whereas TSSB needs 2175 seconds (on Pentium 133). While, for the most difficult group TA21–TA30, the respective CPU times are 20.4 and 34910 seconds. Besides, It is noteworthy that in the terms of PRD values, TSGW produces slightly better results than TSSB.

All these results confirm the favorable performance of TSGW in the terms of CPU times and PRD values as well.

5. Conclusions

In this paper we have presented and discussed some new properties of blocks in the job-shop problem. These properties allow us to propose a new, very fast algorithm based on the tabu search approach. In order to decrease the computational effort for the search in TS, we propose calculation of the lower bounds on the makespans instead of computing makespans explicitly for use in selecting the best solution. These lower bounds are used to evaluate the moves for selecting the “best” one. Also, we propose a tabu list with dynamic length which is changed cyclically as the current iteration number of TS increases, using a “pick” in order to carry the search to another area of the solution space. Finally, some perturbations associated with block properties are periodically applied. Computational experiments are given and compared with the results yielded by the best algorithms discussed in the literature. These results show that the algorithm proposed provides much better results than the recent modern approaches. A particular superiority of our algorithm is observed for so-called “hard” problems for which the number of jobs is close to the number of machines. Nevertheless, some improvements in our algorithm are possible. For instance, attempts to refine the lower bounds and perturbations may induce a further reduction of the computational times.

The results obtained encourage us to extend the ideas proposed to other hard problems of sequencing, for example, to the flow-shop problem.

Acknowledgements

This research was supported by KBN Grant 4 T11A 016 24. The authors are grateful to Cesar Rego and the referees for their useful comments and suggestions.

References

- Aarts, E. and J.K. Lenstra (1997) Local Search in Combinatorial Optimization. Wiley, New York.
- Adams, J., E. Balas and D. Zawack (1988) "The Shifting Bottleneck Procedure for Job-Shop Scheduling," *Management Science*, 34(6):391–401.
- Applegate, D. and W. Cook (1991) "A Computational Study of the Job-Shop Scheduling Problem," *ORSA Journal of Computing*, 3:149–156.
- Balas, E. (1969) "Machine Sequencing via Disjunctive Graphs: An Implicit Enumeration Algorithm," *Operations Research*, 17:941–957.
- Balas, E. and A. Vazacopoulos (1998) "Guided Local Search with Shifting Bottleneck for Job-Shop Scheduling," *Management Science*, 44(2):262–275.
- Carlier, J. (1982) "The One-Machine Sequencing Problem," *European Journal of Operational Research*, 1:42–47.
- Carlier, J. and E. Pinson (1989) "An Algorithm for Solving the Job Shop Problem," *Management Science*, 35:164–176.
- Dongarra, J.J. (2004) Performance of Various Computers using Standard Linear Equations Software. Working paper. Computer Science Department, University of Tennessee, USA. <http://www.netlib.org/benchmark/performance.ps>.
- DellAmico, M. and M. Trubian (1993) "Applying Tabu Search to the Job-Shop Scheduling Problem," *Annals of Operations Research*, 4:231–252.
- Fisher, H. and G.L. Thompson (1963) Probabilistic Learning Combinations of Local Job-Shop Scheduling Rules. In J.F. Muth, G.L. Thompson, Editors, *Industrial Scheduling*, Prencite-Hall, Englewood Cliffs, New York.
- Glover, F. (1989) "Tabu search. Part I," *ORSA Journal of Computing*, 1:190–206.
- Glover, F. (1990) "Tabu search. Part II," *ORSA Journal of Computing*, 2:4–32.
- Grabowski, J. (1979) Generalized problems of operations sequencing in the discrete production systems. (Polish), Monographs 9, Scientific Papers of the Institute of Technical Cybernetics of Wroclaw Technical University.
- Grabowski, J. (1980) "On Two-Machine Scheduling with Release and Due Dates to Minimize Maximum Lateness," *Opsearch*, 17:133–154.
- Grabowski, J. (1982) A new Algorithm of Solving the Flow-Shop Problem. In G. Feichtinger and P. Kall, Editors, *Operations Research in Progress*, Reidel Publishing Company, Dordrecht, 57–75.
- Grabowski, J., E. Skubalska and C. Smutnicki (1983) "On Flow-Shop Scheduling with Release and Due Dates to Minimize Maximum Lateness," *Journal of the Operational Research Society*, 34:615–620.
- Grabowski, J., E. Nowicki and S. Zdrzalka (1986) "A Block Approach for Single Machine Scheduling with Release Dates and Due Dates," *European Journal of Operational Research*, 26:278–285.

- Grabowski, J. and J. Janiak (1987) "Job-Shop Scheduling with Resource-Time Models of Operations," *European Journal of Operational Research*, 28:58–73.
- Grabowski, J., E. Nowicki and C. Smutnicki (1988) Block Algorithm for Scheduling of Operations in Job-Shop System. (Polish), *Przeglad Statystyczny*, 35:67–80.
- Grabowski, J. and J. Pempera (2001) New Block Properties for the Permutation Flow-Shop Problem with Application in TS. *Journal of the Operational Research Society*, 52:210–220.
- Internet, <http://mscmga.ms.ic.ac.uk/info.html>.
- Laarhoven, P.V., E. Aarts and J.K. Lenstra (1992) "Job-Shop Scheduling by Simulated Annealing," *Operations Research*, 40:113–125.
- Lawrence, S. (1984) Supplement to "Resource Constrained Project Scheduling: An Experimental Investigation of Heuristic Scheduling Techniques," Technical Report, GSIA, Carnegie Mellon University.
- Matsuo, H., C.J. Suh and R.S. Sullivan (1988) Controlled Search Simulated Annealing Method for the General Job-Shop Scheduling Problem. Working Paper 03-04-88, Department of Management, Graduate School of Business, The University of Texas at Austin.
- Morton, T. and D. Pentico (1993) *Heuristic Scheduling Systems*. Wiley, New York.
- Nowicki, E. and C. Smutnicki (1996a) "A Fast Tabu Search Algorithm for the Permutation Flow-Shop Problem," *European Journal of Operational Research*, 91:160–175.
- Nowicki, E. and C. Smutnicki (1996b) "A Fast Tabu Search Algorithm for the Job-Shop Problem," *Management Science*, 42(6):97–813.
- Pezzella, F. and E. Merelli (2000) "A Tabu Search Method Guided by Shifting Bottleneck for the Job-Shop Scheduling Problem". *European Journal of Operational Research*, 120:297–310.
- Smutnicki, C. (1998) A Two-Machine Permutation Flow-Shop Scheduling with Buffers. *OR Spectrum*, 20:229–235.
- Taillard, E. (1993) "Benchmarks for Basic Scheduling Problems," *European Journal of Operational Research*, 64:278–285.
- Vaessens, R., E. Aarts and J.K. Lenstra (1996) "Job Shop Scheduling by Local Search," *INFORMS Journal of Computing*, 8:303–317.
- Zdrzalka, S. and J. Grabowski (1989) "An Algorithm for Single Machine Sequencing with Release Dates to Minimize Maximum Cost," *Discrete Applied Mathematics*, 23:73–89.

Chapter 6

TABU SEARCH HEURISTICS FOR THE VEHICLE ROUTING PROBLEM

Jean-François Cordeau and Gilbert Laporte

Canada Research Chair in Distribution Management & GERAD, HEC Montréal, 3000ch. Côte-Sainte-Catherine, Montréal, Canada H3T2A7, {cordeau;gilbert}@crt.umontreal.ca.

Abstract This article reviews some of the most important tabu search heuristics for the vehicle routing problem. Some of the main tabu search features are first described: neighbourhood structures, short term memory, long term memory, intensification. The tabu search algorithms are then described, followed by computational results and the conclusion.

Keywords: Vehicle Routing Problem, Tabu Search, Heuristics

1. Introduction

The classical *Vehicle Routing Problem* (VRP) is defined on an undirected graph $G = (V, E)$ where $V = \{v_0, v_1, \dots, v_n\}$ is a vertex set and $E = \{(v_i, v_j) : v_i, v_j \in V, i < j\}$ is an edge set. Vertex v_0 represents a *depot* at which are based m identical vehicles of capacity Q , while the remaining vertices are *cities* or *customers*. A non-negative *cost*, *distance* or *travel time* matrix $C = (c_{ij})$ is defined on E . Each customer v_i has a non-negative demand q_i and a non-negative service time s_i . The VRP consists of determining a set of m vehicle routes i) of minimum total cost; ii) starting and ending at the depot; and such that iii) each customer is visited exactly once by exactly one vehicle; iv) the total demand of any route does not exceed Q ; v) the total duration of any route does not exceed a preset bound D . The number of vehicles is either an input value or a decision variable. In the latter case vehicle fixed costs are sometimes incorporated in the objective function. Most authors in the field work with a fixed number of vehicles. It should be noted, however, that the best

solutions reported (in terms of distance) do not always use the least possible number of vehicles.

The VRP is an important combinatorial optimization problem. It also occupies a central place in distribution management. Toth and Vigo (2002) report that the use of computerized methods in distribution processes often results in savings ranging from 5% to 20% in transportation costs. Golden, Assad and Wasil (2002) and Barker (2002) describe several case studies where the application of VRP algorithms has led to substantial cost savings.

The VRP was introduced by Dantzig and Ramser (1959) more than four decades ago. Since then there has been a steady evolution in the design of solution methodologies, both exact and approximate, for this problem. Yet, no known exact algorithm is capable of *consistently* solving to optimality instances involving more than 50 customers (Golden et al., 1998; Naddef and Rinaldi, 2002). Heuristics are usually used in practice.

Heuristics include *constructive heuristics* (e.g., Clarke and Wright, 1964) which gradually build a feasible solution while attempting to keep solution cost as low as possible, *two-phase heuristics* in which customers are first clustered into feasible routes and actual routes are then constructed (e.g., Gillett and Miller, 1974; Fisher and Jaikumar, 1981), and *improvement methods* which either act on single routes by the application of a *Traveling Salesman Problem* (TSP) heuristic, or on several routes by performing customer reallocations or exchanges. The best known intra-route moves are the classical r -opt interchanges (Croes, 1958; Lin, 1965) and several variants such as Or-opt (Or, 1976), 2-opt* (Potvin and Rousseau, 1995), and 4-opt* (Renaud, Boctor and Laporte, 1996). Inter-route exchanges include those of Fahrion and Wrede (1990), Thompson and Psaraftis (1992), Van Breedam (1994), and Kinderwater and Savelsbergh (1997). As reported by Laporte and Semet (2002), classical heuristics such as the savings method (Clarke and Wright, 1964) and the sweep algorithm (Gillett and Miller, 1974) usually have a high execution speed but often produce solutions having a large gap with respect to the best known solution value (typically between 3% and 7%).

In the last fifteen years, several *metaheuristics* have been put forward for the solution of the VRP. These typically perform a thorough exploration of the solution space, allowing deteriorating and even infeasible intermediate solutions. A number of methods maintain a pool of good solutions which are recombined to produce even better ones. Gendreau, Laporte and Potvin (2002) have identified six families of metaheuristics for the VRP: simulated annealing, deterministic annealing, tabu search, genetic algorithms, ant systems, and neural networks. While the success of any particular method is related to its implementation features, it is fair to say that tabu search (TS) has been highly successful when applied to the VRP. For comparative computational results obtained by means of classical and metaheuristics over the Christofides, Mingozzi

and Toth (1979) (CMT) fourteen benchmark instances, see Gendreau, Laporte and Potvin (2002), and Laporte and Semet (2002).

Our purpose is to compare and analyze some of the best TS implementations for the VRP. We will first describe in Section 2 four important features of TS algorithms. This will be followed in Section 3 by an analysis of the individual TS heuristics with respect to these features and by an assessment and conclusion in Section 4.

2. Tabu search features

Tabu Search was proposed by Glover (1986) and has quickly become one of the best and most widespread local search methods for combinatorial optimization. It performs an exploration of the solution space by moving from a solution x_t identified at iteration t to the best solution x_{t+1} in a subset of the neighbourhood $N(x_t)$ of x_t . Since x_{t+1} does not necessarily improve upon x_t , a *tabu* mechanism is put in place to prevent the process from cycling over a sequence of solutions. A naïve way to prevent cycles is to forbid the process from going back to previously encountered solutions, but doing so would typically require excessive bookkeeping. Instead, some attributes of past solutions are registered and any solution possessing these attributes may not be considered for θ iterations. This mechanism is often referred to as *short term memory*. Other features such as *diversification* and *intensification* are often implemented. The purpose of diversification is to ensure that the search process will not be restricted to a limited portion of the solution space. A possible way to implement diversification is to keep track of past solutions and penalize frequently performed moves. This mechanism was first applied to the VRP by Taillard (1992), and is often called *long term memory*. Intensification consists of performing an accentuated search around the best known solutions. Several survey papers and books have been written on TS, among which Hertz and de Werra (1991), Glover and Laguna (1993, 1997), and Gendreau (2003). In what follows, we examine neighbourhood structures, short term and long term memory strategies, and intensification methods in the VRP context.

2.1 Neighbourhood structures

There are two main ways to define neighbourhood structures in TS algorithms for the classical VRP. The first, termed λ -interchanges by Osman (1991, 1993) consists of exchanging up to λ customers between *two* routes. The second, called *ejection chains* (Rego and Roucairol, 1996) typically acts on more than two routes at the same time.

In λ -interchanges, the value of λ is often restricted to 1 or 2 in order to limit the number of possibilities. Such operations are best described by couples (λ_1, λ_2) (with $\lambda_1 \leq \lambda$ and $\lambda_2 \leq \lambda$) to represent an operation where λ_1 customers are

moved from route 1 to route 2, and λ_2 customers are moved from route 2 to route 1. Thus, disregarding symmetries, the following moves are possible in 2-exchanges: (2, 2), (2, 1), (2, 0), (1, 1), (1, 0). Note that λ -interchanges include swaps (1, 1) and shifts (1, 0). Several variants of this scheme are possible, such as limiting the set of routes or vertices considered for a λ -interchange, or performing a local reoptimization of the routes in which removals or insertions take place. Some of the classical improvement procedures described in Section 1 could be applied in this context. Note that λ -interchanges do not encompass intra-route moves but could be generalized to allow for this possibility.

Ejection chains (Xu and Kelly, 1996; Rego and Roucairol, 1996; Rego, 1998) consist of first identifying a set of routes and then moving vertices in a cyclic manner from route k_1 to route k_2 , from route k_2 to route k_3 , etc. Because the k_i s are not necessarily different, ejection chains in fact allow for both intra-route and inter-route moves.

Designing efficient TS algorithms entails striking a good balance between the quality of the neighbourhood and computational efficiency. More complicated neighbourhoods yield an enriched search since more solutions are considered at each iteration, but at the same time there is a risk that the computational effort will become excessive. This can be circumvented if sparse data structures are used, as in the granular tabu search algorithm (Toth and Vigo, 2003), or if the level of an ejection chain is limited, as in Rego and Roucairol (1996) or Rego (1998) where the complexity of each step is reduced from $O(n^3)$ to $O(n^2)$.

Another issue related to neighbourhoods is whether intermediate infeasible solutions are considered during the search process. One advantage of allowing such moves is that they enable sequences of the type (x_t, x_{t+1}, x_{t+2}) where x_t and x_{t+2} are both feasible, but x_{t+1} is infeasible and x_{t+2} is better than x_t . One way of handling infeasibilities, proposed by Gendreau, Hertz and Laporte (1994), is through the use of a penalized objective function of the type

$$c'(x) = c(x) + \alpha Q(x) + \beta D(x),$$

where $c(x)$ is the routing cost of solution x , $Q(x)$ and $D(x)$ measure the total excess demand and total excess duration of all routes, respectively, and α and β are self-adjusting parameters. Initially set equal to 1, these parameters are periodically reduced (increased) if the last μ solutions were all feasible (all infeasible), and μ is a user-controlled parameter. The advantage of considering infeasible moves is greater in contexts where simple neighbourhood structures are used, such as (1, 0) moves. When sophisticated neighbourhood schemes are employed, like ejection chains, the search process may be able to proceed from x_t to x_{t+2} directly, without going through the infeasible solution x_{t+1} .

2.2 Short term memory

To prevent cycling, it is common to prohibit *reverse moves* for θ iterations. Thus a customer moved from route r to route s at iteration t may be prohibited from being reinserted in route r until iteration $t + \theta$, or it may not leave route s until iteration $t + \theta$. The first documented application of this now common rule can be found in Osman (1991). In Osman (1991, 1993) a fixed value of θ is determined through regression analysis. In his study on the *Quadratic Assignment Problem* Taillard (1991) suggested varying the value of θ during the search. This idea was applied to the VRP by Taillard (1992, 1993) and Gendreau, Hertz and Laporte (1994) who suggest selecting, at each iteration, the value of θ in a given interval $[\underline{\theta}, \bar{\theta}]$ according to a uniform distribution. Other authors, e.g., Cordeau, Laporte and Mercier (2001) use the same value of θ throughout the search.

It is also common to use an *aspiration criterion* to revoke the tabu status of a move if this causes no risk of cycling, for example if this yields a better overall incumbent feasible solution, or a better incumbent among the set of solutions possessing a certain attribute.

2.3 Long term memory

To diversify the search, most TS implementations penalize frequently used solution attributes or frequently performed moves (Glover, 1989). This can be done by adding to the routing cost $c(x_{t+1})$ of x_{t+1} a penalty term equal to the product of three factors: 1) a factor measuring the past frequency of the move; 2) a factor measuring instance size (such as \sqrt{n}); 3) a user-controlled scaling factor. In practice, implementing such a mechanism is both effective and computationally inexpensive. This idea was first proposed by Glover (1989) and later fine tuned by Taillard (1992, 1993, 1994) and by Gendreau, Hertz and Laporte (1994).

2.4 Intensification

Intensification consists of accentuating the search in promising regions. Periodic route improvements by means of TSP algorithms may be classified as intensification techniques. Another way of performing intensification is to conduct a search using a wider neighbourhood structure around some of the best known solutions.

The concept of *Adaptive Memory* is probably the most powerful intensification tool for TS. An adaptive memory is a population of good solutions encountered during the search procedure. It was applied to the VRP for the first time by Rochat and Taillard (1995). Similar to what is done in genetic algorithms, these authors combine solution elements from the population to

construct new solutions. If their value is less than those of some solutions in the population, they are retained and the worst solutions are discarded from the population. In the VRP context, a new solution is initialized from a number of non-overlapping routes extracted from good solutions. Typically these routes do not cover all vertices for otherwise there would be overlaps. The search is then initiated from the selected routes and the unrouted vertices.

3. An overview of TS algorithms

We now present an overview of some of the most popular TS algorithms for the VRP, with an emphasis on the most innovative features of each algorithm.

3.1 Osman's algorithm

It was Osman (1991, 1993) who proposed the concept of λ -interchanges. In his implementation he uses $\lambda = 1$ to generate an initial feasible solution and $\lambda = 2$ in the descent phase. The value $\lambda = 2$ allows a mix of single and double vertex moves, and single and double vertex swaps between vehicle routes. Osman tested two strategies for selecting a neighbour solution. In the first, called best admissible (BA), the best non-tabu solution is selected. In the second, called first best admissible (FBA), the first admissible improving solution is selected if one exists, otherwise the best admissible solution is retained. Osman shows through empirical testing that with the same stopping criterion FBA produces slightly better solutions, but this variant is much slower than BA. Osman's TS implementation uses fixed tabu tenures, no long-term memory mechanism and no intensification schemes.

3.2 The Gendreau, Hertz and Laporte Tabu route algorithm

In Tabu route (Gendreau, Hertz and Laporte, 1994) neighbour solutions are obtained by moving a vertex from its current route r to another route s containing one of its closest neighbours. Insertion into route s is performed concurrently with a local reoptimization, using the GENI mechanism for the TSP (Gendreau, Hertz and Laporte, 1992). This may result in creating a new route or in deleting one. To limit the neighbourhood size, only a randomly selected subset of vertices are considered for reinsertion in other routes. The concept of a penalized objective $c'(x)$ was introduced in Tabu route. The parameters α and β are adjusted every 10 iterations: if the ten previous solutions were feasible with respect to capacity (route duration), then $\alpha(\beta)$ is divided by 2; if they were all infeasible, then $\alpha(\beta)$ is multiplied by 2; otherwise, $\alpha(\beta)$ remains unchanged. Any vertex removed from route r at iteration t is prevented from being reinserted in that route until iteration $t + \theta$, where θ is randomly selected in $[5, 10]$, unless

of course such a move would yield a new incumbent solution. A continuous diversification scheme is also applied. Routes are periodically updated using the US post-optimization heuristic for the TSP (Gendreau, Hertz and Laporte, 1992). Two types of intensification features are used. False starts are employed to perform a limited search from several starting solutions, and then running the main search procedure starting from the best solution obtained. In addition, after a given number of iterations without improvement, a more intense search is performed starting with the incumbent, but allowing this time more vertices to be moved from their current route.

3.3 Taillard's algorithm

While it was published one year earlier than Taburoute, Taillard's (1993) algorithm was developed during the same period. It uses a 1-interchange mechanism without local reoptimization, and without allowing infeasibilities. This scheme is much simpler than that used in Taburoute and allows for more iterations to be performed during the same running time. The tabu mechanism and the long-term memory policy are the same as in Taburoute. Periodically, routes are reoptimized using the exact TSP algorithm of Volgenant and Jonker (1983). The search procedure developed by Taillard employs a decomposition scheme that lends itself to the use of parallel computing. In planar problem, the set of customers is first partitioned into sectors centered at the depot, and also into concentric circles. Search is performed in each subregion by a different processor. The subregion boundaries are updated periodically to provide a diversification effect. In non-planar problems, regions are defined through the computation of shortest spanning arborescences rooted at the depot. Taillard's algorithm remains to this day one of the TS heuristics for the VRP yielding the best solution values. On the fourteen CMT instances, it has produced twelve of the best known solutions.

3.4 The Rochat and Taillard Adaptive Memory Procedure

The Adaptive Memory Procedure (AMP) implemented by Rochat and Taillard was presented under the title "Probabilistic Diversification and Intensification". The method should not be regarded as a VRP heuristic *per se*, but rather as a general procedure applicable to several contexts and in conjunction with several heuristic schemes. For example, it was applied by Bozkaya, Erkut and Laporte (2003) to postoptimize political districts obtained by means of a TS heuristic. Its application to the VRP has helped improve two of the solutions generated by Taillard on the CMT instances, thus yielding the best known incumbent for thirteen of the fourteen instances.

3.5 The Xu and Kelly algorithm

In their TS algorithm, Xu and Kelly (1996) define neighbours by oscillating between ejection chains and vertex swaps between two routes. The ejection chains are determined by minimizing a flow on a network of the type depicted in Figure 6.1. Level 2 and level 3 arcs have a flow of 1 or 0 to indicate whether a customer is removed from a route (level 2) or reinserted in a route (level 3).

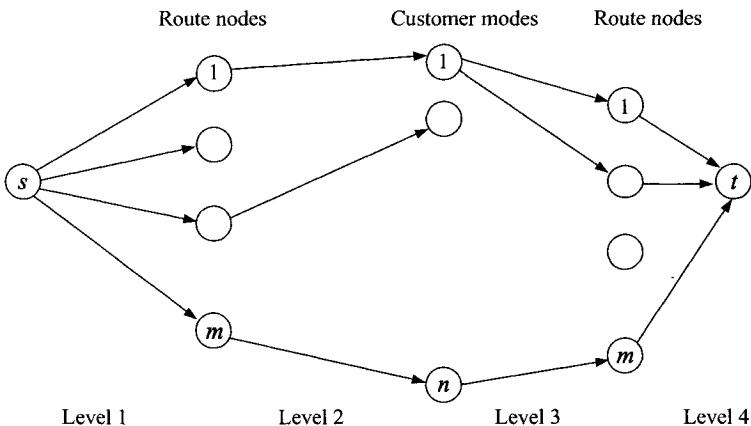


Figure 6.1. Network flow problem for the Xu and Kelly algorithm.

Since several customers can be removed from the same route or reinserted into the same route, all arc costs are approximations only. As in some of the previous algorithms, individual routes are periodically reoptimized by means of 3-opt and 2-opt operations. Intermediate infeasible solutions with respect to capacity are allowed (with $\alpha = 1$), fixed tabu tenures are used, and a long-term frequency based memory is applied.

A pool of best known solutions is maintained and the search procedure is periodically applied to the solutions of this pool in the hope of generating even better ones. Overall, this algorithm has produced several good solutions on the capacity constrained VRP but is rather time consuming and complicated because it requires the setting of several parameters.

3.6 The Rego and Roucairol Tabuchain algorithm

The Rego and Roucairol Tabuchain algorithm (1996) uses ejection chains involving ℓ levels, or routes, to define neighbourhoods. Basically an ejection process bumps a vertex from one level to the successive level, starting at level 1 and ending at level $\ell - 1$. At any level of the ejection chain, the last bumped

vertex may possibly be relocated in the position of the vertex removed from level 1, but also possibly elsewhere (Figure 6.2).

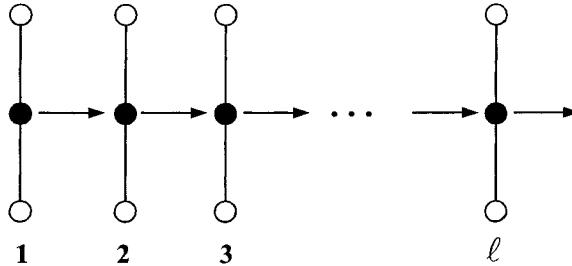


Figure 6.2. An ℓ -level ejection chain. Three vertices are shown for each route. The middle one is ejected.

An ejection chain is considered only if no arc (edge) appears more than once in the solution, but routes that violate capacity or duration constraints are accepted. A sequential and a parallel version of this algorithm were implemented by the authors.

3.7 Rego's Flower algorithm

The Flower algorithm described by Rego (1998) applies an ejection process to move from a VRP solution made up of several *blossoms* (or routes) to another solution by suitably deleting and introducing edges. Applying these operations create intermediate structures consisting of one path, called *stem*, emanating from the depot, and several blossoms attached to it. Thus a blossom may be transformed into a stem (Figure 6.3a) or divided into a blossom and a stem (Figure 6.3b). During this process, the number of active vehicle routes may vary. Candidate solutions consisting only of blossoms are obtained by performing ejection moves that maintain the flower structures. This is done by suitably deleting edges from the structure and inserting new ones. This process is embedded within a TS mechanism by declaring tabu the reintroduction of an edge that has just been deleted. As in some of the previous algorithms, variable tabu tenures are used and a long-term diversification strategy based on move frequencies is applied. Here, the penalty term is randomly generated in an interval whose length is related to \sqrt{n} .

3.8 The Toth and Vigo granular tabu search algorithm

The granularity concept proposed by Toth and Vigo (2003) does not only apply to the VRP or to TS algorithms, but to discrete optimization on the whole. Like AMP, it is a highly portable mechanism. The idea is to permanently remove from consideration long edges that have only a small likelihood of belonging

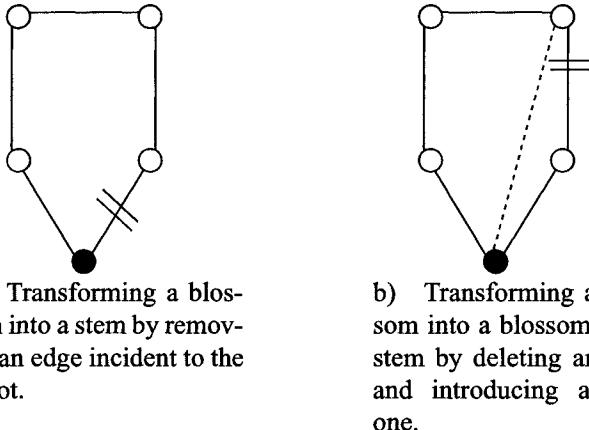


Figure 6.3. Two ejection chain rules in the Flower algorithm. In both cases the depot is represented by the black vertex.

to an optimal solution. More specifically, a threshold is defined and the search is performed on the restricted edge set $E(\nu) = \{(v_i, v_j) \in E : c_{ij} \leq \nu\} \cup I$, where I is a set of *important* edges defined as those incident to the depot. The value of ν is set equal to $\beta\bar{c}$, where β is called a *sparsification parameter*, and \bar{c} is the average edge cost in a good feasible solution quickly obtained, for example, by the Clarke and Wright (1964) algorithm. In practice, selecting β in the interval [1.0, 2.0] results in the elimination of 80% to 90% of all edges. Toth and Vigo have applied this idea in conjunction with some of the main concepts proposed by Taillard (1993) and by Gendreau, Hertz and Laporte (1994).

3.9 The Unified Tabu Search Algorithm of Cordeau, Gendreau, Laporte and Mercier

The Unified Tabu Search Algorithm (UTSA) was first developed in the context of the Periodic VRP and of the Multi-Depot VRP (Cordeau, Gendreau and Laporte, 1997) for which it produced highly competitive results. It was later applied to the Site Dependent VRP with time windows (Cordeau and Laporte, 2001), to the Periodic VRP with and the Multi-Depot VRP with time windows (Cordeau, Laporte and Mercier, 2001) and, finally, to the classical VRP with capacity and route length constraints (Cordeau et al., 2002). UTSA uses several of the features of TabuRoute, namely the same neighbourhood structure, GENI insertions, long-term frequency based penalties. It also differs from TabuRoute in a number of ways. The search is applied to a single initial solution, fixed tabu durations are used and no intensification phase is applied. In other words

an efficiency gain was achieved by sacrificing some of the sophistication of Taburoute.

While UTSA also allows intermediate infeasible solutions, it either divides or multiplies α and β by a factor $1 + \delta$ at each iteration according to whether the previous solution was feasible or not with respect to either capacity or route duration. The tabu mechanism operates on an attribute set $B(x) = \{(i, k) : u_i$ is visited by vehicle k in solution $x\}$. Neighbour solutions are obtained by removing (i, k) from $B(x)$ and replacing it with (i, k') , where $k' \neq k$. Attribute (i, k) is then declared tabu for a number of iterations, and the aspiration criterion is defined relative to attribute (i, k) .

Two interesting features of UTSA are its relative simplicity (it operates with very few parameters) and its flexibility. It has been successfully applied to a large variety of VRP variants by always keeping its parameters at the same value.

4. Assessment and conclusion

Tabu search clearly stands out as one of the best heuristics for the VRP. Over the last fifteen years several implementations have been developed and tested. It is fair to say they have been highly successful in tackling this difficult problem. This success is due in part to a number of key ideas contained in several implementations: efficient neighbourhood evaluation through proper data structures, the allowance of infeasible solutions during the search, the use of self-adjusting parameters, continuous diversification, ejection chains, adaptive memory, and granularity. More than any other feature, these ideas have contributed to the success of TS heuristics and have withstood the test of time.

We present in Table 6.1 comparative computational results for the implementations described in Section 3, over the fourteen CMT benchmark instances. While a proven optimum is known for the first instance only, the best known solutions obtained by Rochat and Taillard (1995) and by Mester and Bäysy (2005) are believed to be near-optimal and are used to assess the degree of accuracy of other heuristics. Most TS implementations used in the comparison yield average deviations of less than 1% from the best known. Computation times are difficult to compare since they are not always reported, different computers are used, some implementations use parallel computing (Taillard, 1993; Rego and Roucairol, 1996), and it is not always clear how many runs were made. For example, Taillard (1993) and Rochat and Taillard (1995) only report the time required to reach a solution whose value is within a given percentage of the best known. In the case of parallel implementations, an upper bound on the running time of an equivalent sequential algorithm can be found by adding up the time taken by each processor. It is worth mentioning that the recent

study by Dongarra (2004) can help compare, to some extent, the relative speed of computers.

We believe there is little scope for major accuracy improvements in the future for the CMT instances. Instead, some effort should be devoted to developing leaner implementations, using fewer parameters and simpler design, even if accuracy deteriorates slightly as a result. Also, algorithms should be easily adaptable to handle the multiplicity of side constraints that arise in practical contexts. As noted by Cordeau et al. (2002) the research community is used to assessing heuristics with respect to accuracy and speed alone, but simplicity and flexibility are often neglected. Yet these two attributes are essential to adoption by end-users and should be central to the design of the next generation of algorithms.

Acknowledgements

This research was partly funded by the Canadian Natural Sciences and Engineering Research Council (NSERC) under grants 227837–00 and OGP0039682, and by the Fonds pour la Formation de chercheurs et l'aide à la recherche (FCAR) under grant 2002–ER–73080. This support is gratefully acknowledged. Thanks are due to Cesar Rego and to a referee for their valuable comments.

Table 6.1. Summary of computational results

Instance	n	Type ¹	Osman			Tabu route			Taillard			Rochat and Taillard			
			Value ²	Deviation	Minutes ³	Value ⁴	Deviation	Minutes ⁵	Value	Deviation	Minutes ⁶	Value	Deviation	Minutes ⁷	
1	50	C	524.61	0.00	1.90	524.61	0.00	6.00	524.61	0.00	0.82	524.61	0.00	0.18	
2	75	C	844.00	1.05	835.77	0.84	835.77	0.06	53.80	835.26	0.00	0.88	835.26	0.00	1.13
3	100	C	838.00	1.44	25.72	829.45	0.40	18.40	826.14	0.00	9.67	826.14	0.00	15.00	
4	150	C	1044.35	1.55	59.33	1036.16	0.75	58.80	1028.42	0.00	63.33	1028.42	0.00	30.00	
5	199	C	1334.55	3.35	54.10	1322.65	2.43	90.90	1298.79	0.58	50.00	1291.45	0.01	—	
6	50	C,D	555.43	0.00	2.88	555.43	0.00	13.50	555.43	0.00	0.28	555.43	0.00	—	
7	75	C,D	911.00	0.15	17.67	931.23	2.37	54.60	909.68	0.00	0.85	909.68	0.00	—	
8	100	C,D	878.00	1.39	49.99	865.94	0.00	25.60	865.94	0.00	18.33	865.94	0.00	—	
9	150	C,D	1184.00	1.85	76.26	1177.76	1.31	71.00	1162.55	0.00	18.33	1162.55	0.00	—	
10	199	C,D	1441.00	3.23	76.02	1418.51	1.62	99.50	1397.94	0.15	—	1395.85	0.00	—	
11	120	C	1043.00	0.09	24.07	1073.47	3.01	22.20	1042.11	0.00	—	1042.11	0.00	45.00	
12	100	C	819.59	0.00	14.87	819.56	0.00	16.00	819.56	0.00	5.67	819.56	0.00	5.83	
13	120	C,D	1547.00	0.38	47.23	1547.93	0.44	59.20	1541.14	0.00	65.00	1541.14	0.00	—	
14	100	C,D	866.37	0.00	19.60	866.37	0.00	63.70	866.37	0.00	25.00	866.37	0.00	—	
Average % deviation from best and time															
• for all instances	1.03	33.60	0.89	46.80	0.05	21.51	0.00	16.19	0.00	—	—	—	—	—	
• for C instances	1.07	25.83	0.95	38.01	0.08	21.73	0.00	16.19	0.00	—	—	—	—	—	
• for C,D instances	1.00	41.37	0.82	55.59	0.02	21.30	0.00	—	—	—	—	—	—	—	

1. C: Capacity restrictions; D: Duration restrictions.

2. FBA version.

3. Minutes on a VAX 8600 computer.

4. Standard algorithm with one set of parameters.

5. Silicon Graphics Workstation (36 MHz, 57 MFlops).

6. Silicon Graphics 4D/35. Time required to reach a solution within 1% of the best known.
7. Silicon Graphics Indigo (100 MHz). Time required to reach a solution within 1% of the best known.

Table 6.1. (continued). Summary of computational results

Instance	n	Type	Xu and Kelly			Tabuchain (sequential)			Tabuchain (parallel)			Flower		
			Value	Deviation	Minutes ⁸	Value	Deviation	Minutes ⁹	Value	Deviation	Minutes ¹⁰	Value	Deviation	Minutes ¹⁰
1	50	C	524.61	0.00	4.89	524.61	0.00	1.05	524.61	0.00	0.85	524.81	0.04	0.17
2	75	C	835.26	0.00	37.94	837.50	0.01	43.38	835.32	0.27	16.80	847.00	1.41	0.38
3	100	C	826.14	0.00	47.45	827.53	0.17	36.72	827.53	0.17	33.90	832.04	0.71	0.93
4	150	C	1029.56	0.11	100.81	1034.29	1.55	48.47	1044.35	2.52	27.20	1047.21	1.83	1.91
5	199	C	1298.58	0.56	207.80	1338.49	3.35	77.07	1334.55	3.66	16.25	1351.18	4.64	7.06
6	50	C,D	—	—	—	555.43	0.00	2.38	555.43	0.00	3.17	559.25	0.69	0.06
7	75	C,D	—	—	—	909.68	0.00	82.95	909.68	0.00	23.10	922.21	1.38	0.50
8	100	C,D	—	—	—	868.29	0.09	18.93	866.75	0.27	8.60	876.97	1.27	1.41
9	150	C,D	—	—	—	1178.84	0.14	29.85	1164.12	1.40	15.55	1191.30	2.47	6.07
10	199	C,D	—	—	—	1420.84	1.79	42.72	1420.84	1.79	52.02	1460.83	4.66	5.06
11	120	C	1042.11	0.00	2.80	1043.54	0.00	11.23	1042.11	0.14	6.30	1052.04	0.95	3.96
12	100	C	819.56	0.00	5.61	819.56	0.00	1.57	819.56	0.00	1.22	821.63	0.25	0.54
13	120	C,D	—	—	—	1550.17	0.59	1.95	1550.17	0.59	2.00	1558.06	1.10	3.69
14	100	C,D	—	—	—	866.53	0.00	24.65	866.37	0.02	9.42	867.79	0.16	0.75
Average % deviation from best and time														
from best and time			—			—			0.77			0.55		
• for all instances			—			—			30.21			15.46		
• for C instances			0.10			58.19			0.96			31.36		
• for C,D instances			—			—			0.58			0.37		
									29.06			0.37		
									16.27			1.68		

8. DEC Alpha workstation (DEC OSF/1 v3.0).

9. Sun Sparc IPC. The value reported for the parallel version is the total time on four of these computers.
10. HP 9000/712.

Table 6.1. (continued). Summary of computational results

Instance	<i>n</i>	Type	Granular Tabu Search			UTSA			Best ¹³
			Value	Deviation	Minutes ¹¹	Value	Deviation	Minutes ¹²	
1	50	C	524.61	0.00	0.21	524.61	0.00	4.57	524.61
2	75	C	838.60	0.40	2.21	835.45	0.02	7.27	835.26
3	100	C	828.56	0.29	2.39	829.44	0.40	11.23	826.14
4	150	C	1033.21	0.47	4.51	1038.44	0.97	18.72	1028.42
5	199	C	1318.25	2.09	7.50	1305.87	1.13	28.10	1291.29
6	50	C,D	555.43	0.00	0.86	555.43	0.00	4.61	555.43
7	75	C,D	920.72	1.21	2.75	909.68	0.00	7.55	909.68
8	100	C,D	869.48	0.41	2.90	866.38	0.05	11.17	865.94
9	150	C,D	1173.12	0.91	5.67	1171.81	0.80	19.17	1162.55
10	199	C,D	1435.74	2.86	9.11	1415.40	1.40	29.74	1395.85
11	120	C	1042.87	0.07	3.18	1074.13	3.07	14.15	1042.11
12	100	C	819.56	0.00	1.70	819.56	0.00	10.99	819.56
13	120	C,D	1545.51	0.28	9.34	1568.91	1.80	14.53	1541.14
14	100	C,D	866.37	0.00	1.41	866.53	0.02	10.65	866.37
Average % deviation from best and time									
• for all instances			0.64	3.84		0.69	13.75		
• for C instances			0.47	3.10		0.80	13.58		
• for C,D instances			0.81	4.58		0.58	13.92		

11. Pentium (200 MHz).

12. Sun Ultrasparc 10 (440 MHz).

13. References in which the best known solution values were first reported: 1, 6, 14 (Osman, 1991); 2, 3, 4, 7, 8, 9, 11, 12, 13 (Taillard, 1993); 5 (Mester and Bräysy, 2005); 10 (Rochat and Taillard, 1995). Some best known solution values may have been previously published in working papers, theses, etc. The solution value of instance 1 is known to be optimal (Hadjiconstantinou, Christofides and Mingozzi, 1995).

References

- Barker, E.K. (2002) "Evolution of Microcomputer-Based Vehicle Routing Software: Case Studies in United States," in *The Vehicle Routing Problem*, P. Toth and D. Vigo (eds), SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 353–362.
- Bozkaya, B., E. Erkut and G. Laporte (2003) "A Tabu Search Heuristic and Adaptive Memory Procedure for Political Districting," *European Journal of Operational Research*, 144:12–26.
- Christofides, N., A. Mingozzi and P. Toth (1979) "The Vehicle Routing Problem," in *Combinatorial Optimization*, N. Christofides, A. Mingozzi and P. Toth (eds), Wiley, Chichester, 315–338.
- Clarke, G. and J.W. Wright (1964) "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points," *Operations Research*, 12:568–581.
- Cordeau, J.-F., M. Gendreau and G. Laporte (1997) "A Tabu Search Heuristic for Periodic and Multi-Depot Vehicle Routing Problems," *Networks*, 30:105–119.
- Cordeau, J.-F., M. Gendreau, G. Laporte, J.-Y. Potvin and F. Semet (2002) "A Guide to Vehicle Routing Heuristics," *Journal of the Operational Research Society*, 53:512–522.
- Cordeau, J.-F. and G. Laporte (2001) "A Tabu Search Algorithm for the Site Dependent Vehicle Routing Problem with Time Windows," *INFOR*, 39:292–298.
- Cordeau, J.-F., G. Laporte and A. Mercier (2001) "A Unified Tabu Search Heuristic for Vehicle Routing Problems with Time Windows," *Journal of the Operational Research Society*, 52:928–936.
- Croes, G.A. (1958) "A Method for Solving Traveling-Salesman Problems," *Operations Research*, 6:791–812.
- Dantzig, G.B. and J.H. Ramser (1959) "The Truck Dispatching Problem," *Management Science*, 6:80–91.
- Dongarra, J.J. (2004) "Performance of Various Computers Using Standard Linear Equations Software," Technical Report CS-89-85. Computer Science Department, University of Tennessee.
- Fahrion, R. and M. Wrede (1990) "On a Principle of Chain-Exchange for Vehicle-Routeing Problems (1-VRP)," *Journal of the Operational Research Society*, 41:821–827.
- Fisher, M.L. and R. Jaikumar (1981) "A Generalized Assignment Heuristic for the Vehicle Routing Problem," *Networks*, 11:109–124.
- Gendreau, M. (2003) "An Introduction to Tabu Search," in *Handbook of Metaheuristics*, F. Glover and G.A. Kochenberger (eds), Kluwer, Boston, 37–54.

- Gendreau, M., A. Hertz and G. Laporte (1992) "New Insertion and Postoptimization Procedures for the Traveling Salesman Problem," *Operations Research*, 40:1086–1094.
- Gendreau, M., A. Hertz and G. Laporte (1994) "A Tabu Search Heuristic for the Vehicle Routing Problem," *Management Science*, 40:1276–1290.
- Gendreau, M., G. Laporte and J.-Y. Potvin (2002) "Metaheuristics for the VRP," in *The Vehicle Routing Problem*, P. Toth and D. Vigo (eds), SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 129–154.
- Gillett, B.E. and L.R. Miller (1974) "A Heuristic Algorithm for the Vehicle Dispatch Problem," *Operations Research*, 22:340–349.
- Glover, F. (1986) "Future Paths for Integer Programming and Links to Artificial Intelligence," *Computers & Operations Research*, 13:533–549.
- Glover, F. (1989) "Tabu Search – Part I," *ORSA Journal on Computing*, 1:190–206.
- Glover, F. and M. Laguna (1993) "Tabu Search," in *Modern Heuristic Techniques for Combinatorial Problems*, C.R. Reeves (ed.), Blackwell, Oxford, 70–150.
- Glover, F. and M. Laguna (1997) *Tabu Search*. Kluwer, Boston.
- Golden, B.L., A.A. Assad and E.A. Wasil (2002) "Routing Vehicles in the Real World: Applications in the Solid Waste, Beverage, Food, Dairy, and Newspaper Industries," in *The Vehicle Routing Problem*, P. Toth and D. Vigo (eds), SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 245–286.
- Golden, B.L., E.A. Wasil, J.P. Kelly and I-M. Chao (1998) "Metaheuristics in Vehicle Routing," in *Fleet Management and Logistics*, T.G. Crainic and G. Laporte (eds), Kluwer, Boston, 33–56.
- Hadjiconstantinou, E.A., N. Christofides and A. Mingozzi (1995) "A New Exact Algorithm for the Vehicle Routing Problem Based on q -Paths and k -Shortest Paths Relaxation," *Annals of Operations Research*, 61:21–43.
- Hertz, A. and D. de Werra (1991) "The Tabu Search Metaheuristic: How We Used It," *Annals of Mathematics and Artificial Intelligence*, 1:111–121.
- Kinderwater, G.A.P. and M.W.P. Savelsbergh (1997) "Vehicle Routing: Handling Edge Exchanges," in *Local Search in Combinatorial Optimization*, E.H.L. Aarts and J.K. Lenstra (eds). Wiley, Chichester, 337–360.
- Laporte, G. and F. Semet (2002) "Classical Heuristics for the Capacitated VRP," in *The Vehicle Routing Problem*, P. Toth and D. Vigo (eds), SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 109–128.
- Lin, S. (1965) "Computer Solution of the Traveling Salesman Problem," *Bell System Technical Journal*, 44:2245–2269.
- Mester, D. and O. Bräysy (2005) "Active Guided Evolution Strategies for the Large Scale Vehicle Routing Problems with Time Windows," *Computers & Operations Research*, forthcoming.

- Naddef, D. and G. Rinaldi (2002) "Branch-and-Cut Algorithms for the VRP," in *The Vehicle Routing Problem*, P. Toth and D. Vigo (eds), SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 53–84.
- Or, I. (1976) "Traveling Salesman-Type Combinatorial Problems and their Relation to the Logistics of Regional Blood Banking," Ph.D. Thesis, Northwestern University, Evanston, IL.
- Osman, I.H. (1991) "Metastrategy Simulated Annealing and Tabu Search Algorithms for Combinatorial Optimization Problems," Ph.D. Thesis, The Management School, Imperial College, London.
- Osman, I.H. (1993) "Metastrategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem," *Annals of Operations Research*, 41:421–451.
- Potvin, J.-Y. and J.-M. Rousseau (1995) "An Exchange Heuristic for Routing Problems with Time Windows," *Journal of the Operational Research Society*, 46:1433–3446.
- Rego, C. (1998) "A Subpath Ejection Method for the Vehicle Routing Problem," *Management Science*, 44:1447–1459.
- Rego, C. and C. Roucairol (1996) "A Parallel Tabu Search Algorithm Using Ejection Chains for the Vehicle Routing Problem," In *Meta-Heuristics: Theory and Applications*, Kluwer, Boston, 661–675.
- Renaud, J., F.F. Boctor and G. Laporte (1996) "A Fast Composite Heuristic for the Symmetric Traveling Salesman Problem," *INFORMS Journal on Computing*, 8:134–143.
- Rochat, Y. and É.D. Taillard (1995) "Probabilistic Diversification and Intensification in Local Search for Vehicle Routing," *Journal of Heuristics*, 1:147–167.
- Taillard, É.D. (1991) "Robust Taboo Search for the Quadratic Assignment Problem," *Parallel Computing*, 17:433–445.
- Taillard, É.D. (1992) "Parallel Heuristic Search Methods for the Vehicle Routing Problem," Working Paper ORWP 92/03, Département de mathématiques, École Fédérale Polytechnique de Lausanne, Switzerland.
- Taillard, É.D. (1993) "Parallel Iterative Search Methods for Vehicle Routing Problem," *Networks*, 23:661–673.
- Taillard, É.D. (1994) "Parallel Taboo Search Techniques for the Job Shop Scheduling Problem," *ORSA Journal on Computing*, 6:108–117.
- Thompson, P.M. and H.N. Psaraftis (1993) "Cyclic Transfer Algorithms for Multi-Vehicle Routing and Scheduling Problems," *Operations Research*, 41:935–946.
- Toth, P. and D. Vigo (2003) "The Granular Tabu Search and its Application to the Vehicle Routing Problems," *INFORMS Journal on Computing*, 15:333–346.

- Toth, P. and D. Vigo (2002) "An Overview of Vehicle Routing Problems," in *The Vehicle Routing Problem*, P. Toth and D. Vigo (eds), SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 1–26.
- Van Breedam, A. (1994) "An Analysis of the Behavior of Heuristics for the Vehicle Routing Problem for a Selection of Problems with Vehicle-Related, Customer-Related, and Time-Related Constraints," Ph.D. Dissertation, University of Antwerp, 1994.
- Volgenant, A. and R. Jonker (1983) "The Symmetric Traveling Salesman Problem and Edge Exchanges in Minimal 1-Tree," *European Journal of Operational Research*, 12:394–403.
- Xu, J. and J.P. Kelly (1996) "A Network Flow-Based Tabu Search Heuristic for the Vehicle Routing Problem," *Transportation Science*, 30:379–393.

Chapter 7

SOME NEW IDEAS IN TS FOR JOB SHOP SCHEDULING

Eugeniusz Nowicki and Czeslaw Smutnicki

*Wroclaw University of Technology, Institute of Engineering Cybernetics, Janiszewskiego 11-17,
50-372 Wroclaw, Poland, {enowicki,smutnick}@ict.pwr.wroc.pl*

Abstract This paper deals with a classic job-shop scheduling problem with the makespan criterion. Some properties of the fast algorithm TSAB of Nowicki and Smutnicki, as well as properties of the solution space, have been discussed and examined. Next, by introducing the original method of diversification, TSAB has been embedded in more advanced algorithmic framework *i*-TSAB, which has far analogy to path relinking technique. Newly proposed algorithm is faster and has better accuracy than TSAB, runs within time of minutes on a PC and already in preliminary tests provides 23 new upper bounds among 35 unsolved yet common Taillard's benchmarks.

Keywords: Job-Shop Scheduling, Tabu Search, Path Relinking

1. Introduction

The job shop scheduling problem is known as the particularly hard combinatorial optimization case. It is also considered as the permanent indicator of practical efficiency of advanced scheduling algorithms, thus manifest indirectly the power of scheduling theory. The problem follows from OR practice, has relatively simple formulation, excellent industrial applications, finite but huge number of solutions and unfortunately is strongly NP-hard. That's why, a lot of various methods have been designed to solve the problem in a quick time. The job-shop problem has been studied by many authors and quite a lot of *optimization algorithms* and *approximate algorithms* have been already proposed. Skipping consciously the long list of particular papers, we refer to only few, very recent, excellent state of art reviews performed in Blazewicz et al. (1996); Cheng et al. (1996); Jain and Meeran (1999); Vaessens et al. (1996).

In early nineties, after a series of works dealing with optimization algorithms (chiefly of Branch-and-Bound type), it became completely clearly for everybody, that pure optimization methods reached a limit. They cannot solve instances with more than 200 operations in a reasonable time (hours, days, weeks), although considerable progress had been made in B&B approach.

Since that time, research effort has focused on approximate approaches. Made works improved significantly algorithms efficacy, measured by the accuracy as opposing to the running time. The way of development has long, rich history, and takes a walk from dispatching priority rules, through the widely studied variants of shifting bottleneck approach, geometric approach, job insertions, local neighborhood search, neural networks, to modern simulated annealing, constraint satisfaction, tabu search, and evolutionary search.

New era started when few job-shop algorithms proposed in Taillard (1993), Nowicki and Smutnicki (1996), DellAmico and Trubian (1993) had appeared - algorithms are based on TS approach of Glover (1989, 1990, 1997). Simply and almost ascetic algorithm TSAB, Nowicki and Smutnicki (1996), designed originally in 1993, found optimal solution of the notorious job-shop instance FT10 (100 operations) in a few seconds on a PC; this instance had waited 26 years, since 1963, to be solved by an optimization algorithm. TSAB is the first one, which enabled to solve, in a very short time on a PC, instances of size up to 2,000 operations with unprecedented accuracy below 4% on average, which is and so considerably better than approximately 20% for special insertion technique, 35% for standard priority rules and over 130% for random solutions. Just after TSAB had appeared, it received enthusiastic as well as skeptical opinions. Enthusiasts tried to examine and enrich TSAB phenomena, chiefly in order to improve further its numerical results - TS was a drug on notorious hardness of the job-shop problem. Skeptics questioned measurement of the computer speed, reported running time, complexity of finding the starting point, etc. – all submitted doubts will be dispelled in the sequel.

Further exploration of TSAB idea evolves into two independent directions: speed acceleration and more sophisticated diversification mechanism. Although a few papers have dealt with these subject, from various points of view, we propose new one, which embeds proposed earlier algorithm TSAB in more general framework with far analogy to the path relinking philosophy. Results of research accompanying to the creation of new algorithm provide new look on TSAB as well as on properties of the solution space. As the immediate practical result of our new approach, we have found 23 better upper bounds among 35 unsolved yet instances from the common benchmark set of Taillard, Taillard (1993), attacked by all job-shop algorithms designed over the world. The proposed algorithm still runs on a standard PC with moderate speed in a time of minutes.

The paper is organized as follows. After the problem formulation, in Section 2, with the help of introduced by us the convenient permutation-graph model, we provide in Section 3 the brief characteristics of fundamental properties, used for the construction of TS algorithms. Section 4 reminds these elements of the classical TSAB algorithm, which are necessary for understanding our new proposals. Section 5 presents several new theoretical properties of TSAB, which increase only TSAB speed, preserving its accuracy. Equipped with better tools, obtained this way, in Section 6 we analyze more precisely TSAB search trajectories and solution space topology. After the strategy aspects of the search discussed in Section 7, we introduce in Section 8 new diversification mechanism, which leads to algorithm *i*-TSAB of high accuracy, presented in Section 9. Numerical tests discussed in Section 10 completes the paper.

2. Problem, Models, Denotations

The job-shop problem is defined formally as follows. There are a set of jobs $J = \{1, \dots, r\}$, a set of machines $M = \{1, \dots, m\}$ and a set of operations $O = \{1, \dots, o\}$. Set O is decomposed into subsets corresponding to the jobs. Job j consists of a sequence of o_j operations indexed consecutively by $(l_{j-1} + 1, \dots, l_{j-1} + o_j)$ which should be processed in that order, where $l_j = \sum_{i=1}^j o_i$, is the total number of operations of the first j jobs, $j = 1, \dots, r$ ($l_0 = 0$), and $\sum_{j=1}^r o_j = o$. Operation i must be processed on machine $\mu_i \in M$ during an uninterrupted processing time $p_i > 0$, $i \in O$. We assume, without loss of generality, that any successive operations of the same job are going to be processed on different machines. Each machine can process at most one operation at a time. A feasible schedule is defined by start times $S_i \geq 0$, $i \in O$, such that the above constraints are satisfied. The problem is to find a feasible schedule that minimizes the makespan $\max_{i \in O}(S_i + p_i)$.

In the analysis we use the permutation-graph model introduced first time in Nowicki and Smutnicki (1996). Set of operations O can be naturally decomposed into subsets $M_k = \{i \in O : \mu_i = k\}$ each of them corresponding to operations which should be processed on machine k , $k \in M$. The processing order of operations on machine k can be defined by permutation $\pi_k = (\pi_k(1), \dots, \pi_k(|M_k|))$ on M_k , $k \in M$; $\pi_k(i)$ denotes the element of M_k which is in position i in π_k . Let Π_k be the set of all permutations on M_k . The processing order of operations on machines is defined by m -tuple $\pi = (\pi_1, \dots, \pi_m)$, where $\pi \in \Pi = \Pi_1 \times \Pi_2 \times \dots \times \Pi_m$.

For the processing order π , we create the digraph $G(\pi) = (O, R \cup E(\pi))$ with a set of nodes O and a set of arcs $R \cup E(\pi)$, where $R = \bigcup_{j=1}^r \bigcup_{i=1}^{o_j-1} \{(l_{j-1} + i, l_{j-1} + i + 1)\}$ and $E(\pi) = \bigcup_{k=1}^m \bigcup_{i=1}^{|\pi_k|-1} \{(\pi_k(i), \pi_k(i + 1))\}$. Arcs from set R represent the processing order of operations in jobs, whereas arcs from set $E(\pi)$ represent the processing order of operations on machines. Each node

$i \in O$ in the digraph has weight p_i and each arc has weight zero. The processing order π is feasible if the graph $G(\pi)$ does not contain a cycle. It is well-known that any feasible processing order π generates a feasible schedule S_i , $i \in O$. Moreover, start time S_i equals the length of the longest path going to the vertex i (but without p_i) in $G(\pi)$, $i \in O$; if a node has no predecessors then the length is zero. Makespan $C_{\max}(\pi)$ for the processing order π equals the length of the longest path (critical path) in $G(\pi)$. Now we can rephrase the job shop problem as that of finding the feasible processing order $\pi \in \Pi$ which minimizes $C_{\max}(\pi)$.

Let $u = (u_1, \dots, u_w)$, where $u_i \in O$, be the critical path in $G(\pi)$ (w is the number of nodes in u). This path can be naturally decomposed into $b \leq w$ subsequences B_1, \dots, B_b called blocks in π on u , where: (i) B_j contains operations processed on the same machine, $1 \leq j \leq b$, (ii) two consecutive blocks contain operations processed on different machines; see among others Grabowski et al. (1988) for detail.

3. Moves, Neighborhoods, Properties

In this section we discuss essential job-shop properties in the context of TS applications, taking account of efficacy of the algorithm.

The *neighborhood* (a basic element of local search procedures) of the given solution π is a subset of solutions “close” to π , generated by the application of various types of *moves*. For solutions derived from permutations, thus also for processing order π , there are commonly used: A-move (swap a pair $v = (x, y)$ of two adjacent operations x, y processed on a machine), S-move (swap a pair $v = (x, y)$ of two operations x, y processed on the same machine), I-move (for a pair $v = (x, y)$ of operations x, y processed on the same machine $k = \mu_x = \mu_y$, cut out operation x from its original position and then insert it immediately after operation y if x precedes y in the permutation π_k , or immediately before operation y otherwise). Let $\pi_{(v)}$ denote the processing order obtained after the move v . The set of all moves of type Z performed from π will be denoted by $V^Z(\pi)$, whereas the obtained neighborhood by $\mathcal{N}^Z(\pi) = \{\pi_{(v)} : v \in V^Z(\pi)\}$, $Z \in \{A, S, I\}$.

There are two main problems with neighborhoods generated by such moves: size and feasibility. To resolve both problems, partially or completely, all known algorithms for job-shop with the makespan criterion employ, directly or indirectly, at least special elimination properties followed from the critical path, Balas (1969), Laarhoven et al. (1992), and/or blocks, Grabowski et al. (1988). The general aim of these properties is to eliminate from the considerations some selected moves, for which it is known a priori, without computing the makespan, that they will not immediately improve $C_{\max}(\pi)$. These advisable moves will be called next *useless* moves. The elimination of useless

moves correlates with feasibility (a non-useless move can lead to $\pi_{(v)}$ with a cycle in the graph $G(\pi_{(v)})$) and connectivity (for any starting solution, there exists search trajectory leading to optimal solution). Note that the check of feasibility is so expensive as the explicit makespan calculation. An incomplete (non-exhaustive) reduction leaves connectivity and sometimes unfeasible moves, whereas too deep reduction eliminates all useless moves but also loses connectivity. A reasonable balance between these extreme techniques is welcome.

All known and these newly designed neighborhoods for the job-shop problem apply moves to operations from a critical path only; other moves can be reduced due to elimination property of this path. We will refer to two such neighborhoods, both generated by subsets of moves from $V^A(\pi)$. The first neighborhood follows from the paper Laarhoven et al. (1992), is generated by move set $V^{A1}(\pi) \subseteq V^A(\pi)$, where $v = (x, y) \in V^{A1}(\pi)$ if only x and y belongs to the critical path u in $G(\pi)$. All these moves are feasible but some of them are useless. The second neighborhood is generated by the move set $V^{A2}(\pi) \subseteq V^{A1}(\pi)$, obtained from $V^{A1}(\pi)$ by elimination of all useless moves with the help of block property, Grabowski et al. (1988); this technique is also known as “interchanges near the borderline of blocks on a single critical path”. More precisely, we build $V^{A2}(\pi)$ on the base of the critical path u in $G(\pi)$ and blocks B_1, \dots, B_b defined for this u . We accept only these moves from $V^{A1}(\pi)$, that swap the first two and last two operations in every block B_2, \dots, B_{b-1} , each of which contains at least two operations. In the first block B_1 we accept move that swap only the last two operations, and via symmetry in the last block – that swap only the first two operations. It was shown that, if $V^{A2}(\pi)$ is empty, then the processing order π is optimal, Nowicki and Smutnicki (1996). Neighborhood $\mathcal{N}^{A1}(\pi)$ is connected, whereas $\mathcal{N}^{A2}(\pi)$ is not connected.

4. Algorithm TSAB

In this section we briefly outline algorithm TSAB, which constitutes the foundation for the whole analysis carried out next.

TSAB works on the move set $V^{A2}(\pi)$. For simplicity of denotation the move set $V^{A2}(\pi)$ and the neighborhood $\mathcal{N}^{A2}(\pi)$ will be denoted frequently by $H(\pi)$ and $\mathcal{H}(\pi)$, respectively. In TSAB, the single neighborhood $\mathcal{H}(\pi)$ is searched using Neighborhood Searching Procedure (NSP), see Figure 7.1. It uses tabu list $T = (T_1, \dots, T_{maxt})$ of a fixed length $maxt$, which stores forbidden moves $v = (x, y)$. Operator \oplus adds the move to the list in standard way. For a move $v' = (x, y)$ performed, an *inverse move* $\bar{v}' = (y, x)$ becomes forbidden and will be added to T ; this provides new form of tabu list $T' := T \oplus \bar{v}'$.

Algorithm NSP

Starts with a processing order π , a non-empty $H(\pi)$, a current tabu list T , a best known makespan C^* , and returns move v' , the new processing order π' and the modified tabu list T' .

Step 1. Find sets $U = H(\pi) \setminus T$ and $FP = \{v \in H(\pi) \cap T : C_{\max}(\pi_{(v)}) < C^*\}$. If $U \cup FP \neq \emptyset$ then select $v' \in U \cup FP$ so that $C_{\max}(\pi_{(v')}) = \min\{C_{\max}(\pi_{(v)}) : v \in U \cup FP\}$ and goto 3.

Step 2. If $|H(\pi)| = 1$ then select $v' \in H(\pi)$. Otherwise repeat $T := T \oplus T_{\max t}$ until $H(\pi) \setminus T \neq \emptyset$. Then select $v' \in H(\pi) \setminus T$.

Step 3. Set $\pi' := \pi_{(v')}$ and $T' := T \oplus \overline{v'}$.

Figure 7.1. Neighborhood Searching Procedure (NSP)

NSP searches using three categories of moves: unforbidden $U = H(\pi) \setminus T$, forbidden but profitable $FP = \{v \in H(\pi) \cap T : C_{\max}(\pi_{(v)}) < C^*\}$ and forbidden but nonprofitable $FN = H(\pi) \setminus (U \cup FP)$. We select the best move v' among those in the set $U \cup FP$. If all moves are in FN (i.e. $U \cup FP = \emptyset$) and $|FN| > 1$, we modify the tabu list accordingly to the recipe “repeat $T := T \oplus T_{\max t}$ until $H(\pi) \setminus T \neq \emptyset$ ”. After this eventual modification, the set $H(\pi) \setminus T$ contains one move which is being chosen next.

The basic TS algorithm, called TSA, simply iterates NSP. It has no diversification, it works fast due to small neighborhood and owns quite good numerical properties. The advanced TS algorithm called Tabu Search Algorithm with Back Jump Tracking (TSAB) has some diversification elements, related to a strategy that resumes the search from unvisited neighbors of solutions previously generated, see Figure 7.2.

Algorithm TSAB

Begins with the processing order π^* (found by any heuristic algorithm), $\pi = \pi^*$, $C^* = C_{\max}(\pi^*)$, $T = \emptyset$, $L = \emptyset$ (i.e. $l = 0$), $iter = 0$ and $save := true$. The flag $save$ controls the process of storing on the list $L = (L_1, \dots, L_l)$. TSAB returns the best processing order π^* and $C^* = C_{\max}(\pi^*)$.

Step 1. Set $iter := iter + 1$. Find the set of moves $H(\pi)$.

If $H(\pi) = \emptyset$ then *STOP*; π^* is optimal.

Step 2. Find the move $v' \in H(\pi)$, the neighbor $\pi' = \pi_{(v')}$ and the modified tabu T' by applying *NSP*.

If $save = true$ and $H(\pi) \setminus v' \neq \emptyset$ then set $L := shift(L)$, $l = \min\{l + 1, maxl\}$, $L_l = (\pi, H(\pi) \setminus v', T)$.

Set $\pi := \pi'$, $T := T'$ and $save := false$.

Step 3. If $C_{\max}(\pi) < C^*$ then set $\pi^* := \pi$, $C^* := C_{\max}(\pi)$, $iter := 0$, $save := true$ and go to 1.

Step 4. If ($iter \leq maxiter$) and (not *IsCykle(iter)*) then go to 1.

Step 5. If $l \neq 0$ then set $(\pi, H(\pi), T) := L_l$, $l := l - 1$, $iter := 1$, $save := true$ and go to 2; otherwise *STOP*.

Figure 7.2. Tabu Search Algorithm with Back Jump Tracing (TSAB)

During the run of TSAB we store, on the list $L = (L_1, \dots, L_l)$, the last $l \leq maxl$ best processing orders, where $maxl$ is a parameter. The element $L_j = (\pi, V, T)$ of this list is a triple: a processing order, set of moves, and a tabu list, $1 \leq j \leq l$. If we have found, in some iteration $iter$, the processing order π such that $C_{\max}(\pi) < C^*$ (see Step 3) and $H(\pi) \setminus v' \neq \emptyset$, then we add do the list the element $L_l = (\pi, H(\pi) \setminus v', T)$, $l = \min\{l + 1, maxl\}$, where v' is a move which will be performed from π . (Note that $H(\pi)$ and v' will be found in iteration $iter + 1$, see Step 1 and 2.) In order to prevent overloading, the operator “shift” is processed on the list before new list element will be added; $shift(L) = L$ if $l < maxl$, otherwise operator $shift(L)$ moves list elements to the left, i.e. $L_j := L_{j+1}$, $j = 1, \dots, maxl - 1$. Just after $maxiter$ non-improving iterations has been performed, TSAB continues search starting with the processing order, set of moves and the tabu list located in the last position in list L . TSAB is stopped if list L is empty. In order to detect and break cycles that can occur during the search, TSAB refers to originally designed *cycle detector*. Boolean function *IsCykle(iter)*, see Step 4, returns *true* at iteration $iter$ if there exists the period δ , $1 \leq \delta \leq max\delta$, such that $C^{i-\delta} = C^i$, $i = iter + 1 - maxc \cdot max\delta, \dots, iter$, where C^i is the makespan found at

iteration i , and $\max\delta$, $\max c$ are given parameters. Parameters $\max c$ and $\max\delta$ are set experimentally as a compromise between the amount of computations and "certainty" of a cycle existence.

5. Advances in TSAB

In this section we provide some advanced, unpublished yet, theoretical elements, which improve only running time of TSAB without any change of its accuracy – the algorithm provides the same solutions but significantly faster. We refer here to two of them: INSA advanced implementation and *accelerators*.

INSA ADVANCED IMPLEMENTATION. In the paper of Nowicki and Smutnicki (1996) we recommended to link TSAB with dedicated starting algorithm INSA, designed with the use of so called *insertion technique*. Many our tests have shown that this combination “fits well”, which means that INSA generates relatively good starting solutions, which enables TSAB to pass quickly to the exploration of promising regions of the solution space. It does not mean, however, that INSA solutions are excellent; the average relative error to the lower bound (ARE) over the benchmarks from Taillard (1993) is about 20%, see column ARE in Table 7.1. Next, these solutions are effectively and significantly improved by TSAB. We made also some tests with other starting algorithms. Poor starting solution, for example random solution, forces TSAB to make an additional number of iterations just before it reaches the makespan value comparable to that obtained by INSA, see Figure 7.6 (right). The final search result does not depend significantly on starting solution, however, the application of INSA allow us to go quicker to good solutions. From Figure 7.6 (right) and 7.7 (left) one can observe that the best solutions found are close in terms of the makespan value, but are distant in the sense of the distance measure. In these figures “distance to the best” means the distance D^4 (see Section 6) to a fixed good reference solution – the best one found during 10,000 iterations of TSAB started from INSA. This good opinion about INSA suggests everybody to increase its speed rather than to modify its fundamental structure. We did so.

An ad hoc implementation of INSA has the computational complexity $O(\delta^2 \max_{k \in M} |M_k|)$, which suggests that it can take a significant part of TSAB running time, particularly for large-size instances. In fact, INSA can be implemented as $O(o^2)$ procedure, which will be shown in the sequel. Thus, its running time is comparable to any standard priority algorithm with dynamic priorities. The implementation with smaller complexity is non-trivial and needs some auxiliary explanations. To this order we remain briefly the INSA structure.

The algorithm constructs the processing order starting from a primal partial order, inserting in each step a single operation, Figure 7.3. In detail, we start from the order containing only all operations of single job, the one with the

Algorithm INSA

INSA returns the approximate processing order $\sigma = (\sigma_1, \dots, \sigma_m)$.

Step 0. Find initial partial processing order σ by scheduling only job 1 having operations $\{1, \dots, l_1\}$.

Step 1. Find permutation ω of the remain operations $\{l_1 + 1, \dots, o\}$ according to non-increasing their processing times.

Step 2. For $s = 1$ to $o - l_1$ perform *Step 3*.

Step 3. For partial processing order σ , operation $x = \omega(s)$ and machine $a = \mu_x$ find $z = |\sigma_a| + 1$ processing orders σ^j , where $\sigma_k^j = \sigma_k$, $k \neq a$, and $\sigma_a^j = (\sigma_a(1), \dots, \sigma_a(j-1), x, \sigma_a(j), \dots, \sigma_a(|\sigma_a|))$, $j = 1, \dots, z$. Find the best order σ' such that $d_x(\sigma') = \min_{1 \leq j \leq z} \{d_x(\sigma^j) : G(\sigma^j) \text{ does not contain a cycle}\}$. Set $\sigma := \sigma'$.

Figure 7.3. INSertion Algorithm (INSA)

greatest sum of processing times (one can assume that this job has index 1). Next, we successively insert operations of remaining jobs in the sequence of non-increasing operation processing times. Denote by $\sigma = (\sigma_1, \dots, \sigma_m)$ the partial processing order, where $\sigma_k = (\sigma_k(1), \dots, \sigma_k(|\sigma_k|))$, $|\sigma_k| \leq |M_k|$ is a partial processing order of operations on k -th machine, $k \in M$ (by analogy to π). For σ we define the analogous partial graph $G(\sigma) = (O, R \cup E(\sigma))$. Let $d_i(\sigma)$ denote the length of the longest path passing through vertex i in $G(\sigma)$. Denote by x the operation which is inserted into σ (obviously x is not in σ) and by $a = \mu_x$ the machine on which inserted operation x should be processed. We generate set of $z = |\sigma_a| + 1$ permutations obtained from σ_a by the insertion of operation x on consecutive positions, i.e., immediately before operation $\sigma_a(i)$, $i = 1, \dots, |\sigma_a|$ and after the operation $\sigma_a(|\sigma_a|)$. This set corresponds to z new partial processing orders σ^j , $j = 1, \dots, z$ on machines, having fixed order σ_k on k -th machine, $k \in M \setminus \{a\}$, and varied on a -th machine. We skip unfeasible processing orders (with cycle in the graph $G(\sigma^j)$) and calculate for each feasible partial solution the length of the longest path $d_x(\sigma^j)$ passing through vertex x ; the processing order which yields the minimum length is taken as the partial processing order for the next step.

Observe that the value $d_x(\sigma^j)$ need not to be calculated expensively as the path in the graph $G(\sigma^j)$ in a time $O(o)$, but can be found in a time $O(1)$ on the base of some paths from $G(\sigma)$. More precisely, let r_i (q_i) denote the length of the longest path in $G(\sigma)$ going to the node i (going out from the node

Table 7.1. Speed and accuracy tests of INSA

instance group	r	m	o	INSA time [s]		speed increase	ARE [%]
				old	new		
ta01 - ta10	15	15	225	0.12	0.01	18.3	14.6
ta11 - ta20	20	15	300	0.26	0.01	25.5	20.7
ta21 - ta30	20	20	400	0.46	0.02	24.8	23.5
ta31 - ta40	30	15	450	0.85	0.03	34.1	21.4
ta41 - ta50	30	20	600	1.48	0.04	34.4	27.9
ta51 - ta60	50	15	750	3.73	0.07	54.0	16.4
ta61 - ta70	50	20	1000	6.56	0.12	54.7	20.1
ta71 - ta80	100	20	2000	64.63	0.51	127.8	13.3

i , respectively), including this node weight. It has been proved (see Appendix A) that

$$d_x(\sigma^j) = \max\{R, r_{\sigma_a(j-1)}\} + p_x + \max\{Q, q_{\sigma_a(j)}\}, \quad (7.1)$$

where $R = r_{x-1}$ if x is not the first operation of a job and $R = 0$ otherwise, $Q = q_{x+1}$ if x is not the last operation of a job and $Q = 0$ otherwise. Moreover, there are no needs to check whether $G(\sigma^j)$ has a cycle. In Appendix B it has been proved the nontrivial fact, that among processing orders σ^j , $j = 1, \dots, z$, checked by INSA, the one which corresponds to the minimal value of right hand-side of expression (7.1) is always feasible. This allows us to replace complex condition “ $d_x(\sigma') = \min_{1 \leq j \leq z} \{d_x(\sigma^j) : G(\sigma^j) \text{ does not contain a cycle}\}$ ” in Step 3 of INSA, see Figure 7.3, by the simple formulae “ $d_x(\sigma') = \min_{1 \leq j \leq z} d_x(\sigma^j)$ ”. Immediately from these two properties we get the smaller computational complexity of INSA, $O(\partial^2)$.

A numerical property of the proposed INSA implementation has been tested experimentally. We provide, in Table 7.1, the average running times per instance (on a PC with Pentium III 900MHz processor), of old and new INSA, for 80 Taillard's benchmarks instances. For the largest group with 2,000 operations (100 jobs, 20 machines) the INSA efficient implementation is more than 100 times faster with respect to the old one.

ACCELERATORS. Accelerators mean the special class of algorithmic components, which allow TSAB to run faster, without any loss of accuracy. From this point of view, INSA efficient implementation is a case of accelerator. Since INSA can also act as an independent algorithm, one can consider this accelerator as not dedicated strictly for TSAB. One of the first accelerators, designed by us and dedicated specially for TSAB, is called Tabu Status accelerator and allows us to check tabu status of a move in a time $O(1)$ instead of $O(\max t)$. Since the length of tabu list in TSAB is small, of order 10, the acceleration of TSAB calculations is unnoticeable, thus we skip its presentation.

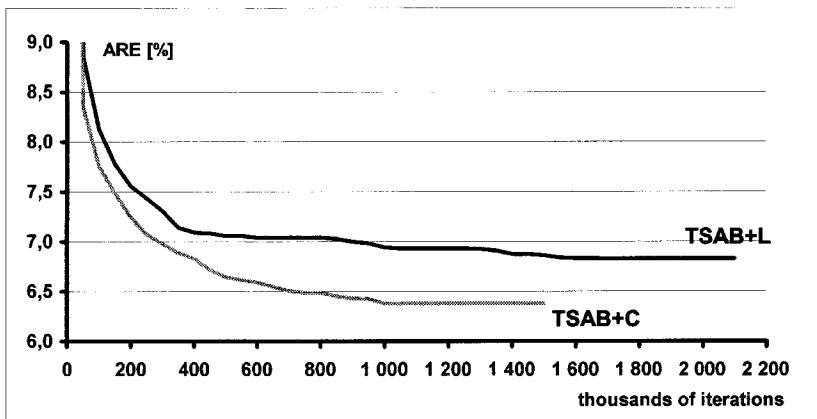


Figure 7.4. Convergence of TSAB+C and TSAB+L on instances ta41 – ta50.

The second, more interesting, accelerator has been designed for the most time consuming part of TSAB. One can check that the very expensive element of TSAB consists in calculation of makespans for neighbors performed during a single neighborhood search - it consumes more than 95% of time. To reduce calculation cost, many authors have proposed to evaluate neighbors by inexpensive lower bound rather than by the makespan. In order to check the impact of such approach on the behavior of TSAB we compared two algorithms, TSAB+C (original) and TSAB+L (equipped with Taillard's lower bound, see for example Eikelder et al. 1999), on Taillard's benchmark instances, assuming $\text{maxiter} = 50,000$ ($\text{maxiter} = 10,000$ if back jump tracking has been performed). Results for the hardest group ta41 – ta50 are shown in Figure 7.4, where the average relative errors to the lower bound (ARE) are drawn in iterations. From Figure 7.4 one can observe that TSAB+C provides better solutions than TSAB+L; ARE for TSAB+C is 6.43%, whereas for TSAB+L is 6.83%. Moreover, TSAB+C needs less iterations than TSAB+L to get the comparable level of makespan value. Unfortunately, time per iteration for TSAB+C is more than ten times longer than that of TSAB+L.

To resolve the speed problem, we designed for TSAB dedicated NSP-Ac accelerator, which modifies classical NSP procedure by using the skilful decomposition and aggregation of makespan calculations performed for the single (whole) neighborhood. NSP-Ac uses advanced analysis of paths in graphs $G(\pi)$, $G(\pi_{(v)})$, its description and formal proof exceed the size of this paper thus, we only direct the reader to the more elaborated paper with NSP-Ac and its numerical studies, Nowicki and Smutnicki, (2001). Application of NSP-Ac

Table 7.2. Behavior of TSAB depending on *maxiter* on instances ta41 – ta50

<i>maxiter</i>	10,000	20,000	40,000	80,000	120,000	160,000
ARE [%]	7.70	6.73	6.29	5.82	5.68	5.64
AV ITER [10^6]	0.32	0.80	1.15	2.07	2.69	3.30
AV CPU [s]	11.1	27.5	39.5	71.4	92.6	114.0

in TSAB+C implies that time per iteration of TSAB+C is only 30% longer than that of TSAB+L. Hence, NSP-Ac constitutes an interesting alternative for the approximate evaluation of neighbors applied, for example, in Eikelder et al. (1999).

The acceleration of TSAB allow us to explore, at the same time, solution space more intensively, for example by the increase of parameter *maxiter*. Computer tests confirmed, that going this way we can really get better solutions. Unfortunately, we frequently observed the stagnation effect for the appropriately large number of iterations. In Table 7.2 we show the behaviour of TSAB on hard instances ta41 – ta50. Algorithm was run for selected $\text{maxiter} = k = 10\,000, 20\,000, \dots, 160\,000$ ($\text{maxiter} = k/2$ if back jump tracking has been performed). AV ITER is the average number of iterations per instance, in millions, with original TSAB stop criterion, see Figure 7.2. AV CPU is the average CPU time per instance on a PC with Pentium III 900MHz processor. One can observe the saturation effect for *maxiter* over 120,000. Although stagnation effect has appeared, obtained results are very good, comparing them to the best other recent studies, Balas and Vazacopoulos, (1998), Pezzella and Merelli, (2000). In the paper Pezzella and Merelli, (2000), ARE for ta41 – ta50 is equal 6.04%; it has been obtained on a PC with Pentium 133MHz processor in a time greater than 11,500 seconds per instance on average. The nearest ARE value 5.82% algorithm TSAB generates already after 71.4 seconds. The ARE value 5.68% generated by TSAB after 92.6 seconds is close to the value 5.71% obtained by the algorithm SB-RGLS10 from Balas and Vazacopoulos (1994), which is the best construction recommended in the paper Balas and Vazacopoulos, (1998) (running time is not given precisely there). Thus, the further improvement of TSAB accuracy we expect in a proper diversification method, which in order needs certain knowledge about the structure of the solution space.

6. Landscapes, Distances, Valleys

This section aim is to analyze topological properties of the solution space of the problem considered.

Landscape means the relation between goal function value and the distance in the solution space. The fundamental aim of the landscape analysis is to detect regularity of the space, especially the *big valley*, which can play the role of an *attractor* in the space. The presence of big valley means that distances between good locally optimal solutions and the global one are significantly positively correlated with goal function values of these solutions. It also means that majority of very good local optima concentrates in a relatively small area of the solution space. “Regularity” is too powerful qualification since combinatorial problems frequently have the huge number of local optima (usually poor), whereas solutions close in the space may have significantly different goal function value. (For example, in Figure 7.6 (left) there is shown makespan value for the sequence of successive solutions generated in first 10,000 iterations of TSAB algorithm.)

Theoretically, the big valley justifies well philosophy of the *path relinking* method, Glover (1997), designed on the basis of the belief, that going along trajectory linking two local optima we are able to find new local optimum or even global one. In practice, solution space may have much more complicated nature. The big valley can also explain amazing efficacy of Back Jump Tracking method used in TSAB.

Existence of a big valley has been experimentally confirmed already for several combinatorial problems, for example, the symmetric TSP, a class of bipartition of nodes in a graph, the permutation flow shop problem with the makespan criterion, see Boe et al. (1994), Reeves (1998). Encouraging conclusions from the landscape analysis inclined us to carry out the similar research for the job shop problem, as the preparing study for path relinking application. In our case, the solution space contains processing orders $\pi \in \Pi$, the goal function is the makespan $C_{\max}(\pi)$.

The distance between processing orders should be adjusted to the algorithm type, in this sense, that the distance denotes the minimal number of moves (those available in the neighborhood) necessary to transform one solution into another. The definition assumes, by default, that such transformation exists, which implies the *strong connectivity* of the neighborhood. Because such analysis would be too difficult to perform and to make a generalization, we made a broader study of measures associated with three types of moves enumerated in Section 3. We define the distance $D^Z(\pi, \sigma)$ between processing orders $\pi, \sigma \in \Pi$ in the natural way, as the minimal number of Z-moves necessary to transform π into σ , $Z \in \{A, S, I\}$. Because each single analyzed move (of any type) relocates operations on single machine (moves are disjoin), we have

$$D^Z(\pi, \sigma) = \sum_{k=1}^m D_k^Z(\pi_k, \sigma_k), \quad (7.2)$$

Table 7.3. Basic properties of distance measures.

move	A	S	I
measures	$D^A(\alpha, \beta)$	$D^S(\alpha, \beta)$	$D^I(\alpha, \beta)$
recipe	number of inversions in $\alpha^{-1} \circ \beta$	n minus the number of cycles in $\alpha^{-1} \circ \beta$	n minus the length of the maximal increasing subsequence in $\alpha^{-1} \circ \beta$
mean	$\frac{n(n-1)}{4}$	$n - H_n$	$n - 2\sqrt{n}$ *)
variance	$\frac{n(n-1)(2n+5)}{72}$	$H_n - H_n^{(2)}$	$\Theta(n^{\frac{1}{3}})$
complexity	$O(n^2)$	$O(n)$	$O(n \log n)$

*) – asymptotically, $H_n = \sum_{i=1}^n \frac{1}{i}$, $H_n^{(2)} = \sum_{i=1}^n \frac{1}{i^2}$

where $D_k^Z(\pi_k, \sigma_k)$ is the minimal number of Z-moves necessary to take a walk between permutations π_k, σ_k associated with the machine k , $k \in M$, $Z \in \{A, S, I\}$.

From (7.2) it follows, that in order to analyze $D^Z(\pi, \sigma)$ there is enough to focus attention on measures $D_k^Z(\pi_k, \sigma_k)$ for a fixed machine k . Moreover, instead of making analyses of these measures on permutations π_k, σ_k defined on the set $M_k \subset \{1, \dots, o\}$, we can carry out analyses on permutations α, β defined on the set $\{1, \dots, n\}$, $n = |M_k|$. Indeed, let f be any mapping function from M_k onto $\{1, \dots, n\}$. From the definition of measures, we have $D_k^Z(\pi_k, \sigma_k) = D_k^Z(\alpha, \beta)$, where $\alpha = f \circ \pi_k, \beta = f \circ \sigma_k$ are permutations on the set $\{1, \dots, n\}$; \circ denotes the superposition of mappings. In the sequel, index k will be skipped for simplicity. Measures $D^Z(\alpha, \beta)$ ($Z \in \{A, Z, I\}$) have been already studied in the literature, see the review in Knuth (1997). Their selected properties, fundamental for our applications, have been collected in Table 7.3 (α^{-1} is the inverse permutation to α).

To check experimentally the existence of big valley we used 50 benchmark instances ta01-ta50 from the library, OR Library. For each instance we generate $1 + c$ local optima by running TSAB from random but feasible initial solution ($c = 200$) and fix the best solution found with its makespan as the reference values (without loss of generality we can assume that the reference solution has index 0). Then we calculate three experimental measures: (1) distance to the reference optimum $X_1(i) = D^Z(\pi^i, \pi^0), i = 1, \dots, c$, (2) the average value of distances to other local optima $X_2(i) = \frac{1}{c} \sum_{j=1; j \neq i}^c D^Z(\pi^i, \pi^j), i = 0, \dots, c$, (3) difference to reference makespan $Y(i) = C_{\max}(\pi^i) - C_{\max}(\pi^0), i = 0, \dots, c$; $Z \in \{A, I\}$. For each combination $X_1(i), Y(i), i = 1, \dots, c$ and $X_2(i), Y(i)$,

Table 7.4. Distribution of local minima for various metrics.

instance	A-moves				I-moves			
	Average		k	ρ_{av}	Average		k	ρ_{av}
	ρ_{max}	ρ_{av}			ρ_{max}	ρ_{av}		
ta01 – ta10	0.64	0.58	8	9	0.65	0.62	9	9
ta11 – ta20	0.58	0.53	10	10	0.53	0.57	10	10
ta21 – ta30	0.57	0.65	10	10	0.53	0.67	10	10
ta31 – ta40	0.51	0.61	9	10	0.46	0.67	10	10
ta41 – ta50	0.52	0.65	10	10	0.48	0.68	10	10
all	0.56	0.60	47	49	0.53	0.64	49	49

$i = 0, \dots, c$ the correlation coefficient has been found, denoted next by ρ_{max} and ρ_{av} . For each of five groups, we found the average values from sample of ten values of ρ_{max} and ρ_{av} , respectively. Significance of the correlation has been checked with the help of a randomized statistical test (standard tests cannot be applied because of unfulfilled assumptions), Manly (1991). Results are given in Table 7.4, where k denotes the number of instances (among sample of 10) for which correlation coefficient is statistically significant on the level 0.1%.

From the second column of Table 7.4 it follows that there exist essential correlation between the distance (measured from local to the best-known optimum) and makespan value; less distance implies less makespan. Values of correlation coefficients are statistically significant on the level 0.1% for 47 among 50 tested instances, which confirms this thesis. The similar observations one can perform for column 3. Surprisingly, the choice of measure has no meaning for conclusions made.

To show this relation in more accurate way, we plot points $X_1(i), Y(i)$, $i = 1, \dots, c$, Figure 7.5 (left) and points $X_2(i), Y(i)$, $i = 0, \dots, c$, Figure 7.5 (right) for the instance ta45 ($r = 30$ jobs, $m = 20$ machines, $|M_k| = r$, $k \in M$) and the measure D^A . The most distant local optimum found in the experiment is 1454 far from the reference solution (π^0). Theoretically, the average value between any solution and π^0 is 4350, the standard deviation is 125.3, see Table 7.3. Since the distribution of measure D^A is strongly concentrated around its average value, the area with solutions far less than 1454 from π^0 with detected local minima, constitutes only a very small part of the solution space ((4350–1454)/125.3=23.1 standard deviations). Moreover, Figure 7.5 (right) suggests that best solution (π^0) is located in the center of local optima. Analogous conclusions have been made for other instances, so one can incline to hypothesis that big valley typically exists.

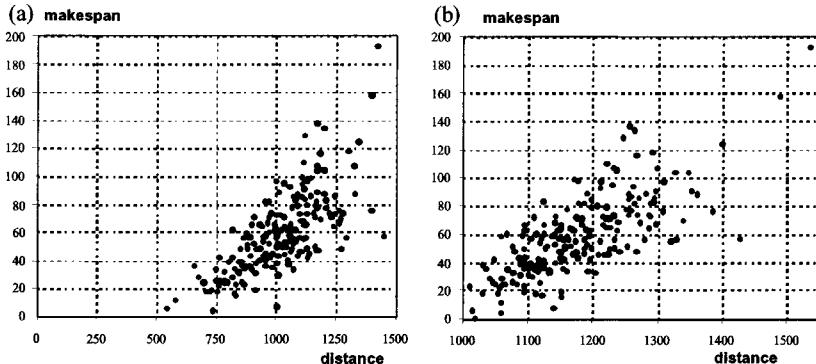


Figure 7.5. Quality of local minima for the instance ta45: (a) distance to the best-known minimum, (b) the average distance to other minima.

7. Intuitions, Analogies, Shocks

This section aim is to discuss strategy aspects of the search technique, referring to some surprising analogies.

The first shock was already happened – Algorithm TSAB, even though ascetic in its structure, was able to break the hard optimization problem like the job shop. The second shock follows from the surprising fact shown in the previous section – the attractive valley is of very small size but attracts TSAB well.

The third shock follows from an approximate analysis of sizes of the solution space and searched areas. We would like to discuss the possibility of making the projection of the solution space on a 2D surface. Let us take the oldest historically instance FT10 (10 jobs, 10 machines, 100 operations), one of the smallest now. Because this is a hypothetical analysis, the used 2D transformation need not be specified here – we only assume that it preserves the distance, i.e. distant solutions in the space correspond to distant points on 2D plane and close solutions to close points, at least in statistical sense.

Among few known representation of solutions (disjunctive graph, sequence of permutations, single permutation) the representation used by us has the smallest redundancy, although enables unfeasible solutions. For FT10 we have $\prod_{i=1}^m (|M_i|!) \approx 4 \cdot 10^{65}$ solutions, including unfeasible ones. Evaluation of the fraction of feasible solutions generally can be a problem, because the result strongly depends on the job structures. Indeed, assuming identical job routes we get a special case well known as non-permutation flow shop problem, where all solutions are feasible. Thus fraction can be extremely equal to 1. On the other hand, for FT10 with its original job structure, we are failed in finding any feasible solution in the sample of 10^9 (billion) random solutions. Therefore,

we only approximated this fraction using an auxiliary test. To this order we consider a sequence of relaxed instances obtained from FT10, each of which has the original job routes, the same number of machines, and the number of jobs equal $r = 2, 3, 4, \dots$. By random sampling, we found these fractions equal approximately $1.5 \cdot 10^{-1}$, $3.2 \cdot 10^{-3}$, $3.2 \cdot 10^{-5}$, $2.6 \cdot 10^{-7}$, $2.8 \cdot 10^{-9}$, for $r = 2, 3, 4, 5, 6$, respectively. These results allow us to approximate fraction for the whole FT10 on the level 10^{-17} .

Next, the remain $4 \cdot 10^{48}$ feasible solution we transform one-to-one on 2D plane, single solution into single point, setting its altitude equal to the makespan value. Finally, we print figure using 2400dpi printer, in which single black dot is of size 0.01×0.01 mm. Hence, we will obtain completely filled surface, without blank spots, which illustrates the landscape and has area of order $4 \cdot 10^{32}$ km². For comparison the biggest known planet Jupiter has only $6.4 \cdot 10^{10}$ km².

Further observations we will make using analogy to the Earth surface. Suppose that our aim is to reach the highest Himalayas top, starting from certain place on the Earth and walking or jumping across the surface. In fact, “jump” is the better analogy since from properties of, for example, measure D^4 it follows, that any place on the Earth surface can be reached in at most $m^{\frac{r(r-1)}{2}} = 450$ steps. Generally, information about regularity of the surface is necessary to find the proper orientation. However, if only we can start the track close to Himalayas, it is undoubtedly advantageous. Note, that after we will found Himalayas we will need a great additional effort to select the highest top. Although Himalayas is unique, big and promising, there are relatively small in the Earth scale. It means that the path between two selected places need not go through it or even pass by these mountains. Search by random sampling and quasi-stochastic method without any special problem properties seems to be too poor. The shock becomes evident when we'll try to draw a local search trajectory. It takes the form of almost invisible spider web (0.01 mm width) covering very slightly very small surface parts. Indeed, 10^9 (billion) solutions generated during the search covers only 0.1 m². Such pure intuitive interpretations allow us to formulate a few hypotheses. Starting from a solution, it is much easier to generate a search trajectory without cycle than with a cycle; the latter case is simply bad luck. Single good starting point is much better than several worse starting points. Valley may be of too small size to be detected by relinking paths. Single, even good, solution found on the relinking path can be useless for the accuracy improvement, its desirable fundamental aim should be to provide alternatively certain reasonable diverse starting point for the new search process. All these facts agree with intuitions “it is not enough to search, one should know where to search”.

To verify some of stated hypotheses we made few experimental tests. TSAB was equipped with memory, which registered all visited solutions and frequency of its visitation. To make the implementation efficient we used the hashing table

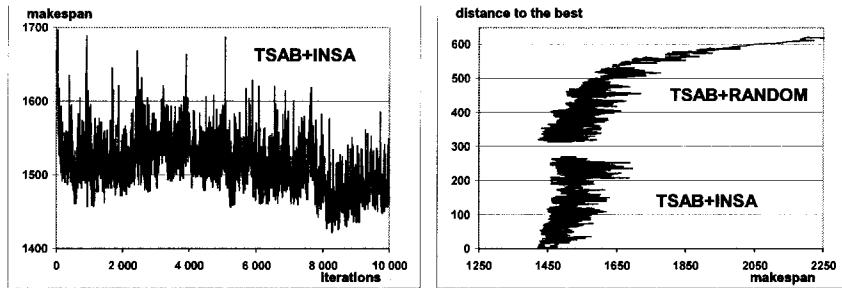


Figure 7.6. Instance ta13: 10,000 first iterations of TSAB started from INSA and RANDOM in various axis.

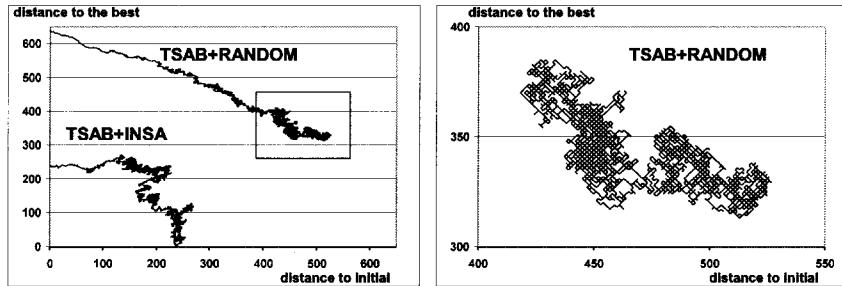


Figure 7.7. Instance ta13: distances to the best-known solution during 10,000 first iterations of TSAB with INSA and RANDOM (left), zoom of second 5,000 iterations (right).

technique, derived from Woodruff and Zemel (1993), with hashing function $h(\pi) = \sum_{i=1}^m \sum_{j=1}^{|\pi_i|} z_{ij} \pi_i(j) \bmod h$ and disjoint lists to resolve collisions. Here, z_{ij} are pre-defined random numbers, h is the size of hashing table and \bmod is operator modulo. Each visited solution π was stored as the list element $(C_{\max}(\pi), \pi, c(\pi))$, where $c(\pi)$ is counter of repetitions. TSAB was run on 100,000 iterations with $maxl = 0$ on a few selected Taillard's benchmarks. Generally, two cases have been observed: (a) *Cycle detector* does not indicate the occurrence of the cycle (b) *Cycle detector* indicates the cycle occurrence. In case (a) an infinitesimal number of repetitions (below 0.01%) of accidental solutions have been observed, which is fully justified, since TS does not forbid repetitions of solutions but strongly prohibits repetitions of the same fragments of the search trajectory (cycling). Case (b) has been observed very rarely, frequency of its occurrence decreases with the increase of parameter $maxt$.

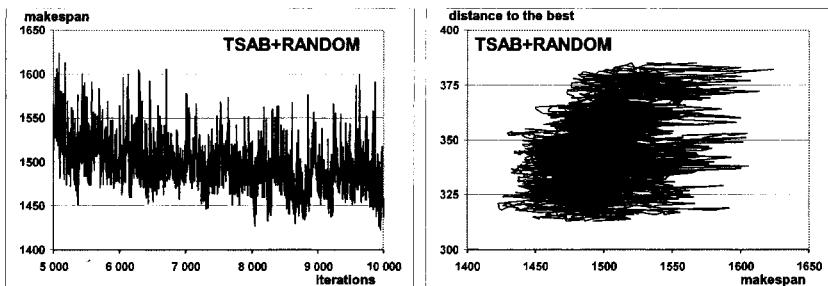


Figure 7.8. Instance ta13: zoom of second 5,000 iterations of TSAB+RANDOM in various axis.

The similar behavior of TSAB has been observed if $maxI > 0$. This inclines us to the thesis that TSAB searches disjoint areas of the solution space.

Figure 7.7 (left) shows that searched area are also distant, although, from time to time, TSAB focuses on a narrow (small) part of the space, performing intensive search of this region, see distinguished part of the search trajectory in Figure 7.7 (left) zoomed in Figure 7.7 (right). Figure 7.8 shows that TSAB does not wonder around because the best known makespan still decreases; TSAB simply visits a local, relatively small sub-valley. In these figures the “distance to the best” means the distance D^A to a fixed reference solution – the best found during 10,000 iterations of TSAB started from INSA.

8. Paths, Goals, Viewpoints

In two successive sections we show a concept of search diversification for TSAB. It is based on so called s-step goal oriented path and milestones, which mark out centers of local search areas.

Let we have two different, feasible processing orders π and σ , the former called *source* and the latter *target*. We would like to generate the completely deterministic, oriented *path* in the solution space from π towards σ . As the distance measure D we take $D^A(\pi, \sigma)$, the proper one for neighborhoods based on A-moves. The neighborhood $N^{A2}(\pi)$ employed originally in TSAB, does not have connectivity, therefore should not be applied to this aim. We use $N^{A1}(\pi)$, similar in the structure and type of moves performed, having connectivity property. Note that, $N^{A1}(\pi)$ has the advantageous small-step property

$$|D^A(\rho_{(v)}, \sigma) - D^A(\rho, \sigma)| = 1 \quad (7.3)$$

for any $v \in V^{A1}(\rho)$, $\rho \in \Pi$. We introduce *s-step goal oriented path* from π towards σ (s-GOP) as a sequence of feasible processing orders $\pi = \pi^0, \pi^1, \dots, \pi^s$,

which starts from π^0 and chooses at each step i ($i = 0, \dots, s - 1$) the processing order $\pi^{i+1} = \pi_{(w)}^i$ for some $w \in V^{A1}(\pi^i)$. Selection of move w is being made with the help of two criteria. The former prefers these moves which decrease the distance to σ , whereas the latter – those leading to the processing orders with the least makespan. Precisely, formulated conditions can be written as follows

$$C_{\max}(\pi_{(w)}^i) = \min\{C_{\max}(\pi_{(v)}^i) : v \in V^{A1}(\pi^i), D^A(\pi_{(v)}^i, \sigma) = d_{\min}^i\} \quad (7.4)$$

where $d_{\min}^i = \min\{D^A(\pi_{(v)}^i, \sigma) : v \in V^{A1}(\pi^i)\}$ is the minimal distance reachable from $V^{A1}(\pi^i)$ towards σ . From (7.3), we have $d_{\min}^i \in \{D^A(\pi^i, \sigma) - 1, D^A(\pi^i, \sigma) + 1\}$. The minimal number s under which σ is reachable (i.e. $\pi^s = \sigma$) equals $D^A(\pi, \sigma)$; this occurs only if for each step i there exist in $V^{A1}(\pi^i)$ at least one move that decreases the distance to σ . Theoretically, the desired move need not exist, thus σ can be reachable in more than $D^A(\pi, \sigma)$ steps or can be unreachable (a cycle occurs). In practice, the desired move can be found in the neighborhood $V^{A1}(\pi^i)$ almost every time, which means that $D^A(\pi^i, \sigma) = D^A(\pi, \sigma) - i$.

s-GOP concept is close to the *path relinking* philosophy, Glover (1997), however, in our application, its fundamental role consists in providing disperse solutions. Each solution from s-GOP we call the *viewpoint* on the route. s-GOP is a special case of the search trajectory, hence the best-known solution can be updated along this path. Nevertheless, this is not its basic aim and occurs very rarely in practice. We would like to find a viewpoint, heedless of its makespan, relatively distant from both the source and target. Therefore, we propose to use the *terminal viewpoint* (π^s) from s-GOP as the starting solution for more intensive space penetration, made next by TSAB. We propose to generate s-GOP with $s = \text{maxd } D^A(\pi, \sigma)$, where $0 < \text{maxd} < 1$ is a control parameter. If we have no preferences, we can set this value to 0.5, which means that the terminal viewpoint is placed approximately in the middle between π and σ .

9. Diverse Search

Broad solution space with the chaotic structure justifies well a dispersion in the search process. Even though valley exists, is of very small size and attracts ideally TSAB in the search process – it still contains a huge number of solutions, has local sub-valleys and deception local extremes. Thus, any intelligent search process is especially welcome.

Let we have the subset of $e + 1$ elite solutions $P = \{\pi^0, \pi^1, \dots, \pi^e\}$ called next *milestones*. Milestones mark out local centers of search areas. Without loss of generality one can assume that π^0 has the smallest makespan and represent the best-known up to now solution. Presuming that big valley exists and elite solutions follow from this valley, we restrict the search area roughly to the

hypersphere with center in π^0 and radius $\max_{1 \leq j \leq e} D^A(\pi^j, \pi^0)$. Our aim is to move step-by-step milestones toward the center, to lessen the search area. We realize this scenario by replacing the most distant solution from the center of P by a new promising solution found during the auxiliary run of TSAB from alternative starting solution. More precisely we perform as follows. First, we find the most distant solution π^k with respect to the center, i.e. such that $D^A(\pi^k, \pi^0) = \max_{1 \leq j \leq e} D^A(\pi^j, \pi^0)$. Next, using s-GOP with source π^0 and target π^k we generate a terminal viewpoint, denoted by $\pi^{0:k}$, which is the starting point for TSAB. The best solution $\pi^{0:k,*}$ found during this run of TSAB replaces (unconditionally) target solution π^k in P . Theoretically, newly obtained solution $\pi^{0:k,*}$ can be worse and/or more distant from π^0 than π^k , however, in computer tests, we observe convergence of the hypersphere radius to zero. Process finishes when the radius of the hypersphere is less than the given threshold value $maxR$.

A small problem exists with primal milestones, since they should be dispersed and located relatively close to big valley. To this aim we use the special procedure, which creates with the help of TSAB, step-by-step, the initial set P . We begin with $P = \{\pi^0, \pi^1\}$, where π^0 is provided by INSA, whereas π^1 is the best solution found during the run of TSAB started from π^0 . Let i denotes the index of last milestones introduced to P , $i = 1, \dots, e - 1$. The milestone $i + 1$ -st is the best result of TSAB started from the terminal viewpoint found on s-GOP from π^i towards π^0 . Process of the milestone generation is repeated until $|P| = 1 + e$. We checked experimentally that distances between all pairs of primal milestones in P are almost equal.

10. Preliminary Computational Results

By embedding TSAB in the structure MILESTONES of diverse search with technique of s-GOPs we obtain more complex algorithmic structure called next i -TSAB. Algorithm i -TSAB has been implemented in Delphi and run on a PC with Pentium III 900MHz processor. Algorithm i -TSAB has several own tuning parameters, besides those derived from TSAB, namely e (the number of milestones), $maxd$ (location of viewpoint), $maxR$ (minimal radius of hypersphere). It is clear that their proper selection influences on final numerical properties of i -TSAB and requires broad computational studies.

Nevertheless, before the precise study of i -TSAB properties, we decided to check whether the proposed ideas are really promising enough. To this order, we set running parameters in an approximate way, without precise tuning experiments, and then we run i -TSAB on the benchmark set, Taillard (1993). These instances has been attacked, since 1993, by all job-shop algorithms designed over the world, frequently not seeing on the running time. The best known results follow from many various studies and have been obtained during long-time

Table 7.5. Improved reference makespans for Taillard's instances

number	$o \times r \times m$	instance	LB – UB	new UB	decrease
1	$300 \times 20 \times 15$	ta11	1323 – 1364	1361	3
2		ta13	1282 – 1349	1345	4
3		ta19	1297 – 1341	1335	6
4		ta20	1318 – 1353	1351	2
5	$400 \times 20 \times 20$	ta22	1511 – 1601	1600	1
6		ta23	1472 – 1558	1557	1
7		ta24	1602 – 1651	1648	3
8		ta27	1616 – 1687	1680	7
9		ta30	1473 – 1585	1584	1
10	$450 \times 30 \times 15$	ta32	1774 – 1803	1798	5
11		ta34	1828 – 1832	1829	3
12		ta37	1771 – 1784	1779	5
13		ta40	1631 – 1686	1674	12
14	$600 \times 30 \times 20$	ta41	1859 – 2023	2021	2
15		ta42	1867 – 1961	1956	5
16		ta43	1809 – 1879	1870	9
17		ta44	1927 – 1998	1992	6
18		ta45	1997 – 2005	2000	5
19		ta46	1940 – 2029	2025	4
20		ta47	1789 – 1913	1911	2
21		ta48	1912 – 1971	1963	8
22		ta49	1915 – 1984	1973	11
23		ta50	1807 – 1937	1928	9

tests, which means that most of instances were run several hours or even days for many variants of each algorithm, Home page of Eric Taillard. Although a great effort has been done, 35 among 80 instances still remain unsolved (the optimal solution has not been found).

In this preliminary test we set the following values of parameters: $maxd = 0.5$, $maxR = 5$. Algorithm TSAB, a component of *i*-TSAB, run with NSP-Ac and parameters $maxt = 8$, $maxl = 5$, $max\delta = 100$, $maxc = 2$ set as recommended in Nowicki and Smutnicki (1996). Parameter *maxiter* was adjusted to a more intensive search by setting *maxiter* = 50,000 if improvement of the makespan has occurred and *maxiter* = 10,000 if back jump tracking has been performed. Each run of *i*-TSAB was limited to 10 min per instance. Runs were repeated for a few selected values of parameter e , $5 \leq e \leq 16$. In this test *i*-TSAB found 23 better upper bounds among 35 unsolved yet ta- instances, see Table 7.5. Note, it improved upper bounds for all 10 instances from the unsolved hardest group ta41 – ta50 with $o = 600$ operations.

Although exhaustive computer tests have been completed, presentation of these results exceeds the size of this study – they are provided in separate paper, Nowicki and Smutnicki, (2001). Based on these research we mention here only that *i*-TSAB is competitive for the best known job-shop algorithms, taking account of the speed as well as the accuracy. For the mentioned already hardest group of instances ta41 – ta50, algorithm *i*-TSAB run one time per instance on 10 minutes with $e = 8$ provides ARE equal 4.88%, which is significantly better than the result 5.71% from the paper Balas and Vazacopoulos, (1998) and 6.04% from the paper Pezzella and Merelli, (2000), already mentioned in Section 5. It is clear that the proposed approach can be applied also to other scheduling problems.

Research was supported by Grant T11A 01624 of the State Committee for Scientific Research executed in 2003-2006.

Appendix: A. Fast evaluation of $d_x(\sigma^j)$ in INSA

This section aim is to prove the equality (7.1). Assume that $G(\sigma^j)$ does not contain a cycle. Analyze the differences between graphs $G(\sigma^j)$ and $G(\sigma)$. Trial insertion of operation x between $\sigma_a(j-1)$ and $\sigma_a(j)$, $a = \mu_x$, does not change the length of longest path going to the node $\sigma_a(j-1)$ and to the node $x-1$ (if x has a job predecessor, i.e. is not the first operation of the job). This insertion does not change also the length of longest path going out of node $\sigma_a(j)$ and $x+1$ (if x has a job successor, i.e. is not the last operation of the job). Since $\sigma_a(j-1)$ becomes the machine-predecessor of operation x and $\sigma_a(j)$ becomes the machine-successor of x , the path $d_x(\sigma^j)$ passing through the node x takes the form given in right hand-side of formula (7.1). This completes the proof.

Appendix: B. Fast checking of feasibility in INSA

This section aim is to prove that among partial processing orders σ^j , $j = 1, \dots, z$, $z = |\sigma_a| + 1$, generated by INSA for operation x , the one which minimizes right-hand of formula (7.1) provides feasible processing order. Proof consists of two parts: (A) we show that among σ^j , $j = 1, \dots, z$, exists at least one feasible processing order, (B) we show that the advisable insertion ensures feasibility.

Part A. Inserted operation x follows from the machine $a = \mu_x$, thus we define two positions for the partial permutation σ_a

$$e = \max\{1 \leq i \leq |\sigma_a|; \text{ exists path from } \sigma_a(i) \text{ to } x \text{ in } G(\sigma)\} \quad (7.B.1)$$

$$f = \min\{1 \leq i \leq |\sigma_a|; \text{ exists path from } x \text{ to } \sigma_a(i) \text{ in } G(\sigma)\} \quad (7.B.2)$$

If sets in (7.B.1) or (7.B.2) are empty, we set $e = 0$ or $f = |\sigma_a| + 1$, respectively. Since $G(\sigma)$ does not contain a cycle, we have $e < f$. Insertions of operation

x in positions $e < j \leq f$ in σ_a provides graph without a cycle. Indeed, graph $G(\sigma^j)$ can be obtained from $G(\sigma)$ by adding at most two arcs $(\sigma_a(j-1), x)$ and $(x, \sigma_a(j))$. Thus, cycle in the graph $G(\sigma^j)$ may have one of two forms: (a) it goes from a node $\sigma_a(j-1)$ to node x by the newly introduced arc $(\sigma_a(j-1), x)$ and next comes back from the node x to $\sigma_a(j-1)$ by some path, which existed already in $G(\sigma)$, (b) it goes from a node x to node $\sigma_a(j)$ by the newly introduced arc $(x, \sigma_a(j))$ and next comes back from the node $\sigma_a(j)$ to x by some path, which existed already in $G(\sigma)$. From definitions (7.B.1) and (7.B.2), required paths do not exist in $G(\sigma)$. It completes the proof of this part.

Part B. For the operation x , inserted in position j , $1 \leq j \leq z$, in permutation σ_a denote the right hand-side part of formula (7.1) by

$$L_j = \max\{R, r_{\sigma_a(j-1)}\} + p_x + \max\{Q, q_{\sigma_a(j)}\}. \quad (7.B.3)$$

The proof will be made by contradiction. Assume that L_h , for some $1 \leq h \leq z$, is the minimal but $G(\sigma^h)$ has a cycle. Thus, it has to be $0 < h \leq e$ or $f < h \leq z$. Consider both cases separately.

“Case: $0 < h \leq e$.” From the definition of e , there exist in $G(\sigma)$ a path W from $\sigma_a(e)$ to x . Then, we have

$$R \geq r_{\sigma_a(e)}, \quad q_{\sigma_a(h)} > \max\{Q, q_{\sigma_a(e+1)}\}. \quad (7.B.4)$$

The first inequality follows from the fact that path W must go through node $x-1$ (because operation x in $G(\sigma)$ is not inserted in σ_a yet), whereas the second is obvious. From (7.B.4) we obtain $L_h \geq R + p_x + q_{\sigma_a(h)} > \max\{R, r_{\sigma_a(e)}\} + p_x + \max\{Q, q_{\sigma_a(e+1)}\} = L_{e+1}$ which contradicts that L_h is minimal.

“Case: $f < h \leq z$.” This case can be proved by analogy.

Both cases contradict assumption $0 < h \leq e$ or $f < h \leq z$, thus it has to be $e < h \leq f$, which means that $G(\sigma^h)$ does not contain a cycle. This completes the proof.

References

- Balas, E. (1969) "Machine Sequencing via Disjunctive Graphs. An Implicit Enumeration Algorithm," *Operations Research*, 17:941-957.
- Balas, E. and A. Vazacopoulos (1998) "Guided Local Search with Shifting Bottleneck for Job-Shop Scheduling". *Management Science*, 44:262-275.
- Balas, E. and A. Vazacopoulos (1994) Guided Local Search with Shifting Bottleneck for Job-Shop Scheduling. Technical Report MSRR-609, GSIA, Carnegie Mellon University, Pittsburg.
- Blazewicz, J., W. Domschke and E. Pesch (1996) "The Job Shop Scheduling Problem. Conventional and new Solution Techniques," *European Journal of Operational Research*, 93:1-33.

- Boese, K., A. Kahng and S. Muddu (1994) "A new Adaptive Multi-Start Technique for Combinatorial Global Optimizations," *Operations Research Letters*, 16:101-113.
- Brucker, P. (1996) *Scheduling Algorithms*. Springer-Verlag, Berlin.
- Cheng, R., M. Gen and Y. Tsujimura (1996) "A Tutorial Survey of Job-Shop Scheduling Problems using Genetic Algorithms, Part II: hybrid genetic search strategies," *Computers & Industrial Engineering*, 36:343-364.
- DellAmico, M. and M. Trubian (1993) "Applying Tabu Search to the Job-Shop Scheduling Problem," *Annals of Operations Research*, 4:231-252.
- Ten Eikelder, H.M.M., B.J.M. Aarts, M.G.A. Verhoeven and E.H.L. Aarts (1999) Sequential and Parallel Local Search Algorithms for Job-Shop Scheduling. In S. Voß, S. Martello, I.H. Osman, and C. Roucairol, Editors, *Meta-Heuristic: Advances and Trends in Local Search Paradigms for Optimization*. Kluwer Academic Publishers, Massachusetts, USA.
- Glover, F. and M. Laguna (1997) *Tabu search*. Kluwer Academic Publishers. Massachusetts, USA.
- Glover, F. (1989) "Tabu search. Part I," *ORSA Journal of Computing*, 1:190-206.
- Glover, F. (1990) "Tabu search. Part II," *ORSA Journal of Computing*, 2:4-32.
- Grabowski, J., E. Nowicki and C. Smutnicki (1988) Block Algorithm for Scheduling of Operations in Job-Shop System, (in Polish). *Przeglad Statystyczny*, 35:67-80.
- Jain, A.S., B. Rangaswamy and S. Meeran (2000) "New and "stronger" Job-Shop Neighbourhoods: A Focus on the Method of Nowicki and Smutnicki," *Journal of Heuristics*, 6(4):457-480.
- Jain, A.S. and S. Meeran (1999) "Deterministic Job-Shop Scheduling: Past, Present and Future," *European Journal of Operational Research*, 113:390-434.
- Knuth, D.E. (1997) *The Art of Computer Programming*. Addison Wesley Longman.
- Laarhoven, P.V., E. Aarts and J.K. Lenstra (1992) "Job-Shop Scheduling by Simulated Annealing," *Operations Research*, 40:113-125.
- Manly, B.F.J. (1991) *Randomization and Monte Carlo Methods in biology*. Chapman and Hall, London.
- Nowicki, E. and C. Smutnicki (1996) "A Fast Tabu Search Algorithm for the Job-Shop Problem," *Management Science*, 42(6):797-813.
- Nowicki, E. and C. Smutnicki (2001) New Tools to Solve the Job-Shop Problem. Technical Report 51/2001 (extended version in Technical Report 60/2002). Institute of Engineering Cybernetics, Wroclaw University of Technology, Wroclaw, Poland, <http://www.zsd.ict.pwr.wroc.pl>
- OR Library. <http://mscmga.ms.ic.ac.uk/info.html>.

- Pezzella, F. and E. Merelli (2000) "A Tabu Search Method Guided by Shifting Bottleneck for the Job-Shop Scheduling Problem," *European Journal of Operational Research*, 120:297-310.
- Reeves, C. (1998) Landscapes, Operators and Heuristic Search. Technical Report, School of MIS, Coventry University.
- Home page of E. Taillard. <http://www.eivd.ch/ina/collaborateurs/etd/default.htm>
- Taillard, E. (1993) "Benchmarks for Basic Scheduling Problems," *European Journal of Operational Research*, 64:278-285.
- Vaessens, R., E. Aarts and J.K. Lenstra (1996) "Job Shop Scheduling by Local Search," *INFORMS Journal of Computing*, 8:303-317.
- Woodruff, D.L. and E. Zemel (1993) "Hashing Vectors for Tabu Search," *Annals of Operations Research*, 41:123-137.

Chapter 8

A Tabu Search Heuristic for the Uncapacitated Facility Location Problem

Minghe Sun

*College of Business, The University of Texas at San Antonio, San Antonio, TX 78249, USA,
msun@lonestar.utsa.edu*

Abstract: A tabu search heuristic procedure is developed to solve the uncapacitated facility location problem. The heuristic procedure uses tabu search to guide the solution process when evolving from one solution to another in order to search for an optimal solution. A move is defined to be the closing or opening of a facility. The net change in the total cost resulting from a move is used to measure the attractiveness of a move. Searching only a small subset of the feasible solutions that contains the optimal solution, the procedure is computationally very efficient. A computational experiment is conducted to test the performance of the procedure and computational results are reported. The procedure can easily find optimal solutions for test problems with known optimal solutions from the literature. Solutions obtained with this tabu search procedure completely dominate those obtained with the Lagrangian method for randomly generated test problems.

Keywords: Facility Location, Tabu Search, Heuristics, Optimization

1. Introduction

Due to their strategic nature, facility location problems have attracted the attention of researchers and practitioners for many years (Chan, 2000; Daskin, 1995; Mirchandani and Francis, 1990). The success or failure of businesses and public facilities depends in a large part on their locations. Effective supply chain management has led to increased profit and increased market share for many companies. One strategic decision in supply chain management is facility location (Simchi-Levi, Kaminsky and Simchi-Levi, 2000). Because of the variety of facility location problems, there are a variety of facility location models to represent these problems (Brandeau and

Chiu, 1989; Chhajed, Francis and Lowe, 1993; Daskin, 1995; Ghosh and Harche, 1993; Krarup and Pruzan, 1983, 1990). Most of these problems are combinatorial in nature and are hard to solve. Exact algorithms exist only for small problems and heuristic procedures have to be used for large practical problems. This study addresses the uncapacitated facility location (UFL) problem by developing a tabu search heuristic procedure to solve it. In this problem, a fixed cost is associated with the establishment of each facility and another fixed cost is associated with the opening and using of each road from a customer to a facility.

This heuristic procedure employs the short term and long term memory processes of the tabu search meta-heuristic (Glover, 1989, 1990a, 1990b; Glover and Laguna, 1997; Glover, Taillard and de Werra 1993) and searches only a small subset of the feasible solutions that contains the optimal solution. Therefore, the procedure is computationally very efficient. The performance of the procedure is tested through a computational experiment using both test problems from the literature and test problems randomly generated. Computational results show that the tabu search procedure can find optimal solutions for all test problems with known optimal solutions from the literature. With randomly generated test problems, the tabu search procedure can find solutions better than those found by the Lagrangian method without any exception though using slightly more CPU time.

The rest of this chapter is organized as follows. The UFL problem is briefly discussed in Section 2. The tabu search heuristic procedure is developed in Section 3. Computational results are reported in Section 4. Conclusions and further remarks are given in Section 5.

2. The Uncapacitated Facility Location Problem

A UFL problem with m customer locations and n candidate facility location sites can be represented by a network with a total of $m + n$ nodes and mn arcs (Cornuéjols, Nemhauser and Wolsey, 1990; Daskin, 1995). Among the $m + n$ nodes, m of them represent the m customers and the other n represent the n candidate facility location sites. In the UFL model, f_j is used to represent the cost of opening a facility at site j and c_{ij} is used to represent the cost of serving customer i from facility site j or the cost of assigning customer i to facility site j . We assume that $c_{ij} \geq 0$ for all $i = 1, \dots, m$ and $j = 1, \dots, n$ and $f_j > 0$ for all $j = 1, \dots, n$. A binary decision variable y_j is associated with each facility j in the UFL model. Facility site j will be open only if $y_j = 1$ in the solution. A binary decision variable x_{ij}

is associated with the road from customer i to facility j in the model. Customer i will be served by facility j only if $x_{ij} = 1$ in the solution. However, x_{ij} can be treated as a continuous variable in the model and will have a binary value in the solution. The solution process of the UFL problem is to find an optimal solution that satisfies all customer demand and minimizes the total cost.

The UFL problem can be formally stated as (Cornuéjols, Nemhauser and Wolsey, 1990)

$$\min \quad \sum_{i=1}^m \sum_{j=1}^n c_{ij} x_{ij} + \sum_{j=1}^n f_j y_j \quad (1)$$

$$s.t. \quad \sum_{j=1}^n x_{ij} = 1 \quad \text{for } i = 1, \dots, m \quad (2)$$

$$x_{ij} \leq y_j \quad \text{for } i = 1, \dots, m \text{ and } j = 1, \dots, n \quad (3)$$

$$x_{ij} \geq 0 \quad \text{for } i = 1, \dots, m \text{ and } j = 1, \dots, n \quad (4)$$

$$y_j = 0 \text{ or } 1 \quad \text{for } j = 1, \dots, n. \quad (5)$$

In the literature, the UFL problem is also called the simple facility location problem, the simple (or uncapacitated) warehouse location problem and the simple (or uncapacitated) plant location problem. Many successful applications of the UFL model have been reported. Many practical problems without facilities to locate can also be modeled as UFL problems, such as cluster analysis, machine scheduling, economic lot sizing and portfolio management problems (Cornuéjols, Nemhauser and Wolsey, 1990). A real life application of the UFL model is Star LAN design in information technology (Gen, Tsujimura and Ishizaki, 1996). In this application, workstations and terminals are connected to servers. In the design process, location sites for the servers need to be selected so that the total cost of installing the servers and connecting the workstations and terminals is minimized.

Developing solution methods for the UFL problem has been a hot topic of research for the last 40 years. Kuehn and Hamburger (1963) developed the first heuristic method that has two phases. The first phase is a greedy method, called the ADD method, that starts with all facilities closed, keeps

adding (opening) the facility that gives the maximum decrease in the total cost, and stops if adding one more facility will no longer reduce the total cost. The second phase is a local search method in which an open facility and a closed facility are interchanged as long as such an interchange reduces the total cost. Another greedy heuristic is the DROP method that starts with all facilities open, keeps dropping (closing) the facility that gives the maximum decrease in the total cost, and stops if dropping one more facility will no longer reduce the total cost (Cornuéjols, Fisher and Wolsey, 1977; Daskin, 1995; Nemhauser, Wolsey and Fisher, 1978). These early heuristics provided the basis for many more sophisticated heuristic methods and provided an initial incumbent for many exact solution algorithms (Cornuéjols, Fisher and Wolsey, 1977; Nemhauser, Wolsey and Fisher, 1978). Erlenkotter (1978) developed a dual approach for the UFL problem. Although this dual method is an exact algorithm, it can also be used as a heuristic method to find good solutions. One effective and widely used heuristic method is the Lagrangian method (Beasley, 1993) that is based on Lagrangian relaxation and subgradient optimization (Daskin, 1995). More recently Gen, Tsujimura and Ishizaki (1996) and Vaithyanathan, Burke and Magent (1996) used artificial neural network approaches to solve UFL problems.

Although the tabu search meta-heuristic has been successfully applied to many complicated practical combinatorial optimization problems (Glover, 1989, 1990a, 1990b; Glover and Laguna, 1997; Glover, Taillard and de Werra 1993), not much research has been reported in using tabu search to solve UFL problems. However, tabu search procedures have been developed to solve more complicated facility location problems, such as the procedures by Delmaire, Díaz, Fernández and Ortega (1998) and by Tuzun and Burke (1999).

In addition to heuristics, there are a variety of exact algorithms for UFL problems although most of these exact algorithms were developed over a decade ago. Because the UFL problem is NP-hard (Cornuéjols, Nemhauser and Wolsey, 1990), exact algorithms may not be able to solve large practical problems. Krarup and Pruzan (1983) and Cornuéjols, Nemhauser and Wolsey (1990) gave excellent surveys and reviews of applications and solution methods of UFL problems.

Some UFL test problems used by other researchers and reported in the literature are collected, stored and updated in the OR-Library (Beasley, 1990). The test problems in the OR-Library provide a means for researchers to test their algorithms or heuristic procedures using the same set of test problems with known optimal solutions. Therefore, the performances of different heuristic procedures, especially their effectiveness in finding good solutions, become comparable even they are not directly compared.

3. The Tabu Search Heuristic Procedure

In this section, the details of the development of the tabu search heuristic procedure are discussed. We start with a discussion of the solution space being searched by the procedure and then describe the components of the heuristic procedure. We also discuss how the net changes in total cost are computed when the procedure evolves from one feasible solution to another within the solution space. After discussing the components, we give a step-by-step outline of the tabu search heuristic procedure.

3.1 The Solution Space

Let J denote the index set of all candidate facility location sites. For a given feasible solution, J is partitioned into two subsets J_0 and J_1 , where J_0 consists of the indices of those facilities that are currently closed and J_1 consists of the indices of those facilities that are currently open, *i.e.*,

$$J_0 = \{j \in J \mid y_j = 0\} \text{ and } J_1 = \{j \in J \mid y_j = 1\}. \quad (6)$$

We use n_0 and n_1 to denote the number of closed and the number of open facilities, respectively, of a given solution, *i.e.*, $n_0 = |J_0|$ and $n_1 = |J_1|$.

Let I denote the index set of customers. For each customer $i \in I$, d_i^1 is defined to be the index of the facility location, such that

$$c_{id_i^1} = \min\{c_{ij} \mid j \in J_1\}, \quad (7)$$

i.e., the cost of assigning customer i to facility location d_i^1 is the lowest among all facility locations that are currently open. If there is a tie, *i.e.*, if more than one j satisfies (7), any one of them can be designated as d_i^1 . Because the facilities are uncapacitated, the solution obtained by assigning each customer i to the corresponding facility d_i^1 is a feasible solution (Cornuéjols, Nemhauser and Wolsey, 1990). Such a solution also has the lowest total cost among all feasible solutions for the given partition of J into J_0 and J_1 . Therefore, we only need to evaluate one solution for each partition of J into J_0 and J_1 . Given a partition of J into J_0 and J_1 , the total cost in (1) of the corresponding solution can be computed using (8) in the following

$$z = \sum_{i=1}^m c_{id_i^1} + \sum_{j \in J_1} f_j . \quad (8)$$

The objective of the UFL problem is to find a partition of J with a corresponding solution that satisfies the demands of all customers at the minimum total cost. The strategy of this tabu search heuristic procedure is to search for an optimal partition of J among all these different partitions. The best solutions corresponding to these partitions constitute the solution space. An optimal solution can be found if an optimal partition of J is found.

There are other strategies to construct the solution space. One strategy is to search all feasible solutions, resulting in a larger solution space. Under this strategy, a customer i is allowed to be assigned to any facility j , such that $j \in J_1$. Because assigning customer i to facility d_i^1 results in the minimum cost among all feasible solutions for the given partition, searching such a larger solution space is not necessary and inefficient. Another strategy is to search both feasible and infeasible solutions, resulting in even a larger solution space. Under this strategy, a customer i is allowed to be assigned to any facility j , whether $j \in J_0$ or $j \in J_1$. This strategy is not efficient because infeasible solutions will never be chosen as the final solution.

When $n_1 > 1$, we also define d_i^2 to be the facility location, such that

$$c_{id_i^2} = \min\{c_{ij} \mid j \in J_1 \wedge j \neq d_i^1\} , \quad (9)$$

i.e., the cost of assigning customer i to facility location d_i^2 is the second lowest among all facilities that are currently open. If facility d_i^1 is closed, customer i will be reassigned to facility d_i^2 . When $n_1 = 1$, d_i^2 is not needed and is not defined.

3.2 Move

A move is defined to be the status change of one facility from open to closed or from closed to open. Thus a move leads to a new solution from the current solution. A solution is feasible if at least one facility is open. Any facility may change its status at a given solution as long as the resulting solution is feasible. Therefore, the only restriction on a move is that a facility cannot be closed if it is the only facility that is currently open. At each

feasible solution, there are n different candidate moves when $n_1 > 1$ and $n - 1$ candidate moves when $n_1 = 1$.

We use Δz_j to represent the net change in total cost as a result of the move switching the status of facility j . The value of Δz_j is used to measure the attractiveness of this move. The lower the value of Δz_j , the more attractive the move switching the status of facility j . The value of Δz_j is computed before a move is selected.

Thus, a move is a transition from one partition of J to another by taking one element from J_0 and putting it into J_1 or by taking one element from J_1 and putting it into J_0 . Each partition determines one solution. The two solutions before and after a move are adjacent because one can be reached from the other within one move. We use k to count the number of moves made since the start of the searching process.

3.3 Computation of the Net Changes in the Total Cost

Given the current partition of J into J_0 and J_1 , each customer i is assigned to facility d_i^1 in the corresponding solution. If facility j , such that $j \in J_0$, is opened, then a customer i will be reassigned to facility j only if $c_{ij} < c_{id_i^1}$. Otherwise, customer i will stay with facility d_i^1 . We use P_j to represent the set of indices of customers who will be reassigned to facility j after it is opened, i.e.,

$$P_j = \{i \in I \mid c_{ij} < c_{id_i^1}\}. \quad (10)$$

If facility j is opened, then Δz_j is given by

$$\Delta z_j = f_j + \sum_{i \in P_j} (c_{ij} - c_{id_i^1}). \quad (11)$$

To keep the resulting solution feasible, a facility j , such that $j \in J_1$, cannot be closed if it is the only facility that is currently open. In this case, we let $\Delta z_j = \infty$. If two or more facilities are currently open, then a facility may be closed and the resulting solution is still feasible. If facility j is closed, a customer i needs to be reassigned to facility d_i^2 only if $d_i^1 = j$ for the

resulting solution to be feasible and to have the minimum cost among all feasible solutions for the resulting partition. We use \mathcal{Q}_j to represent the set of indices of all customers who are currently assigned to facility j , *i.e.*,

$$\mathcal{Q}_j = \{i \in I \mid d_i^1 = j\}. \quad (12)$$

The net change in total cost Δz_j if facility j is to be closed is given by

$$\Delta z_j = -f_j + \sum_{i \in \mathcal{Q}_j} (c_{id_i^2} - c_{ij}). \quad (13)$$

If $\Delta z_j < 0$, the total cost will decrease after making the move that opens or closes facility j . A solution is a local optimal solution if $\Delta z_j \geq 0$ for all $j = 1, \dots, n$. However, it is sometimes necessary to make moves switching the status of facility j with $\Delta z_j \geq 0$ in order to move out of local optimal solutions. After a move is made, the values of the Δz_j 's are recomputed using (11) and (13).

3.4 The Short Term Memory Process

A vector $t \in \mathbb{R}^n$ is used to implement the recency based memory. For each facility j , t_j represents the move number, *i.e.*, the value of k , at which facility j changed status the last time. The total cost of the best solution found during the current short term memory process is denoted by z_0 and z_0 is updated each time a better solution is found. The move at which a better solution is found is denoted by k_0 and k_0 is updated when z_0 is updated. We use l to represent the tabu size, *i.e.*, each facility is kept at its new status for at least l moves after its status has been changed unless the aspiration criterion is satisfied. In this procedure l is variable and is allowed to vary between its lower limit l_1 and upper limit l_2 , *i.e.*, $l_1 \leq l \leq l_2$. However, the value of l does not change in the duration of the short term memory process and is reset only when the short term memory process restarts each time after the long term memory process finishes. A move switching the status of facility j is imposed the tabu status if facility j changed status within the last l moves.

For each move, we select a facility \bar{j} to switch status, such that,

$$\Delta z_{\bar{j}} = \min\{\Delta z_j \mid j = 1, \dots, n \text{ and facility } j \text{ is not flagged}\}. \quad (14)$$

Then the following tabu condition of the move is checked

$$k - t_{\bar{j}} \leq l. \quad (15)$$

If the tabu condition in (15) is satisfied, the move is tabu. The move will be made if it is not tabu, otherwise, the aspiration criterion in (16) is checked,

$$z + \Delta z_{\bar{j}} < z_0, \quad (16)$$

i.e., the resulting solution is better than the best solution found since the current short term memory process has started. Aspiration criteria used by Sun (1998), by Sun, Aronson, McKeown and Drinka (1998) and by Sun and McKeown (1993) are similar to the one used in this study and have been proven to be successful. If this aspiration criterion is satisfied, the move will be made even it is tabu. Otherwise, if it is tabu and this aspiration criterion is not satisfied, the move will not be made and facility \bar{j} is flagged. In this case, another facility is selected according to (14), the tabu status of the newly selected move is checked, and so on. This process is repeated until a move that is not tabu, or tabu but satisfies the aspiration criterion, is found and the move is made.

Let $\alpha > 0$ be such a coefficient that the short term memory process will stop if no solution better than z_0 can be found after αn moves have been made, *i.e.*, the short term memory process stops once the following condition in (17) is satisfied

$$k - k_0 > \alpha n. \quad (17)$$

The value of α is chosen by the user of the heuristic procedure before the solution process starts.

3.5 The Long Term Memory Process

In the long term memory process, a different criterion is used to choose a move among all candidate moves. The strategy used in the long term memory process is similar to those used by Sun, Aronson, McKeown and Drinka (1998) and by Sun and McKeown (1993). The procedure allows the long term memory process to be invoked C times and c is used to count the

number of times the long term memory process has been invoked. The value of C is chosen by the user before the solution process starts. After the long term memory process is invoked the c^{th} time, c moves are made in the current long term memory process before the short term memory process is restarted. For each move, a facility \bar{j} is chosen to switch status according to the following criterion

$$t_{\bar{j}} = \min\{t_j \mid j = 1, \dots, n\}. \quad (18)$$

Given the tabu condition specified in (15), each of the c facilities switching status in the long term memory process will be kept at its new status for at least l moves. By doing so, the search is led to a new region in the solution space to achieve the diversification purpose.

After the short term memory process is terminated, the procedure checks if the long term memory process has been invoked C times. If not, the long term memory process is invoked, otherwise, the tabu search procedure terminates with the best solution found as the final solution. A new round of the short term memory process starts each time after the long term memory process is finished. This process is continued until the long term memory process has been invoked C times.

We call α and C the parameters of the tabu search heuristic procedure. By varying the values of these parameters, we can control the thoroughness and extensiveness of the searching process. The searching period from the invoking of the long term memory process through the short term memory process is called a search cycle.

3.6 The Tabu Search Procedure

The search process starts from a local optimal solution. A local optimal solution can be obtained with any greedy method. The DROP method (Cornuéjols, Nemhauser and Wolsey, 1990; Daskin, 1995) is used in the implementation of this tabu search procedure. No differences in performance were noticed when the ADD method (Kuehn and Hamburger, 1963), instead of the DROP method, was used in the procedure. The best solution found since the search started is denoted by z_{00} and z_{00} is updated each time when a solution better than z_{00} is found. In the procedure, c_0 is used to count the number of moves made in the long term memory process each time the long term memory process is invoked.

The following is a step-by-step description of the tabu search procedure.

- Step 1. Use a greedy method to find a local optimal solution. Let z represent the total cost of the current solution. Let $z_0 = z$ and $z_{00} = z$. Determine l_1 and l_2 and determine the initial tabu size l , such that $l_1 \leq l \leq l_2$. Let $t_j < -l$ for all $j = 1, \dots, n$ to initialize the vector \mathbf{t} . Determine the value of the coefficient α and determine the number of times that the long term memory process is invoked, i.e., the value of the integer C . Let $k = 0$, $k_0 = 0$, $c_0 = 0$ and $c = 0$.
- Step 2. Compute Δz_j with (11) or (13) for each facility $j = 1, \dots, n$ and mark each facility j as unflagged.
- Step 3. If $k - k_0 \leq \alpha n$, go to Step 4; otherwise go to Step 8.
- Step 4. Select a move switching the status of facility location \bar{j} according to (14).
- Step 5. Check the tabu status of the selected move according to (15). If (15) is satisfied, go to Step 6; otherwise, go to Step 7.
- Step 6. Check the aspiration criterion of the selected move as specified in (16). If (16) is satisfied, go to Step 7, otherwise, mark facility \bar{j} as flagged and go to Step 4.
- Step 7. If $y_{\bar{j}} = 1$, let $n_1 = n_1 - 1$, otherwise let $n_1 = n_1 + 1$. Let $y_{\bar{j}} = 1 - y_{\bar{j}}$, $z = z + \Delta z_{\bar{j}}$, $t_{\bar{j}} = k$ and $k = k + 1$. If $z < z_0$, let $z_0 = z$ and $k_0 = k$. If $z < z_{00}$, let $z_{00} = z$. Go to Step 2.
- Step 8. If $c = C$, Stop. If $c_0 \leq c$, go to Step 9; otherwise, go to Step 10.
- Step 9. Let $c_0 = c_0 + 1$. Select a location \bar{j} according to (18) and go to Step 7.
- Step 10. Let $c_0 = 0$, $c = c + 1$, $z_0 = z$ and $k_0 = k$. Reset the value of l such that $l_1 \leq l \leq l_2$ and go to Step 4.

4. Computational Results

The proposed tabu search procedure was implemented in Fortran. For comparison purpose, a Lagrangian method (Beasley, 1993; Daskin, 1995) is also coded in Fortran. In addition to serving as a benchmark, the Lagrangian method also provides a lower bound on the value of the objective function of the optimal solution if the final solution obtained cannot be proven to be optimal. The Fortran codes were all compiled with the f77 compiler at optimization level 3 under the UNIX operating system. The computational experiments were conducted on a SUN Enterprise 3000 computer and the

computational results reported in this section reflect the performances of the tabu search procedure and the Lagrangian method on this computer.

4.1 Test Problems

Test problems both from the literature (Beasley, 1990) and randomly generated are used to conduct the computational experiment. Problems from the literature provide a benchmark to measure the performance of the tabu search heuristic procedure relative to other existing procedures. Because the optimal solution of each of these problems is known, the results of these problems can be used to verify the validity of the implementations of the Lagrangian method and the tabu search heuristic.

There are a few reasons for using randomly generated test problems. One reason is that not many test problems are available in the OR-Library and, therefore, these problems may not represent a large variety of real life problems. Another reason is that test problems in the OR-Library are possibly relatively easy to solve because of their relative small size. Even larger problems are relatively easy because their optimal solutions could be easily verified. Randomly generated test problems provide a much wider test bed. These randomly generated test problems vary along the following three dimensions, problem size, problem shape and ranges of cost coefficients. These three dimensions may be called the properties of the test problems. Each set of the given problem properties defines a problem category. Results of totally 18 problem categories are reported. For each given problem category, 15 randomly generated test problems are used. Therefore, results of 270 randomly generated test problems are reported in this section. The results of one problem category appeared in all the tables reporting results for randomly generated test problems.

4.2 Results of Test Problems from the Literature

Currently there are 15 UFL test problems in the OR-Library (Beasley, 1990). Among these 15 test problems, 12 of them are relatively small in size ranging from $m \times n = 50 \times 16$ to $m \times n = 50 \times 50$ and the other 3 test problems are relatively large with $m \times n = 1000 \times 100$. All these 15 test problems were used in this study.

Results of these problems are shown in Table 8.1. Each row of the table gives the results of a single test problem. The names of these test problems as used in the OR-Library are listed in Column 1. The size of each test problem, measured by $m \times n$, is listed in Column 2.

For the tabu search heuristic procedure, the solutions were obtained with the parameters $10 \leq l \leq 20$, $C = 5$ and $\alpha = 2.5$. The numbers in the column

of c^{opt} are the number of search cycles in which the optimal solution was found with the tabu search procedure. Given $C = 5$, the maximum possible value for c^{opt} is $c^{opt} = 6$. From this column we can see that the optimal solution was found multiple times for each test problem. The numbers in the column of c^o are the number of times the long term memory process was invoked before the optimal solution was found the first time. A $c^o = 0$ means that the optimal solution was found before the long term memory process was invoked the first time. From this column we can see that the optimal solution had been found for all, but one, of the 12 small test problems even before the long term memory process was invoked once. Optimal solutions were also easily obtained when different ranges of l and different values of C and α were used in the tabu search procedure. The Lagrangian method that we implemented for this study found optimal solutions for all, but the last one, of these test problems.

Table 8.1. Computational Results of Test Problems from the Literature

Problem		Tabu Search			Lagrangian
Name	Size	c^o	c^{opt}	CPU Time (seconds)	CPU Time (seconds)
Cap71	50×16	0	6	0.07	0.07
Cap72	50×16	0	5	0.06	0.08
Cap73	50×16	0	6	0.06	0.08
Cap74	50×16	0	6	0.06	0.07
Cap101	50×25	0	5	0.10	0.07
Cap102	50×25	0	6	0.09	0.12
Cap103	50×25	0	6	0.12	0.10
Cap104	50×25	0	6	0.09	0.06
Cap131	50×50	0	2	0.28	0.20
Cap132	50×50	0	4	0.28	0.19
Cap133	50×50	0	5	0.30	0.28
Cap134	50×50	2	2	0.27	0.20
Capa	1000×100	0	4	24.14	7.97
Capb	1000×100	3	2	24.34	8.69
Capc	1000×100	2	2	23.89	9.19

The two procedures took approximately the same amount of CPU time for the smaller test problems as shown in the table. For larger problems, the tabu search heuristic took more CPU time than the Lagrangian method. However, the CPU time used by the tabu search heuristic procedure could be substantially reduced and the optimal solution of each problem could still be found. From the results in the c^o column of Table 8.1, we can see that the optimal solution of each test problem was found early in the searching process. Therefore, by reducing the value of C , the tabu search procedure would use much less CPU time without sacrificing solution quality. We are

more concerned with the quality of the final solution than with the computation time if a problem can be solved within a reasonable amount of CPU time and, therefore, a $C = 5$ was used for all these test problems in the computational experiment.

Grolimund and Ganascia (1997) reported computational results on the first 12 relatively small test problems (problems Cap71-Cap134) of their tabu search procedure although they did not report computational results on the last 3 relatively large test problems. Using several CPU hours for each test problem, their most effective procedure obtained solutions 14% worse than the optimal solutions on the average while their least effective procedure obtained solutions 34% worse than the optimal solutions on the average. As noted in Table 8.1 and discussed above, the tabu search procedure developed in this study found optimal solutions for 11 out of these 12 small test problems even before the long term memory process was invoked once. In fact, the greedy method implemented in this tabu search procedure found optimal solutions for 6 out of these 12 small test problems.

4.3 Results of Randomly Generated Test Problems

For randomly generated test problems we report statistical results for problems with similar properties instead of reporting the results for each individual problems. Because the optimal solution of a randomly generated test problem is unknown, our focus will be put on the solution quality. Let \underline{z} denote the lower bound obtained with the Lagrangian method on the value of the objective function of the optimal solution. Let z^t and z^l respectively denote the total cost of the best solution found by the tabu search heuristic procedure and the Lagrangian method. Then q defined in (19) in the following is used to measure the relative quality of the solutions obtained with the two methods

$$q = \frac{z^l - \underline{z}}{z^t - \underline{z}} \times 100. \quad (19)$$

If $z^t = z^l$, then $q = 100$. For a given test problem, the larger the value of q , the better the solution obtained with the tabu search procedure than that obtained with the Lagrangian method. The average, minimum and maximum of the values of q of the 15 test problems in each problem category are reported under the heading Cost Ratio in the following tables.

In addition to the relative solution quality, we also reported the CPU time taken by these methods to solve these problems. In the following tables, the

average CPU time in seconds of the 15 test problems in each problem category is reported.

As for test problems from the literature, the results reported in the following tables were obtained with parameter values $10 \leq l \leq 20$, $C = 5$ and $\alpha = 2.5$ in the tabu search procedure. For each and every test problem, the tabu search heuristic procedure found a solution better than that found by the Lagrangian method without any exception. We also tried different ranges for l and different values for C and α . Each time the tabu search heuristic procedure found a solution better than that found by the Lagrangian method for each and every test problem. As for test problems from the literature, with $C = 5$ and $\alpha = 2.5$ we purposely conducted a thorough search for a better solution for each problem with the tabu search procedure. When the value of C was reduced, however, the tabu search heuristic procedure used much less CPU time and still found a solution better than that found by the Lagrangian method for each test problem.

4.3.1 Effect of Problem Sizes

Results of test problems with different sizes are reported in Table 8.2. Problem size is measured with $m \times n$. We started with test problems with $m \times n = 500 \times 50$. Then we increased the value of m at an interval of 500 and increased the value of n at an interval of 50. Four test problem categories are used with the sizes varying from 500×50 to 2000×200 . For all these test problems, the cost coefficients are in the following ranges $1000 \leq c_{ij} \leq 2000$ and $10000 \leq f_j \leq 20000$.

Table 8.2. Results of Problems with Different Sizes

$m \times n$	Cost Ratio			CPU Time (seconds)	
	Average	Minimum	Maximum	Tabu Search	Lagrange
$m \times n = 500 \times 50$	120.51	105.64	144.04	2.78	1.88
$m \times n = 1000 \times 100$	117.53	100.94	134.47	21.25	7.62
$m \times n = 1500 \times 150$	119.81	110.49	131.82	90.97	17.92
$m \times n = 2000 \times 200$	117.92	109.29	130.24	242.51	40.99

The average value of q as defined in (19) is about 120 for all four problem categories, which means that the tabu search procedure found much better solutions than the Lagrangian method on the average. The minimum value of q of all these test problems is over 100, which means that the tabu search procedure found a better solution for each individual test problem without any exception. Because similar values of q were obtained for all

four problem categories, there is not evidence to show that problem size affects the relative effectiveness of the two solution procedures.

With the values of C and α used in the computational experiment, the tabu search procedure took more CPU time than the Lagrangian method did. As expected, larger problems took more CPU time than smaller problems with both the tabu search procedure and the Lagrangian method.

4.3.2 Effects of Problem Shape

Problem shape is measured by the relative values of m and n . First by fixing the value of n at $n=100$, we increased the value of m from 500 to 2000 at an interval of 500. The results of these four problem categories are reported in Table 8.3. Then by fixing the value of m at $m=1000$, we increased the value of n from 50 to 200 at an interval of 50. Results of these four problem categories are reported in Table 8.4. The cost coefficients are in the ranges $1000 \leq c_{ij} \leq 2000$ and $10000 \leq f_j \leq 20000$ for all these test problems.

Table 8.3. Results of Problems with Different m

$m \times n$	Cost Ratio			CPU Time (seconds)	
	Average	Minimum	Maximum	Tabu Search	Lagrange
$m \times n = 500 \times 100$	122.69	105.75	135.30	11.53	3.95
$m \times n = 1000 \times 100$	117.53	100.94	134.47	21.25	7.62
$m \times n = 1500 \times 100$	119.65	106.58	129.58	43.51	11.63
$m \times n = 2000 \times 100$	116.99	103.73	127.42	53.47	15.25

As for problems with different sizes, the average value of q as defined in (19) of each test problem category is about 120 indicating that the tabu search heuristic found much better solutions on the average than the Lagrangian method. The minimum value of q is also over 100 which means that the solutions of the tabu search heuristic completely dominate the solutions of the Lagrangian method.

The tabu search heuristic again used more CPU time than the Lagrangian method due to the values of C and α used in the tabu search heuristic. The average CPU time increased faster for problems with a fixed m when n increased than for problems with a fixed n when m increased. Problems with a relatively larger n take more CPU time because the CPU time needed is more closely related to n than to m as indicated in (17), the stopping criterion of the short term memory process of the tabu search heuristic.

Table 8.4. Results of Problems with Different n

$m \times n$	Cost Ratio			CPU Time (seconds)	
	Average	Minimum	Maximum	Tabu Search	Lagrange
$m \times n = 1000 \times 50$	118.04	107.61	130.81	7.06	3.55
$m \times n = 1000 \times 100$	117.53	100.94	134.47	21.25	7.62
$m \times n = 1000 \times 150$	120.11	107.43	133.51	60.43	11.78
$m \times n = 1000 \times 200$	120.20	113.56	130.31	106.90	16.28

4.3.3 Effects of Cost Coefficient Ranges

Ranges of cost coefficients are measured by the lower and upper limits of c_{ij} and f_j within which the cost coefficients were generated. We first fixed f_j in the range $10000 \leq f_j \leq 20000$ and varied the ranges of c_{ij} from $500 \leq c_{ij} \leq 1000$ to $8000 \leq c_{ij} \leq 16000$. The values of the lower and upper limits of c_{ij} are both doubled from one problem category to the next, resulting in a total of 5 problem categories. The computational results of these test problems are reported in Table 8.5. We then fixed the range of c_{ij} at $1000 \leq c_{ij} \leq 2000$ and varied the ranges of f_j from $1250 \leq f_j \leq 2500$ to $20000 \leq f_j \leq 40000$. The lower and upper limits of f_j are also doubled from one problem category to the next, resulting in a total of 5 problem categories. The computational results of these test problems are reported in Table 8.6. For all these test problems, the problem size is fixed at $m \times n = 1000 \times 100$.

Table 8.5. Results of Problems with Different c_{ij} Ranges

c_{ij}	Cost Ratio			CPU Time (seconds)	
	Average	Minimum	Maximum	Tabu Search	Lagrange
$500 \leq c_{ij} \leq 1000$	114.62	106.69	121.65	23.77	7.82
$1000 \leq c_{ij} \leq 2000$	117.53	100.94	134.47	21.25	7.62
$2000 \leq c_{ij} \leq 4000$	122.18	114.68	135.83	26.91	7.59
$4000 \leq c_{ij} \leq 8000$	129.54	120.64	141.83	27.00	8.49
$8000 \leq c_{ij} \leq 16000$	138.51	124.55	166.71	29.01	9.57

For problems with a fixed range of f_j , the average value of q as defined in (19) increases as the values of c_{ij} increase as shown in Table 8.5. For problems with a fixed range of c_{ij} , the average value of q decreases as the values of f_j increase as shown in Table 8.6. These trends may indicate that

the tabu search heuristic is relatively more effective than the Lagrangian method for problems with relatively large values of c_{ij} and relatively small values of f_j . The average value of q is between 115 and 150 for all these problem categories, indicating that the tabu search heuristic found much better solutions than the Lagrangian method on the average. Once more, the minimum value of q is over 100, indicating that the tabu search heuristic found a solution better than that found with the Lagrangian method for each individual test problem without any exception. In fact, the minimum value 100.94 of q is obtained on one test problem in the problem category with $1000 \leq c_{ij} \leq 2000$, $10000 \leq f_j \leq 20000$ and $m \times n = 1000 \times 100$ and the results for this problem category appeared in all tables for randomly generated test problems.

The tabu search heuristic again used more CPU time than the Lagrangian method for these test problems because we purposely conducted a thorough search with the tabu search heuristic. No dramatic differences in average CPU times are observed for problems with different ranges of c_{ij} or different ranges of f_j , indicating that the values of m and n are determinants of CPU time while the values of cost coefficients are not.

Table 8.6. Results of Problems with Different f_j Ranges

f_j	Cost Ratio			CPU Time (seconds)	
	Average	Minimum	Maximum	Tabu Search	Lagrange
$1250 \leq f_j \leq 2500$	147.52	128.66	174.92	28.62	10.44
$2500 \leq f_j \leq 5000$	131.27	113.29	150.18	28.53	9.30
$5000 \leq f_j \leq 10000$	124.75	110.40	151.26	26.97	7.52
$10000 \leq f_j \leq 20000$	117.53	100.94	134.47	21.25	7.62
$20000 \leq f_j \leq 40000$	115.50	107.52	125.59	24.72	8.25

For all randomly generated test problems, the best solution was found very early in the solution process. In fact, the best solution was found before the long term memory process was invoked even once for over 90% (251 out of 270) of these randomly generated test problems. However, to be on the safe side, the long term memory process was invoked $C = 5$ times for all the test problems in the computational experiment. Therefore, much less CPU time is needed than listed in the tables to reach the best solution and the tabu search heuristic works much better than it looks.

5. Conclusions

A tabu search heuristic procedure for the UFL problem is developed, implemented and tested. The procedure is tested against the Lagrangian method with test problems with known optimal solutions used by other researchers and test problems randomly generated. The tabu search heuristic found optimal solutions within a single run for all test problems from the literature. For each randomly generated test problem, the tabu search procedure found a solution in a single run better than that found by the Lagrangian method without any exception. The CPU time taken by the tabu search procedure is reasonable although it uses more CPU time than the Lagrangian method. We believe that solution quality is more important than solution time given that a problem can be solved within a reasonable amount of CPU time. Therefore, the extra CPU time taken by the tabu search heuristic procedure was well spent.

In addition to applying the UFL model and the tabu search procedure to real life problems, developing more effective and efficient heuristic solution procedures appears to be a research direction. Future research in this direction may focus on the selection of the parameters in the tabu search procedure to improve its effectiveness and efficiency. Another possibility is to use a candidate list approach when selecting a move to make so as to reduce CPU time needed. In this way, the net changes in the total cost are computed for only a subset of the facility locations for each move.

References

- Beasley, J.E. (1990) "OR-Library: Distributing Test Problems by Electronic Mail," *Journal of the Operational Research Society*, 41(11):1069–1072.
- Beasley, J.E. (1993) "Lagrangian Heuristics for Location Problems," *European Journal of Operational Research*, 65:383–399.
- Brandeau, M.L. and S.L. Chiu (1989) "Overview of Representative Problems in Location Research," *Management Science*, 35(6):695–674.
- Chan, Y. (2001) *Location Theory and Decision Analysis*, South-Western College Publishing, Cincinnati, Ohio.
- Chhajed, D., R.L. Francis and T.J. Lowe (1993) "Contributions of Operations Research to Location Analysis," *Location Science*, 1:263–287.
- Cornuéjols, G., M.L. Fisher and L.A. Wolsey (1977) "Location of Bank Accounts to Optimize Float: An Analytic Study of Exact and Approximate Algorithms," *Management Science*, 23:789–810.

- Cornuéjols, G., G.L. Nemhauser and L.A. Wolsey (1990) "The Uncapacitated Facility Location Problem," in P.B. Mirchandani and R. L. Francis (eds), *Discrete Location Theory*, Wiley, New York, 119–171.
- Daskin, M.S. (1995) *Network and Discrete Location, Models, Algorithms, and Applications*, Wiley, New York.
- Delmaire, H., J.A. Díaz, E. Fernández and M. Ortega (1998) "Reactive GRASP and Tabu Search Based Heuristics for the Single Source Capacitated Plant Location Problem," *Information Systems and Operations Research*, 37(3):194–225.
- Erlenkotter, D. (1978) "A Dual-Based Procedure for Uncapacitated Facility Location," *Operations Research*, 26:992–1009.
- Gen, M., Y. Tsujimura and S. Ishizaki (1996) "Optimal Design of a Star-LAN Using Neural Networks," *Computers and Industrial Engineering*, 31(3/4):855–859.
- Ghosh, A. and F. Harche (1993) "Location-Allocation Models in the Private Sector: Progress, Problems, and Prospects," *Location Science*, 1:81–106.
- Glover, F. (1989) "Tabu Search, Part I," *ORSA Journal on Computing*, 1(3): 190–206.
- Glover, F. (1990a) "Tabu Search, Part II," *ORSA Journal on Computing*, 2(1): 4–32.
- Glover, F. (1990b) "Tabu Search: A Tutorial," *Interfaces*, 20(4):74–94.
- Glover, F. and M. Laguna (1997) *Tabu Search*, Kluwer Academic Publishers, Hingham, MA.
- Glover, F., E. Taillard and D. de Werra (1993) "A User's Guide to Tabu Search," *Annals of Operational Research*, 41(1-4):3–28.
- Grolimund, S. and J.G. Ganascia (1997) "Driving Tabu Search with Case-Based Reasoning," *European Journal of Operational Research*, 103(2): 326–338.
- Krarup, J. and P.M. Pruzan (1983) "The Simple Plant Location Problem: Survey and Synthesis," *European Journal of Operational Research*, 12(1): 36–81.
- Krarup, J. and P.M. Pruzan (1990) "Ingredients of Location Analysis," in *Discrete Location Theory* (P. B. Mirchandani and R. L. Francis, eds), Wiley, New York, 1–54.
- Kuehn, A.A. and M.J. Hamburger (1963) "A Heuristic Program for Locating Warehouses," *Management Science*, 9:643–666.
- Mirchandani, P.B. and R.L. Francis (1990) *Discrete Location Theory*, Wiley, New York.
- Nemhauser, G.L., L.A. Wolsey and L.M. Fisher (1978) "An Analysis of Approximations for Maximizing Submodular Set Functions, I," *Mathematical Programming*, 14:265–294.

- Simchi-Levi, D., P. Kaminsky and E. Simchi-Levi (2000) *Designing and Managing the Supply Chain, Concepts, Strategies and Case Studies*, Irwin McGraw-Hill, Boston, Massachusetts.
- Sun, M. (1998) "A Tabu Search Heuristic Procedure for Solving the Transportation Problem with Exclusionary Side Constraints," *Journal of Heuristics*, 3(4):305–326.
- Sun, M., J.E. Aronson, P.G. McKeown and D. Drinka (1998) "A Tabu Search Heuristic Procedure for the Fixed Charge Transportation Problem," *European Journal of Operational Research*, 106(2-3):441–456.
- Sun, M. and P.G. McKeown (1993) "Tabu Search Applied to the General Fixed Charge Problem," *Annals of Operations Research*, 41:405–420.
- Tuzun, D. and L.I. Burke (1999) "A Two-Phase Tabu Search Approach to the Location Routing Problem," *European Journal of Operational Research*, 116(1):87–99.
- Vaithyanathan, S., L. Burke and M. Magent (1996) "Massively Parallel Analog Tabu Search Using Neural Networks Applied to Simple Plant Location Problems," *European Journal of Operational Research*, 93:317–330.

Chapter 9

ADAPTIVE MEMORY SEARCH GUIDANCE FOR SATISFIABILITY PROBLEMS

Arne Løkketangen¹ and Fred Glover²

¹*Molde College, Molde, Norway, arne.lokketangen@himolde.no;*

²*Leeds School of Business, University of Colorado at Boulder, USA,
fred.glover@colorado.edu*

Abstract: Satisfiability problems (SAT) are capable of representing many important real-world problems, like planning, scheduling, and robotic movement. Efficient encodings exist for many of these applications and thus having good solvers for these problems is of critical significance. We look at how adaptive memory and surrogate constraint processes can be used as search guidance for both constructive and local search heuristics for satisfiability problems, and how many well-known heuristics for SAT can be seen as special cases. We also discuss how adaptive memory learning processes can reduce the customary reliance on randomization for diversification so often seen in the literature. More specifically, we look at the tradeoff between the cost of maintaining extra memory search guidance structures and the potential benefit they have on the search. Computational results on a portfolio of satisfiability problems from SATLIB illustrating these tradeoffs are presented.

Keywords: Adaptive Memory, Local Search, Satisfiability Problems

1. Introduction

Many important real-world problems can be represented as satisfiability problems. These include planning, scheduling and robotic movement, and efficient encodings exist for many of these. Efficient solvers for these problems are thus of critical significance. SAT has thus received substantial attention in recent years, and efficient SAT solvers exist.

What sets SAT apart from other combinatorial optimization problems is that SAT is basically a feasibility problem. Once a variable assignment is

found that satisfies all the clauses (see Section 2), the problem is solved, and this condition is readily detected. In SAT there is thus no guidance from the normal objective function. Guidance is customarily instead based on the amount of infeasibility, usually by counting the number of unsatisfied clauses for a given solution, possibly modified by the clause length.

There are many approaches to solving the SAT problem. Constructive methods range from complete tree-search (DPL - Davis-Putnam-Loveland , see Davis , Logemann and Loveland, 1962, Davis and Putnam, 1960), to constructive heuristics like GRASP (Resende and Feo, 1996) and surrogate constraint based learning heuristics (Løkketangen and Glover, 1997). Most heuristic solvers for SAT are based on local search starting from randomly or otherwise constructed starting solutions. For a nice overview of many of the heuristics for SAT, see Hoos (1998). See also Section 4.

The work presented in this paper is based on previous work by the authors on the satisfiability problem, where the basic framework and search mechanisms was developed . For details, see Løkketangen and Glover (1997).

We will show how the judicious use of surrogate constraint based local search guidance, with the augmentation of adaptive memory structures for short and long-term learning and forgetting, provides superior search guidance, at an extra computational cost per iteration. Many of the popular heuristics for SAT can be derived as special cases, and we show that additional heuristic power results by considering more general forms of this guidance framework. We also discuss how adaptive memory processes can reduce the customary reliance on randomization for diversification so often seen in the literature.

We report computational tests that compare solution attempts both in terms of execution time and number of local search steps, using a set of state-of-the-art local search heuristics that are augmented by varying degrees of search guidance, and adaptive memory capabilities. The tradeoffs between the increased solution time required by fuller reliance on adaptive memory, and the reduced numbers of iterations that are required to obtain feasible solutions, are illustrated on a portfolio of satisfiability problems taken from SATLIB (see SATLIB).

The layout of this extended abstract is as follows. This introduction is followed in Section 2 by a description of the SAT problem. In Section 3 we look at surrogate constraints, while a brief outline of SAT solvers is presented in Section 4. Our choice of search guidance mechanisms is described in Section 5, and the computational results in are in Section 6, followed by the conclusions in Section 5.

2. The SAT Problem

The Satisfiability problem originates from the realm of logic theorem proving, and was the first problem proven to be NP-Complete (Cook 1971). All other NP-Complete problems can be reduced to SAT in polynomial time. The SAT problem can be defined as follows. Given the logical function, consisting of combinations of disjunctions, conjunctions and negations of a set of variables x_1, \dots, x_N , then the SAT problem is to find a set of truth assignments to the literals that will make $\Phi(x)$ true (or false):

$$SAT: \Phi(x) = \Phi(x_1, \dots, x_N) = \begin{cases} \text{true} \\ \text{false} \end{cases}$$

The logical function $\Phi(x)$ is usually represented in CNF, *Conjunctive Normal Form*. $\Phi(x)$ then consists of a set of conjunctions of clauses $c_i(x)$, written $\Phi = c_1 \wedge c_2 \wedge \dots \wedge c_M$, where each clause is a disjunction of complemented and uncomplemented variables, called literals, with M being the number of clauses. As a simple example, let $\Phi(x)$ be the following formula containing 3 variables and 5 clauses:

$$\Phi(x) = (x_1 \vee x_2) \wedge (x_1 \vee x_3) \wedge (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2) \wedge (x_1 \vee \neg x_3)$$

2.1 Mathematical Formulation

To get to the more customary mathematical formulation, replace true/false with 1/0, disjunction with +, representing each conjunction as a separate constraint row. Let literals be represented by x_j and their complements by $1 - x_j$. This gives us:

$$\begin{aligned} Ax &\geq b \\ x &\text{ binary} \\ \text{where } A &\text{ is an } m \times n \text{ matrix of 0's, 1's and -1's} \\ \text{and } b &\text{ and } x \text{ are } n \times 1 \text{ column vectors.} \end{aligned}$$

The i^{th} constraint of the system,

$$A_i x \geq b_i$$

has the property that the number of -1's in the row vector A_i equals $1 - b_i$, where b_i is an integer ≤ 1 .

To get a more convenient representation, we split each variable x_i into its complemented and uncomplemented occurrences, we get the following constraint set for the example, with the variable pair z_i and $z_{i\#}$ represents x_i :

$$\begin{array}{lll}
 z_1 + z_2 & \geq 1 & (w_1) \\
 z_1 & + z_3 & \geq 1 & (w_2) \\
 z_2 & + z_3 & \geq 1 & (w_3) \\
 & z_4 + z_5 & \geq 1 & (w_4) \\
 z_1 & + z_6 \geq 1 & (w_5)
 \end{array}$$

(The w 's are weights used for learning purposes in surrogate constraint evaluations, see the next section):

Our final model is then

$$Dz \geq 1 \quad (2.1)$$

$$z_i + z_{i\#} = 1 \quad (2.2)$$

where D is the 0-1 matrix obtained by substituting the z 's for the x_i 's. The last constraint (2.2) is handled implicitly in the search heuristics we describe.

3. Surrogate Constraints

A Surrogate Constraint (SC) is a weighted linear combinations of the original problem constraints (Glover, 1977), and provides a powerful way to capture constraint information to be used for search guidance. The basic use of SC methods for both constructive and local search heuristics for SAT is described in Løkketangen and Glover (1997).

Given the transformed constraint set of (2.1), we introduce a nonnegative vector w of weights w_i , to generate a surrogate constraint

$$sz \geq s_o$$

where $s = wD$ and $s_o = \Sigma w$. The surrogate constraint therefore results by weighting each constraint row j by w_j and summing. Assuming (as is the case initially in our searches) that all the w_i 's are 1, we get the following surrogate constraint from the example (other weightings will of course result in a different SC):

$$3z_1 + 2z_2 + 2z_3 + z_4 + z_5 + z_6 \geq 5$$

This surrogate indicates that the best would be to set z_1 to 1, and thus select x_1 to be *true*. If one also considers that the pair (z_1, z_4) really represents the same variable, x_1 , and both cannot simultaneously be set, we can form a *derived* surrogate constraint by replacing s_j with $s_j - \text{Min}(s_j, s_{j\#})$ in the surrogate constraint evaluation. We then get:

$$2z_1 + z_2 + z_3 \geq 2$$

indicating even stronger that x_I should be set to *true*. In the event of ties, we choose the variable with the maximum $s_j - s_{j\#}$ value.

We will use (derived) surrogate constraint based search guidance for the local searches, augmented by various levels of adaptive memory capabilities.

4. On Solving SAT

In this section we will look briefly at the most common constructive and iterative local search paradigms for solving SAT, and some of the existing solvers. For a nice overview of many of the heuristics for SAT, see Hoos (1998), and the bibliography at SATLIB. There are myriads of different solvers for SAT, but most falls into one of the broader categories, and shares most features with other solvers.

4.1 Constructive Methods

Within the constructive solver class, there is a big distinction between *complete* methods, guaranteeing to prove that a formula is satisfiable or not, and *heuristic methods* designed to try to find a solution if one exists. The best known complete method for SAT is DPL – Davis-Putnam-Loveland (Davis, Logemann and Loveland, 1962, Davis and Putnam, 1960). This is a deterministic tree-search with backtracking. The problem with this approach is the limited size of the problem instances that can be solved in reasonable time.

A constructive search usually contains the following elements and search flow:

1. All variables initially unassigned
2. Construct solution by assigning a truth-value to one variable at the time.
(Neighborhood is the set of remaining unassigned variables).
3. When no feasible assignments can be made:
 - Full Backtrack (complete method - DPL)
 - Limited backtracking with restart - (DPL with restarts)
4. Finish construction and:
 - Submit to (limited) local search and restart – (GRASP, SC-Learn)
5. Need move evaluation guidance. This is usually based on change in feasibility..
6. For restart-methods, guidance should be modified by history (SC-Learn)
Among constructive heuristic methods are GRASP (Resende and Feo, 1996), DPL with restarts (Gomes, Selman and Kautz, 1998), and SC-Learn, a surrogate constraint based learning heuristics (Løkketangen and Glover, 1997).

DPL with restart (Gomes, Selman and Kautz, 1998) is a DPL-based tree-search with limited backtracking, and only in the bottom of the search tree. This work is inspired by the phenomenon of heavy-tailed cost distributions, in that at any time during the experiment there is a non-negligible probability of hitting a problem that requires exponentially more time to solve than any that has been solved before (Gomes et. al. 1998). Instead of risking spending such a long time futilely searching, the search is restarted, but different, controlled randomized choices are made in the new search.

GRASP - Greedy Randomized Adaptive Search Procedure (Resende and Feo, 1996),. This is a constructive heuristic followed by a short greedy local search, trying all combinations of improving flips. It can be called a shotgun method, as its aim is generate a diverse set of solutions quickly, some of which might be the solution. The basic heuristic for assigning one variable value is:

- For each unassigned variable, count the number of clauses that are satisfied by assigning it True (and similarly for False).
- Sort the values. Select randomly among the top half evaluations (or max 50).

This corresponds to a basic Surrogate Constraint using uniform weighting,, and no normalization. There is also no learning, or use of memory, between restarts.

SC-Learn (Løkketangen and Glover, 1997) uses adaptive memory structures to learn between runs. More specifically, it gives added focus on the clauses that have been difficult to satisfy so far. Surrogate constraints are used for move evaluations.

4.2 Iterative Local Search Methods

All of the iterative local search methods for SAT are incomplete methods, in that the non-existence of a solution can not be proven. An iterative local search usually contains the following elements and search flow:

1. All variables are assigned a truth-value at all times
2. The starting solution (or starting point) is usually based on a random assignment to the variables or based on a construction heuristic.
3. A move is the flip of a variable. A flip means assigning the opposite value to a variable. (i.e. change $1 \rightarrow 0$ or $0 \rightarrow 1$).
4. The search neighborhood is either the full set of variables, or just those that appear in unsatisfied clauses.
5. Move evaluation is based on changes in feasibility. I.e. select moves that reduce the number of unsatisfied clauses. This measure can be modified by history.

6. The move selection is greedy (i.e. take the best move according to the move evaluation).
7. A random restart is applied after a certain number of moves, to diversify the search after stagnation
8. The stopping criterion is a simple time limit, a cutoff on the number of allowable flips or the identification of a solution.

There are extremely many iterative local search methods for SAT. Among the first, and most well-known, are GSAT (Selman, Levesque and Mitchell, 1992), and the whole family of search methods derived from it. (Walksat, GSAT+Tabu, Novelty,...). For an overview, see Hoos (1998). These methods are generally very simple and have fast iterations. Random restarts are usually employed when restarting.

GSAT starts from a randomly generated starting solution. The moves are variable flips. Move evaluation is based on the change in the number of satisfied clauses. (Choose randomly among ties). Don't allow downhill (worsening) moves. Do a random restart after a certain number of flips. This corresponds to using the derived surrogate constraint, without the SC choice rule (for ties).

Novelty (McAllester, Selman and Kautz, 1997). This is considered one of the best local search heuristics for SAT. Each iteration a violated clause is selected randomly. Then the best (in terms of improved infeasibility) variable to flip in this clause is identified. (In the case of ties, select the least recently flipped variable). If this variable is not the most recently flipped, flip it. Otherwise select the next best variable with probability p , and with probability $1-p$ select the best variable. This heuristic works very well on random 3-sat.

SC-Learn (Løkketangen and Glover, 1997) starts from a randomly generated starting solution. The moves are variable flips. A simple tabu search is added to avoid move reversals. Diversification is with the modification of clause weights used in the surrogate constraint based move evaluations.

5. Search Guidance Structures and Mechanisms

In this section we will look at enhancements to the iterative SC-Learn heuristics (Løkketangen and Glover, 1997). More specifically we will look at adding forgetting to the learning, sensitivity to learning weights and the introduction of controlled randomization in the move selection. We will also look at examples of how the different search mechanisms are far from independent, and that when adding a new search mechanism, the already implemented ones can change behaviour.

All the gathering of information about the search development during the search process, updating of the adaptive memory structures, and the processing of the gathered information takes additional computing time. The purpose of this is to provide better search guidance, thus needing fewer iterations to get to the solution. (Note that the solution can be reached in at most N steps, where N is the number of variables. The actual number of flips needed by many of the local search methods are often several orders of magnitude larger). It is therefore of interest to look at this trade-off between randomization and use of adaptive memory structures for search guidance and diversification. A new heuristic, SC-RN is developed, and will be described below.

5.1 Learning

We use frequency based information to modify the clause weights in a judicious way. Given the current solution vector, we know that at least one of the variables in one of the violated clauses has the wrong value, and hence place an emphasis on changing values of these variables. We do this in the SC framework by increasing the weights of the *violated clauses* every iteration. (This was also used in Løkketangen and Glover, 1997, and a different weighting scheme was tried in Frank, 1996). We have found that the increment used is not important, and a value of 1 is used in the tests. Preliminary testing has also shown that resetting these weights at fixed intervals has no discernible effect.

5.2 Forgetting

The accuracy of the information embedded in the clause weights vanes over time, and should have decreasing impact. This is accomplished by increasing the weight used in the learning process slightly every iteration. This leads to a discounting of the oldest values. Preliminary testing have shown that the value of the forgetting (or discounting) increment likewise is not important as long as it is significantly smaller than the actual weights.

5.3 Tabu Tenure and the Non-independence of Search Mechanisms

Our search uses the basic tabu criterion of not to flip a variable that has recently been flipped. (A good treatment of tabu search is in Glover and Laguna, 1997). One problem is to determine the optimal, or best, tabu tenure in terms of some problem parameter, like the number of variables. Mazure, Saïs and Grégoire (1997) added a simple tabu criterion as described above to GSAT (naming the new method TSAT). One of their findings was a linear

relationship between the optimal tabu tenure and problem size, according to the following formula:

$$TT_{OPT} = 0.01875 * N + 2.8125$$

with N being the number of variables for random hard 3-SAT instances.

We similarly tried different values of TT combined with the basic learning scheme on the test instance *aim-50-2_0-yes-2* taken from SATLIB. This problem has 50 variables, and is not very difficult. Table 9.1 shows the rate of success for 5 runs from random starting positions with a maximum of 5000 flips, and varying TT, using a weight increment of 1. The table indicates a best TT of 10.

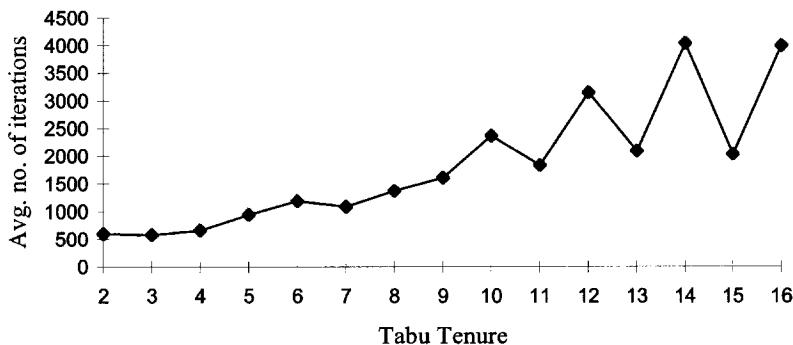


Figure 9.1. Search performance for TT with discounted learning

When rerunning the same test, but with discounting of the learning weights (using an increment of 0.1), we get the results of Table 9.1. The solution is now found for all values of TT between 2 and 16. In Figure 9.1 is shown the average search effort (in terms of flips) for each of the TT's where a solution was found for all runs. This graph shows clearly that even though the search always finds the optimal solution for a TT value in the range 2 to 17, it uses far more iterations for the larger values of the TT. The best values for the TT are 2 or 3, while without discounting this value is much larger. The addition of the discounting mechanism clearly changes the effective TT values, and has a great stabilizing effect.

Table 9.1. Effect of TT with simple learning

TT	Rate of Success
5	2/5
10	5/5
15	4/5
20	3/5

Table 9.2. Effect of TT with discounted learning

TT	Rate of success	Avg. no of iterations
1	9/10	-
2	10/10	599
3	10/10	584
4	10/10	664
5	10/10	946
6	10/10	1189
7	10/10	1082
8	10/10	1372
9	10/10	1606
10	10/10	2367
11	10/10	1832
12	10/10	3152
13	10/10	2095
14	10/10	4040
15	10/10	2041
16	10/10	3996
17	8/10	-
20	6/10	-

For the tests in Section 6 a short, fixed tabu tenure of value 4 was used. This choice avoids stumbling back in the previously visited search path, given that the local neighborhood evaluations typically consist of many equal values and forms a local plateau, even when learning and forgetting is employed. The value of the tabu tenure does not appear sensitive to changes within a modest range. The tabu mechanism thus has a very local function, while the learning and forgetting provides the necessary diversification.

5.4 Probabilistic Move Acceptance

The only diversification present in our SC guided search is the random selection between the moves that tie for the best evaluation. More than one move may tie for this, but many fewer ties occur when the learning and

forgetting mechanisms are employed than when only the SC evaluations are used. This leads to less diversification in the search.

Since our guidance is not perfect, there is no guarantee that the actual best move gets the best evaluation. Assuming our move evaluations have some merit, we select probabilistically among the best moves, disproportionately favoring those with higher evaluations (see Løkketangen and Glover, 1996). The process selects the best move with probability p , and if the move is rejected then applies the same selection rule to the next best, etc. This gives rise to an exponentially decreasing move acceptance.

5.5 Choice of Starting Solution

We ran preliminary tests comparing starting from randomly generated truth assignments to the variables, and the best solution found after 10 iterations of the constructive search with learning as described in Løkketangen and Glover (1997). With 10 runs on each problem, we did not find any significant difference between the results. One reason why the constructed starting point did not yield better results might be that the different solutions from the constructive phase for the same problem tend to be similar, such that bad trends might be amplified.

5.6 A New Method – SC-RN

Our full method with learning and forgetting spend a lot of time in maintaining the adaptive memory structures.

Move evaluation is very costly in the SC Learn method with added forgetting. The problem is the “forgetting” part, as this increments the learning weights. This requires a full evaluation of the SC’s are after every flip. (Preliminary testing using incremental update, ignoring the slightly changed weight values, gave very bad results).

Considering that in every unsatisfied clause at least one of the variables has the wrong value, we form a reduced neighbourhood consisting only of the variables in the unsatisfied clauses. The new method can be labelled SCLFPRN – *Surrogate Constraint based move evaluation with Learning and Forgetting, Probabilistic move acceptance and Reduced Neighborhood*. For short we call it SC – RN.

6. Computational Results

We investigate the tradeoff issues when applying different levels of adaptive memory mechanisms, and also when using guidance based on surrogate constraints for various neighborhood sizes. The testing reported in this section is intended to illustrate these tradeoffs. For purposes of

comparison we chose three heuristics. As a representative for a method using rather myopic search guidance, but with very fast iterations, we chose Novelty (McAllester, Selman and Kautz, 1997), hereafter called *NOV*. The iterative SC-Learn heuristic (Løkketangen and Glover, 1997) was augmented with forgetting and probabilistic move acceptance and called *SC-W*. As an intermediary we used the method described in Section 5.6, *SC-RN*, as it uses a smaller neighbourhood than *SC-W*, thus basing its decisions on less information.

6.1 Test Setup

Testing has been done on benchmark SAT-problems, both structured and randomized, taken from SATLIB (see SATLIB). The problem sizes range from 48 variables * 400 clauses to 4713 variables * 21991 clauses.

As all the methods include a probabilistic parameter, p , we chose one of the simpler test cases from SATLIB, *jnh*, to tune this parameter individually for each heuristic, and keep it fixed for all subsequent tests. The best values (and thus the values used) for p are shown in Table 9.3.

Table 9.3. Best values for p

	<i>p</i>
SC-W	0.8
SC-RN	0.6
NOV	0.8

We tested each heuristic from random starting solutions, using different random seeds, with 10 runs per test case. We allowed in general up to 10^7 iterations for Novelty, 10^6 for SC-RN and 10^5 for SC-W. Restarts were not used. A fixed TT of 4 was used. The results are both in term of flips and time. The tests have been run on a 300 MHz Pentium III running Windows 98 (The sub-second timing is thus somewhat inaccurate).

6.2 Test Results

The results are shown in Tables 9.4, 9.5 and 9.6. For each test case is shown the size (in terms of variables and clauses), and the average number of flips and the corresponding time used for each of the three heuristics.

A dash in the flips column indicates that the heuristic failed to find a solution for at least one of the tries, with the actual number of successful runs are shown in the flips column.

Table 9.4 shows the results for some of the smaller test cases. Not surprisingly NOV is very good on the random 3-SAT instance *uf100-100*, spending virtually no time in finding the solution. SC-RN spends more flips

and time, while SC-W is comparable with NOV in terms of flips, but not on time.

The rest of the test cases in the table are structured, par-8-1 is a parity problem encoding, and par-8-1-c is a compressed version of the same problem. The ais test cases are taken from a problem in music theory (all interval series). On these problems SC-W is clearly most successful, only having 2 unsuccessful runs, while NOV does rather badly.

Table 9.4. Smaller test-cases

Problem	vars	clauses	NOV	NOV	SC-RN	SC-RN	SC-W	SC-W
			f	t	f	t	f	t
<i>max-flips</i>			10^7		10^6		10^5	
uf100-100	100	430	371	0	1090	0.1	397	0.01
par8-1-c	64	254	1149	0	4484	0.5	1070	.13
par-8-1	350	1149	1/10	-	8/10	-	9/10	-
ais6	61	581	0/10	-	3326	0.1	465	0.1
ais8	265	5666	0/10	-	187850	9.5	6269	1
ais12	1141	10719	0/10	-	0/10	-	9/10	-

Table 9.5. Intermediate test-cases

Problem	vars	clauses	NOV	NOV	SC-RN	SC-RN	SC-W	SC-W
			f	t	f	t	f	t
<i>max-flips</i>			10^7		10^6		10^5	
uf200-100	200	860	8860	0.2	64608	0.1	397	0.1
flat100_3_0	300	1117	9060	0.2	5/10	-	3882	1.5
sw100-10-p4	500	3100	2/10	-	89502	3.7	11075	10
sw100-10-p6	500	3100	2/10	-	5/10	-	4/10	-
anomaly	48	261	124	0	164	0	135	0
medium	116	953	852	0.1	331	0	445	0.1

Table 9.5 shows the next set of results, for small to intermediate test cases. Interestingly, SC-W uses many fewer flips on the random 3-SAT instance uf200-100 than NOV, even being faster. In general SC-W needs fewer flips on most of the instances. NOV does reasonably well on these instances, while SC-RN is slightly worse.

Table 9.6. Large structured test-cases

Problem	vars	clauses	NOV	NOV	SC-RN	SC-RN	SC-W	SC-W
			f	t	f	t	f	t
<i>max-flips</i>			10^7		10^6		10^5	
bw_large.a	459	4675	60299	1.4	5571	1	5132	6.5
bw_large.b	1087	13772	4.79 M	105	31048	10.4	56611	198
logistics.a	828	6718	0/10	-	2/10	-	18922	50
logistics.b	843	7301	0/10	-	102465	14	23565	50
logistics.c	1141	10719	0/10	-	91739	59	21248	103
logistics.d	4713	21991	0/10	-	88091	26	72600	470

In Table 9.6 are results for the runs on large structured instances. SC-W seems very good on these, while NOV fails on the logistics instances. SC-RN does quite well, only failing on logistics.a.

As can be seen from the results, SC-W solves most test cases, but spends a long time per iteration. It seems particularly good on the large structured instances. This is according to expectations, as the learning mechanisms learn *structure*.

SC-RN needs more flips, and is more unstable. It solves most structured instances, while showing bad performance on some random problem classes. Each iteration is much faster than SC-W. This is the behaviour we would expect, as it bases its search guidance on a smaller neighbourhood, and thus less information, than SC-W.

NOV fails on bigger instances, and on instances having a lot of structure. This is not very surprising, as NOV does not have any memory mechanisms to capture structure. It does use recency information, but only within a clause, and then only to choose between the two (locally) best variables. In clauses with only 3 variables, this seems to work very well, but NOV clearly has problems with problems having longer clauses and problems with structure.

7. Conclusions

The heuristics we describe rely on a set of advanced mechanisms for their working. Testing clearly shows that care should be taken when combining mechanism, often necessitating changes in their customary settings.

The computational testing clearly illustrates that the use of surrogate constraints provides good guidance. The addition of simple learning gives greatly improved results. Discounted learning (forgetting) is effective, and has a stabilizing effect. As is expected, best results are obtained for the structured test cases. The extra cost in maintaining memory and guidance structures thus seems well spent on several classes of test instances.

Appropriate guidance structures based on surrogate constraint evaluations, incorporating adaptive memory guidance based on recency, frequency, learning and forgetting – thus yield results that compare favourably with state-of-the-art randomized local search heuristics for SAT.

References

- Cook, S.A. (1971) "The Complexity of Theorem-Proving Procedures," *Proceedings of the Third ACM Symposium on Theory of Computing*. 151–158.
- Davis, M., G. Logemann and D. Loveland (1962) "A Machine Program for Theorem Proving," *Comm. ACM*, 5:394–397.
- Davis, M. and H. Putnam (1960) "A Computing Procedure for Quantification Theory," *Journal of ACM*, 7:201–215.
- Frank, J. (1996) "Weighting for Godot: Learning Heuristics for Gsat," *Proceedings of the 13th International Conference on Artificial Intelligence*, 338–343.
- Hoos, H. (1998) "Stochastic Local Search - Methods, Models, Applications," Ph.D. Dissertation, Fachbereich Informatik, Technische Universität Darmstadt.
- Glover, F. (1977) "Heuristics for Integer Programming using Surrogate Constraints," *Decision Sciences* 8:156–166.
- Glover, F. and M. Laguna (1997) *Tabu Search*. Kluwer Academic Publishers.
- Gomes, C., B. Selman, K. McAlloon and C. Tretkoff (1998) "Randomization in Backtrack Search: Exploiting Heavy-Tailed Profiles for Solving Hard Scheduling Problems". In Proceedings AIPS-98.
- Gomes, C., B. Selman and H. Kautz (1998) "Boosting Combinatorial Search Through Randomization," In Proceedings AAAI98.
- Løkketangen, A. and F. Glover (1997) "Surrogate Constraint Analysis—New Heuristics and Learning Schemes for Satisfiability Problems," *Satisfiability Problem: Theory and Applications. DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 35:537–572.
- Løkketangen, A. and F. Glover (1996) Probabilistic Move Selection in Tabu Search for 0/1 Mixed Integer Programming Problems. *Metaheuristics: Theory and Applications*, Kluwer Academic Publishers, 467–489.
- McAllester, D., B. Selman and H. Kautz (1997) "Evidence for Invariants in Local Search," In Proceedings AAAI97.
- Mazure, B., L. Saïs and É. Grégoire (1997) Tabu Search for SAT. In Proceedings AAAI 97.
- Resende, M. and T. Feo (1996) "A GRASP for Satisfiability," in *Cliques, Coloring and Satisfiability. The Second DIMACS Implementation Challenge*, D.S. Johnson and M.A. Trick (eds.), *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, 26:499–520, American Mathematical Society.
- SATLIB – The Satisfiability Library.
<http://www.intellektik.informatik.tu-darmstadt.de/SATLIB/>
- Selman, B., H.J. Levesque and D. Mitchell (1992) "A New Method for Solving Hard Satisfiability Problems," *Proceedings AAAI 92*, 440–446.

Chapter 10

LESSONS FROM APPLYING AND EXPERIMENTING WITH SCATTER SEARCH

Manuel Laguna¹ and Vinícius A. Armentano²

¹*Leads School of Business, University of Colorado, Boulder, CO 80309-0419, USA,
laguna@colorado.edu*

²*Faculdade de Engenharia Eletrica e de Computacao, C. P. 6101, Campinas, SP 13083-970,
Brazil, vinicius@densis.fee.unicamp.br*

Abstract: Scatter search is an evolutionary method that has been successfully applied to hard optimization problems. The fundamental concepts and principles of the method were first proposed in the 1970s and were based on formulations, dating back to the 1960s, for combining decision rules and problem constraints. The method uses strategies for search diversification and intensification that have proved effective in a variety of optimization problems. This paper presents a number of findings (lessons) from the application of scatter search to combinatorial optimization problems (e.g., production scheduling and the linear ordering problem) and nonlinear optimization problems (e.g., multi-modal functions and neural network training). We describe our findings and the context in which we have learned each lesson. We believe that some of our findings are not context specific and therefore may benefit future applications of scatter search.

Keywords: Scatter Search, Metaheuristics, Optimization

1. Introduction

Scatter search (SS) was first introduced in 1977 (Glover 1977) as a heuristic for integer programming and it was based on strategies presented at a management science and engineering management conference held in Austin, Texas in September of 1967. The scatter search methodology has evolved since it was first proposed and hence we devote the rest of this section to discuss this evolution.

1.1 Original Description (1977)

The following is a slightly modified version of the description of scatter search in Glover (1977), which is the first published description of this methodology. Scatter search uses a succession of coordinated initializations to generate solutions. The solutions are purposefully (i.e., non-randomly) generated to take into account characteristics in various parts of the solution space. Scatter search orients its explorations systematically relative to a set of reference points. Reference points may consist, for example, of the extreme points of a simplex obtained by truncating the linear programming basis cone, or of good solutions obtained by prior problem solving efforts.

The approach begins by identifying a convex combination, or weighted center of gravity, of the reference points. This central point, together with subsets of the initial reference points, is then used to define new sub-regions. Thereupon, analogous central points of the sub-regions are examined in a logical sequence (e.g., generally sweeping from smaller objective function values to larger ones in a minimization problem). Finally, these latter points are rounded to obtain the desired solutions. (Rounding, for any problems other than those with the simplest structures, should be either an iterative or a generalized adjacency procedure to accommodate interdependencies in the problem variables.)

The 1977 scatter search version is depicted in Figure 10.1. Each of the points numbered 1 through 16 is the central point of an apparent sub-region of the simplex A, B and C. Point 8 is the center of (A, B, C) itself. In this example, A, B and C may not constitute the original reference points (which could, for example, have been 6, 7 and 11 or 4, 5, 12 and 13). The choice depends on the distribution of the points relative to each other and the feasible region generally. Thus, for example, it can be desirable to use envelopes containing derived reference points, and to use weights that bias central points away from central points. When scatter search is carried out relative to reference points that lie on a single line, then it is reduced to a form of linear search.

In general, to keep the effort within desirable limits for larger dimensions, the points generated according to the example pattern may be examined in their indicated numerical sequence (or in an alternative sequence dictated by the slope of the objective function contour) until a convenient cutoff point is reached. Or one may examine a less refined collection of points.

The description terminates indicating that because scatter search may be applied to reference points obtained from a historical progression of solutions attempts and may also be used to influence this progression; the approach is conveniently suited to application with learning strategies.

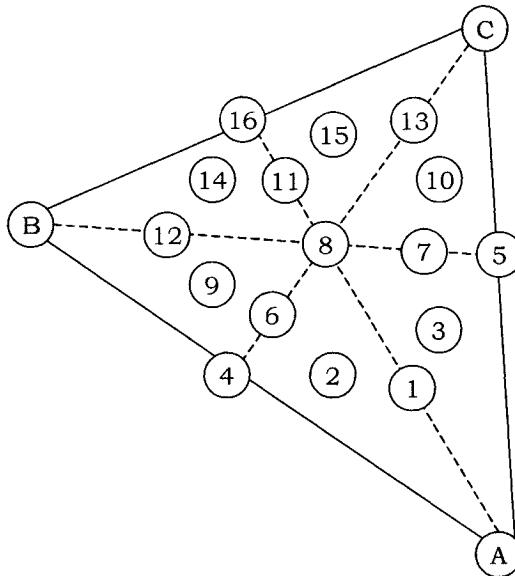


Figure 10.1. Illustrative example of the 1977 scatter search proposal

The following comments apply to this description of scatter search:

1. The approach focuses on combining more than two solutions to generate central points. In fact, finding the central point of all the reference points is considered a focal part of the search.
2. Generating solutions in a line, which is the most common strategy in modern scatter search implementation, is considered a reduced form of the more general method
3. There is no reliance on randomization, but no guidelines are given to select “appropriate weights” to find biased central points.
4. Combinations are described to be convex, although non-convex combinations are implicit in the illustrative example to make it possible to generate all the points in Figure 10.1 starting from points 6, 7 and 11
5. The distribution of points relative to each other is considered important but no mechanism is discussed to encourage such dispersion.

To the best of our knowledge, scatter search was never applied or discussed again until 1990, when it was presented at the EPFL Seminar on Operations Research and Artificial Intelligence Search Methods (Lausanne, Switzerland). An article based on this presentation was published later in 1994 (Glover 1994).

1.2 Scatter / Tabu Search Hybrid (1990-1994)

This description of scatter search provides implementation details that were not given in the original proposal. Also in Glover (1994), the range of application is expanded to nonlinear (continuous) optimization problems, binary and permutation problems. The procedure is coupled with tabu search (TS), using forms of adaptive memory and aspiration criteria to influence the selection of points in a reference set consisting of several subsets. The method is described as one that generates “systematically dispersed set of points from a chosen set of reference points.” The concept of weighted combinations is introduced as the main mechanism for generating new trial points on lines that join reference points. This version of scatter search emphasizes line searches and the use of weights to sample points from the line. The following specific elements are introduced:

1. The method starts from a set S of initial trial points from which all other sets derive. The generation of S is not proposed to be random.
2. The current reference set is defined as $R = H^* \cup S^*$, where H^* consists of h^* best solutions in $H-T$ and S^* consists of the s^* best elements of S . H is the set of the best h solutions generated historically throughout the search. T is a subset of solutions in H that are excluded from consideration.
3. An iteration consists of first generating the center of gravity for all the solutions in S^* and also the centers of gravity for all the subsets of size s^*-1 in S^* . These trial points are then paired with solution in R to create search lines for generating additional trial points. Elements of S^* are also paired with each other for the same purpose. Finally, the following solutions are also paired $H^* \times D(S^*)$ and $R \times D(S-S^*)$, where $D(X)$ denotes a diverse subset of set X .
4. When two solutions x and y are paired, trial solutions are generated with the line $z(w) = x + w(y - x)$, for $w = 1/2, 1/3, 2/3, -1/3$ and $4/3$.
5. When the quality of a trial solution z exceeds the average quality of the solutions in S^* , the search intensifies around z . The intensification consists of generating mid points of lines that join z with its nearest neighbors (i.e., trial points found during the line search that generated point z).
6. Adaptive memory of the recency and frequency type is used to control the admission of solutions to H and T .

This version of scatter search relies on line searches that start from central points. That is, the central points are generated as anchor points to initiate line searches using current and historical best solutions. The concept of combining high quality solutions with diverse solutions is also introduced.

The method includes an intensification component, consisting of sampling more solutions from a line that has produced a “better-than-average” solution.

The concept of *structured weighted combinations* is also introduced to handle discrete optimization problems directly. This allows the representation of solutions that is natural to specific problems, such as permutations used in the context of traveling salesperson and job sequencing problems. Binary problems, like the knapsack, can be handled also with structured weighted combinations that are based on three properties:

1. *Representation property* — Each vector represents a set of votes for particular decisions (e.g., the decision of putting element j after element i in a permutation).
2. *Trial solution property* — A well-defined process translates a set of votes prescribed by a vector into a trial solution to the problem of interest.
3. *Update property* — If a decision is made according to the votes of a given vector, a clearly defined rule exists to update all vectors for the residual problem so that both the representation property and the trial solution property continue to hold.

Several voting mechanisms with the aforementioned properties are suggested in connection with permutation problems as well as an adaptive procedure to dynamically change the associated weights. Although all these ideas were illustrated with several examples, including permutation and binary problems as well as a graph-partitioning problem, this version of scatter search has been implemented and tested only in the context of scheduling job on identical parallel machines in order to minimize total weighted tardiness (Mazzini 1998). We discuss some of the lessons from this application in Section 2.

The scatter/Tabu search hybrid discussed in this section has served as the foundation for the search strategies that can be found in many of the recent scatter search implementations.

1.3 Scatter Search Template (1997)

This version of scatter search (Glover, 1997) is in some ways a simplification of the method proposed in Glover (1994) and discussed in the previous subsection. The template employs five procedures:

1. A *diversification generation method* to generate a collection of diverse trial solutions, using an arbitrary trial solution (or seed solution) as an input.

2. An *improvement method* to transform a trial solution into one or more enhanced trial solutions. (Neither the input nor output solutions are required to be feasible, though the output solutions will more usually be expected to be so. If no improvement of the input trial solution results, the “enhanced” solution is considered to be the same as the input solution.)
3. A *reference set update method* to build and maintain a reference set consisting of the b best solutions found (where the value of b is typically small, e.g., approximately 20), organized to provide efficient accessing by other parts of the method.
4. A *subset generation method* to operate on the reference set, to produce a subset of its solutions as a basis for creating combined solutions.
5. A *solution combination method* to transform a given subset of solutions produced by the subset generation method into one or more combined solution vectors.

The template is summarized as follows:

Initial Phase

1. (*Seed Solution Step*) Create one or more seed solutions, which are arbitrary trial solutions used to initiate the remainder of the method.
2. (*Diversification Generation Method*) Use the diversification generation method to generate diverse trial solutions from the seed solution(s).
3. (*Improvement and Reference Set Update Methods*) For each trial solution produced in Step 2, use the improvement method to create one or more enhanced trial solutions. During successive applications of this step, maintain and update a reference set consisting of the b best solutions found.
4. (*Repeat*) Execute Steps 2 and 3 until producing some designated total number of enhanced trial solutions as a source of candidates for the reference set.

Search Phase

5. (*Subset Generation Method*) Generate subsets of the reference set as a basis for creating combined solutions.
6. (*Solution Combination Method*) For each subset X produced in Step 5, use the solution combination method to produce a set $C(X)$ that consists of one or more combined solutions. Treat each member of $C(X)$ as a trial solution for the following step.

7. (*Improvement and Reference Set Update Methods*) For each trial solution produced in Step 6, use the improvement method to create one or more enhanced trial solutions, while continuing to maintain and update the reference set.
8. (*Repeat*) Execute Steps 5-7 in repeated sequence, until reaching a specified cutoff limit on the total number of iterations.

More algorithmic details associated with this template can be found in Laguna and Martí (2003). The diversification generation methods suggested in this template emphasize systematic generation of diversification over randomization. This theme remains unchanged since the original proposal. In this version, however, an explicit reference to local search procedures is made. The improvement method expands the role of the mapping procedure originally suggested to transform feasible or infeasible trial solutions into “enhanced” solutions. Another change is the updating of the reference set, which is greatly simplified when compared to the SS/TS version in Glover (1994). The reference set in the template simply consists of the b best solutions found during the search. The subset generation method is a departure from the original proposal. The method emphasizes the linear combination of two solutions. Combining more than two solutions is also suggested, but the lines searches are not “anchored” at the central points of sub-regions (as suggested in the previous two versions).

The publication of this template for scatter search triggered the interest of researchers and practitioners, who translated these ideas into computer implementations for a variety of problems. Refer to Glover, Laguna and Martí (2000) for a list of scatter search implementations based on this template. These implementations have modified the template in a number of ways and offer lessons for future implementations. The remaining of this paper discusses changes and additions to the scatter search methodology and the lessons we have learned when developing procedures based on this framework.

2. Diversification Generation

The first set of lessons relate to the method for generating diversification within scatter search. The diversification generation method is typically used to initialize the reference set and also to rebuild the reference set during the search.

Lesson 1: Consider the use of frequency memory to develop effective diversification generation methods

Campos et al. (2001) developed 10 diversification generators for a scatter search implementation tackling the linear ordering problem (LOP). Solutions to the LOP can be represented as permutations. The first six generators (labeled DG01-DG06) were based on GRASP constructions. Therefore, these methods use a greedy function to measure the attractiveness of selecting an element for a given position in the permutation. The selection is random among a short list of attractive choices. The seventh procedure (DG07) is a hybrid combination of the first six. The eighth procedure (DG08) simply constructs random permutations. The ninth (DG09) is an implementation of the systematic generator for permutations in Glover (1997). The last procedure (DG10) constructs permutations using a penalized greedy function. The penalties are based on a frequency memory that keeps track of the number of times an element has occupied a position in the permutations previously generated.

To test the merit of each diversification generator, two relative measures were used: 1) a measure of relative diversity Δd and 2) a measure of relative quality ΔC . Figure 10.2 shows the performance of the diversification generators as measured from the point of view of creating diversity while maintaining solution quality.

We can observe in Figure 10.2 that DG10 has the best balance between quality and diversity. The pure random method DG08, on the other hand, is able to generate highly diverse solutions (as measured by Δd) but at the expense of quality. The use of frequency memory and a guiding function provides the desired balance exhibited in DG10.

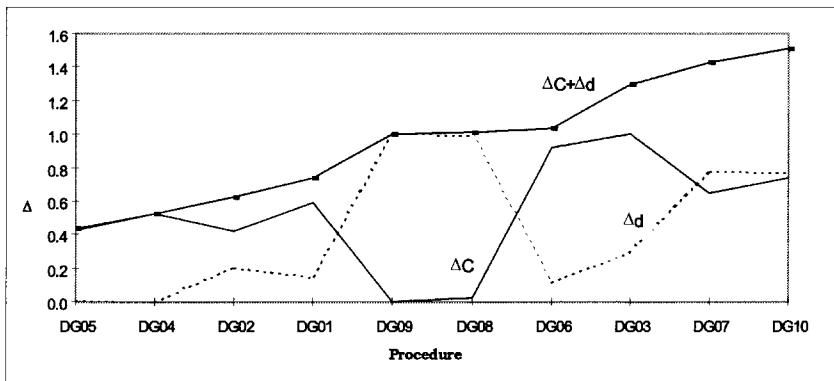


Figure 10.2. Diversity vs. quality for alternative diversification generation methods

We have found that even those diversification methods that include random elements benefit from the use of frequency memory. Glover, Laguna and Martí (2000) implemented scatter search for unconstrained non-linear optimization problems with bounded variables. The diversification generator uses frequency memory and controlled randomization as follows. The range of each variable is divided into 4 sub-ranges of equal size. Then, a solution is constructed in two steps. First a sub-range is randomly selected. The probability of selecting a sub-range is inversely proportional to its frequency count. Then a value is randomly generated within the selected sub-range. The number of times a sub-range has been chosen to generate a value for a given variable is accumulated in an array. This frequency-based generator was also shown superior to alternative designs tested in the same context.

Lesson 2: Consider initializing the reference set from a large set of solutions generated with the diversification generation method

The initial phase of the scatter search template suggests a dynamic building of the first reference set. In other words, the diversification generation method is applied (followed by the improvement method) until b distinct solutions are generated. Unfortunately, this procedure does not fully exploit the benefits of a diversification generation method that is based on frequency memory. To take advantage of such a method, we suggest building a large set of solutions first. Refer to this set as P and to its corresponding size as p . The set is built by applying the diversification generation method followed by the improvement method until p solutions are generated. The reference set R is built by choosing b solutions from P . (We discuss selection rules for choosing solutions for R in the lessons related to the reference set update method.) Typically the size of the set P is such that $p \geq 10^*b$.

3. Improvement Method

Most improvement methods in the context of scatter search consist of a simple local search. One should keep in mind that the improvement method must be capable of starting its descent (in the case of minimization) from an infeasible solution. The improvement method is typically applied to trial solutions generated with either the diversification generation method or the combination method. These methods could operate in a way that the resulting trial solutions might be infeasible and hence the need for an improvement method capable of starting from an infeasible solution. Key

issues related to the improvement method are the frequency of its application and the depth of the associated search.

Lesson 3: *Investigate the selective use of the improvement method instead of applying it to every trial solution*

Ugray, Lasdon and Plummer (2000) develop a solution procedure for large-scale constrained nonlinear optimization problems. The procedure is based on combining a scatter search with a large-scale GRG optimizer. LSGRG uses solutions generated with scatter search as the initial points for finding local optima. The initial solution may not be feasible (with respect to linear or nonlinear constraints), but LSGRG is designed to initiate the search from either infeasible or feasible points. Because an application of LSGRG could be computational expensive, this optimizer is not applied to every solution generated during the scatter search. In this context, the scatter search must be calibrated to generate diverse initial points for a LSGRG exploration in order to avoid visiting the same local optima repeatedly.

Mazzini (1998) found, in the context of scheduling jobs on parallel identical machines, that an improvement method consisting of searching for the best move from either all possible swaps or insertions is computationally costly. To reduce the complexity, the improvement method was limited to examining a reduced list of candidate moves. Even with this limitation, the computational effort was only reduced when the improvement method was selectively applied. In particular, the improvement method to the trial solutions generated every 10 iterations to solutions in $D(H \cup S)$. More sophisticated selection rules were also tried without significantly improving the results.

Lesson 4: *The application of the improvement method to all trial solutions accelerates the convergence of the reference set*

In some situations, such as the scatter search implementations of Campos et al. (2001), Corberán et al. (2002), and Laguna and Martí (2000, 2002), the improvement method is such that it can be feasibly applied to all trial solutions. Although this is a desirable feature from the point of view of finding high quality solutions quickly, the reference set might “prematurely converge” (where convergence here means reaching a state in which no new solutions are admitted to the set). This quick convergence is not a limitation and in fact it might actually be desirable when the allotted searching time is short (e.g., in the context of optimizing simulations). Reference set

convergence can also be resolved by rebuilding the set as described in lesson 7.

In Campos et al. (2001) the improvement method is used to enhance every trial solution. Because of the premature convergence, a reference set size that is larger than customary was tried. In particular, two instances of scatter search were used for experimentation, one with $b = 20$ and $p = 100$ and the other one with $b = 40$ and $p = 400$. The procedure stops when no new solutions are admitted into the reference set and therefore the length of the search depends on the size of the reference set. This design resulted in superior outcomes when compared to one where a small ($b = 10$) reference set was rebuilt several times. The best solutions were found with $b = 40$ at the expense of additional computational time.

Lesson 5: *Perform experiments to find the right balance between the computational time spent generating trial solutions and the time spent improving such solutions*

This is one of the most difficult issues in the design of scatter search procedures that include an improvement method. Most improvement methods consist of a local search procedure that stops as soon as no more improving moves are found. The selection rule for choosing an improving move typically varies between selecting the first improving move and selecting the most improving move. In general no mechanism for breaking out of local optima is used, but this is an ideal place to hybridize scatter search with procedures such as a simple short-term memory tabu search.

4. Reference Set Update Method

Some of the most important lessons that we have learned relate to the way the reference set is initialized, updated, and rebuilt. We have experienced significant changes in the level of performance depending on the strategies used to update the reference set. These are our lessons.

Lesson 6: *Initialize the reference set with half of the set consisting of high quality solutions and the other half with diverse solutions*

The right balance between intensification and diversification is a desired feature of search procedures in general and scatter search in particular. The updating of the reference set can be used to incorporate an appropriate balance in scatter search implementations. In lesson 2 we discussed the use of a set P to initialize the reference set. If the initialization of the reference set were made solely on the basis of solution quality, the set would be

constructed selecting the best (in terms of quality) b solutions from P . If instead, we want to balance quality and diversity, we can use the following initialization procedure for a reference set R of size $b = b_1 + b_2$:

1. Start with $P = \emptyset$. Use the diversification generator to construct a solution s . Apply the improvement method to s to obtain the enhanced solution s^* . If $s^* \notin P$ then, add s^* to P (i.e., $P = P \cup s^*$), otherwise, discard s^* . Repeat this step until $|P| = p$.
2. Order the solutions in P according to their objective function value (where the best overall solution is first on the list).
3. Select the first b_1 solutions in P and add them to R .
4. For each solution x in $P - R$, calculate the minimum dissimilarity $d(R, x)$ to all solutions in R .
5. Select the solution x^* with the maximum dissimilarity $d(R, x^*)$ of all x in $P - R$.
6. Add x^* to R and delete x^* from P . If $|R| = b$ stop, otherwise go to 4.

This initialization procedure uses a max-min criterion to select diverse solutions to be added to the reference set. In particular, it selects the solution that maximizes the minimum value of a dissimilarity measure between the candidate solution and the solutions currently in the reference set. The dissimilarity measure d depends on the problem context. For example, the Euclidean distance can be used in problems whose solution representation is a vector of continuous variables.

Lesson 7: *Rebuild the reference set with the diversification generation method*

The search might reach a point where the reference set converges, that is, it does not change anymore because no trial solution has better quality than the worst solution in R . At this point the search could be terminated or the reference set rebuilt. The diversification generator is a useful method for rebuilding the reference set. Assume that the reference set is ordered in such a way that the first solution is the best in terms of quality. In order to rebuild it, we can keep the first b_1 solutions and use the diversification generation method to add b_2 solutions to the reference set. The solutions are added using the max-min criterion described in the previous lesson.

Lesson 8: *Solution quality is more important than the diversity of the solutions when updating the reference set*

During the search phase, the reference set is updated according to the quality of trial solutions. If the quality of a trial solution exceeds the quality of the worst solution in the reference set, then the trial solution is added to R and the worst solution is deleted from R . Laguna and Martí (2000) experimented with a reference update method that essentially carried two reference sets: 1) R_1 consisting of the best (according to quality) b_1 solutions and 2) R_2 consisting of the b_2 most diverse solutions. The diversity of the solutions in R_2 was measured with the max-min criterion presented in lesson 6, with $R = R_1 \cup R_2$. This updating mechanism did not yield good outcomes because it places similar importance to both quality and diversity. If the initialization and rebuilding mechanisms from lessons 6 and 7, respectively, are used, then there does not seem to be a need for additional diversification at the expense of quality during the updating of R .

Laguna and Martí (2000) also tested a reference update method that divides R into three subsets: a “quality” subset, a “diverse” subset and a “historically good generator” subset (R_3). This three-tier reference set has a similar structure to the one proposed in Glover (1994). The update is such that solutions that are deleted from the quality set R_1 are tested for admission in R_3 . This reference set configuration and updating method was shown inferior to the one that updates the reference set with “quality” solutions only and rebuilds half of it with diverse solutions if necessary.

Lesson 9: Compare the merit of static versus a dynamic updating method

There are basically two ways of updating the reference set: static and dynamic. The static update operates as follows. The combination method is applied to all the subsets generated with the subset generation method in order to create the set X of all trial solutions. The new reference set then consists of the best b solutions from $R \cup X$. This updating mechanism guarantees that a reference solution is used at least once in the combination method.

Alternatively, the reference set may be dynamically updated. The dynamic update consists of testing trial solutions for admission to R as these solutions are generated. Let $R = \{x_1, x_2, x_3\}$ be the initial reference set, where x_1 is the best solution and x_3 is the worst. Suppose that x_1 and x_2 are combined and that this combination generates a trial solution z . If the quality of z is better than the quality of x_3 , under the dynamic updating method, x_3 is immediately eliminated from R and z is added to R . This means that x_3 is not given the opportunity to generate trial solutions. The dynamic updating is more aggressive in terms of incorporating high quality solutions into R faster. However, this aggressiveness may not necessarily translate into improved outcomes.

The appealing feature of the dynamic updating is that at the beginning of the search, low quality solutions are immediately replaced with higher quality solutions and thus making the method more aggressive. Starting the search with a reduced reference set size and using a static update has similar effects. Suppose that the desired reference set size is $b = 15$. To make the search more aggressive within a static updating method, the procedure could start with $b = 10$. The number of trial solutions generated with the initial reference set is reduced and the poor solutions in the reference set are replaced faster. After an initial phase, the size of the reference set can be increased when no new solutions are admitted to the set. This monotonic increase could stop after reaching the target size of 15.

5. Subset Generation Method

This procedure consists of creating different subsets X of R , as a basis for applying the combination method. The procedure seeks to generate subsets that have useful properties, while avoiding the duplication of subsets previously generated. The most common approach for doing this is organized to generate four different collections of subsets of R with the following characteristics:

- all 2-element subsets
- 3-element subsets derived from the 2-element subsets by augmenting each 2-element subset to include the best solution not in this subset
- 4-element subsets derived from the 3-element subsets by augmenting each 3-element subset to include the best solutions not in this subset
- the subsets consisting of the best i elements, for $i = 5$ to b

Lesson 10: *Most of the searching power can be attributed to the combination of 2-element subsets*

Campos et al. (2001) describe an experiment designed to assess the contribution of the different types of combinations embedded in an implementation of scatter search for the linear ordering problem. The goal of the experiment was to determine whether the best solution was generated from the combination of 2, 3, 4 or all b reference solutions. That is, the experiment attempted to identify how often, across a set of benchmark problems, the best solutions came from various combinations of k reference solutions. The experiment showed that at least 80% of the solutions that were admitted to the reference set came from combinations of 2-element subsets.

Our observations and the results of the experiments in Campos et al. (2001) should not be interpreted as a justification for completely disregarding the use of combinations other than those from 2-element subsets. What we learned from this experience is that 2-element subsets do most of the work; however, we also observed that these percentages could change if the subset types were generated in a different sequence.

6. Combination Method

The solution combination method is an element of scatter search that is context-dependent. Although it is possible to design “generic” combination procedures, it is generally more effective to base the design on specific characteristics of the problem setting.

Lesson 11: *The best solutions are often generated by the combination of other high quality solutions*

. Campos et al. (2001) performed an experiment to identify the ranks of the reference solutions that generate the best solutions during the search. The experiment considered that the reference set was ordered with the solution in the first rank being the best overall solution. For this experiment, a $b \times b$ matrix $Source(i,j)$ was created, where $Source(i,j)$ counted the number of times a solution of rank j was a reference point for the current solution with rank i . The experiment showed that solutions of rank 1 are more often generated by combinations that involve other solutions of rank 1.

This lesson tells us that we should consider generating more trial solutions when combining the best solutions in the reference set than when combining the worst solutions. The following combination method illustrates this in the context of nonlinear optimization:

Let x and y be two solutions in the reference set that are used to perform three types of combinations to generate a trial solution z :

$$C1: z = x - d$$

$$C2: z = x + d$$

$$C3: z = y + d$$

where $d = 0.5r(y - x)$ and r is a number between 0 and 1. The combination rules are:

- If both x and y rank among the first b_1 solutions in R , then generate 4 solutions by applying C1 and C3 once and C2 twice.

- If either x or y ranks among the first b_1 solutions in R , then generate 3 solutions by applying C1, C2 and C3 once.
- If neither x nor y ranks among the first b_1 solutions in R , then generate 2 solutions by applying C2 once and either C1 or C3 once.

Note that when a combination is applied twice, different values for r must be selected.

Lesson 12: *Combination methods that include some randomness can be effective*

The combination mechanism illustrated in the previous lesson contains a parameter r that determines the combination weight and that ranges between 0 and 1. This combination method is similar to the one suggested in Glover (1994) and listed as the fourth element in the Scatter/Tabu Search Hybrid description in Section 1. The difference between the procedure above and the one suggested in Glover (1994) is that Glover prescribes five values for the weight parameter w , making the method completely deterministic. By allowing r to be a random number between 0 and 1, the combination method can be applied to the same solutions with different outcomes. This flexibility can be used for intensification purposes, where more solutions can be sampled from a promising line with the same mechanism.

Lesson 13: *The use of multiple combination methods can be effective*

Genetic algorithms (GA) have been particularly good at exploiting this lesson. Most GA implementations use several “operators” to combine parents and generate new trial solutions. The rules to select from the set of available operators vary from one implementation to another. An analogous process can be implemented within the scatter search framework. In Chapter 12, Martí, Laguna and Campos have used this notion within a scatter search implementation for permutation problems, where 7 ways of combining permutations are used.

Adaptive structured combination methods provide a mechanism for implementing multiple forms of combinations within a single framework. In this approach, each reference solution proposes votes for a particular attribute to be incorporated in the new trial solution. The voting scheme is adaptive because a new trial solution is constructed with the sequential addition of the attributes that have obtained the most votes. Variations of the voting scheme result in different ways of combining solutions (which in itself can be considered as creating multiple combination methods or “operators”). Adaptive structured combinations have been implemented in

scatter search procedures for production scheduling (Mazzini 1998; Laguna et al. 2000), project scheduling (Valls et al. 1998) and linear ordering problems (Campos et al. 2001), among others.

7. Conclusions

In this paper, we first discussed the evolution of the scatter search methodology from its original proposal published in 1977 to the detailed template published 20 years later. Following the structure of this template, we presented lessons associated with each of the five methods that typically appear in scatter search implementations. The lessons summarize our most relevant experiences and those of others in relation to implementing scatter search. During the preparation of this manuscript, we came to the realization that the most important lesson may be that there is still much to be learned about developing effective scatter search implementations.

References

- Campos, V., F. Glover, M. Laguna and R. Martí (2001) "An Experimental Evaluation of a Scatter Search for the Linear Ordering Problem," *Journal of Global Optimization*, 21:397-414.
- Corberán, A., E. Fernández, M. Laguna and R. Martí (2002) "Heuristic Solutions to the Problem of Routing School Buses with Multiple Objectives," *Journal of the Operational Research Society*, 53(4):427-435.
- Glover, F. (1977) "Heuristics for Integer Programming Using Surrogate Constraints," *Decision Sciences*, 8:156-166.
- Glover, F. (1994) "Tabu Search for Nonlinear and Parametric Optimization (with Links to Genetic Algorithms)," *Discrete Applied Mathematics*, 49:231-255.
- Glover, F. (1997) "A Template for Scatter Search and Path Relinking," in *Lecture Notes in Computer Science*, 1363, J.K. Hao, E. Lutton, E. Ronald, M. Schoenauer, D. Snyers (Eds.), 13-54.
- Glover, F., M. Laguna and R. Martí (2000) "Fundamentals of Scatter Search and Path Relinking," *Control and Cybernetics*, 39(3):653-684.
- Laguna, M., P. Lino, A. Pérez, S. Quintanilla and V. Valls (2000) "Minimizing Weighted Tardiness of Jobs with Stochastic Interruptions in Parallel Machines," *European Journal of Operational Research*, 127(2): 444-457.
- Laguna, M. and R. Martí (2000) "Experimental Testing of Advanced Scatter Search Designs for Global Optimization of Multimodal Functions," University of Colorado at Boulder.
- Laguna, M. and R. Martí (2002) "Neural Network Prediction in a System for Optimizing Simulations," *IIE Transactions*, 34(3):273-282.
- Laguna, M. and R. Martí (2003) *Scatter Search: Methodology and Implementations in C*, Kluwer Academic Publishers, Boston.

- Mazzini, R. (1998) "Estudo de Meta-Heurísticas Populacionais para a Programação de Máquinas Paralelas com Tempos de Preparação Dependentes da Sequência e Datas de Entrega," Doctoral Thesis, Faculdade de Engenharia Elétrica e de Computação, Universidade Estadual de Campinas.
- Ugray, Z., L.S. Lasdon and J.C. Plummer (2000) "OQGRG: A Scatter Search Approach for Solving Constrained Nonlinear Global Optimization Problems," presented at INFORMS San Antonio, November 5-8.
- Valls, V., M. Laguna, P. Lino, A. Pérez and S. Quintanilla (1998) "Project Scheduling with Stochastic Activity Interruptions," in *Project Scheduling: Recent Models, Algorithms and Applications*, Jan Weglarz (Ed.), Kluwer Academic Publishers, 333-353.

Chapter 11

TABU SEARCH FOR MIXED INTEGER PROGRAMMING

João Pedro Pedrosa

Departamento de Ciéncia de Computadores, Faculdade de Ciéncias da Universidade do Porto,
Rua do Campo Alegre, 823 4150-180 Porto, Portugal. jpp@ncc.up.pt

Abstract This paper introduces tabu search for the solution of general linear integer problems. Search is done on integer variables; if there are continuous variables, their corresponding value is determined through the solution of a linear program, which is also used to evaluate the integer solution. The complete tabu search procedure includes an intensification and diversification procedure, whose effects are analysed on a set of benchmark problems.

Keywords: Tabu search, Linear Integer Programming, Mixed Integer Programming

1. Introduction

In this work we focus on a tabu search for the problem of optimizing a linear function subject to a set of linear constraints, in the presence of integer and, possibly, continuous variables. If the subset of continuous variables is empty, the problem is called *pure integer* (IP). In the more general case, where there are also continuous variables, the problem is usually called *mixed integer* (MIP).

The mathematical programming formulation of a mixed integer linear program is

$$z = \min_{x,y} \{cx + hy : Ax + Gy \geq b, x \in \mathbb{Z}_+^n, y \in \mathbb{R}_+^p\} \quad (11.1)$$

where \mathbb{Z}_+^n is the set of nonnegative, integral n -dimensional vectors and \mathbb{R}_+^p is the set of nonnegative, real p -dimensional vectors. A and G are $m \times n$ and $m \times p$ matrices, respectively, where m is the number of constraints. The integer variables are x , and the continuous variables are y . We assume that there are additional bound restrictions on the integer variables: $l_i \leq x_i \leq u_i$, for $i = 1, \dots, n$.

The strategy proposed consists of fixing the integer variables x by tabu search, and obtaining the corresponding objective z and continuous variables y by solving a linear programming (LP) problem (this idea has been introduced in Pedroso (1998), Pedroso (2002), Neto and Pedroso (2003)). We are therefore using tabu search to fix only integer variables, as the continuous ones can be unambiguously determined in function of them.

In the process of evaluation of a solution, we first formulate an LP by fixing all the variables of the MIP at the values determined by tabu search:

$$z = \min_y \{cx + hy : Ax + Gy \geq b, x = \bar{x}, y \in \mathbb{R}_+^p\} \quad (11.2)$$

We are now able to solve this (purely continuous) linear problem using the simplex algorithm. If it is feasible, the evaluation given to \bar{x} is the objective value z ; as the solution is feasible, we set the sum of violations, ζ , to zero. If this problem is infeasible, we set ζ equal the total constraint violations (obtained at the end of phase I of the simplex algorithm). Notice that for IPs, after fixing the integer variables the remaining problem has no free variables; some LP solvers might not provide the correct values of z or ζ for the fixed variables.

We say that a solution structure i is better than another structure j if $\zeta^i < \zeta^j$, or $\zeta^i = \zeta^j$ and $z^i < z^j$ (for minimization problems).

The initial solution is obtained by rounding the integer variables around their optimal values on LP relaxations. Tabu search starts operating on this solution by making changes exclusively on the integer variables, after which the continuous variables are recomputed through LP solutions.

Modifications of the solution are made using a simple neighborhood structure: incrementing or decrementing one unit to the value of an integer variable of the MIP. This neighborhood for solution x is the set of solutions which differ from x on one element x_i , whose value is one unit above or below x_i . Hence x' is a neighbor solution of x if $x'_i = x_i + 1$, or $x'_i = x_i - 1$, for one index i , and $x'_j = x_j$ for all indices $j \neq i$.

Moves are tabu if they involve a variable which has been changed recently. An aspiration criterion allows tabu moves to be accepted if they lead to the best solution found so far. This is the basic tabu search, based only on short term memory, as described in Glover (1989).

As suggested in Glover (1990), we complement this simple tabu search with intensification and diversification. Intensification allows non-tabu variables to be fixed by branch-and-bound (B&B). Diversification creates new solutions based on LP relaxations, but keeping a part of the current solution unchanged.

We have tested the tabu search with a subset of benchmark problems that are available, in the MPS format, in the Bixby et al. (1998), MIPLIB (1996). Results obtained by tabu search are compared to the solution of B&B. We have used a publicly available implementation of this algorithm, the GLPK software package. This was used also as the LP simplex solver on tabu search, as well as

the B&B solver on the intensification phase, thus allowing a direct comparison to *GLPK* in terms of CPU time required for the solution.

Various libraries provide canned subroutines for tabu search, and some of these can be adapted to integer programming applications. However, such libraries operate primarily as sets of building blocks that are not organized to take particular advantage of considerations relevant to the general MIP setting. A partial exception is the tabu search component of the COIN-OR open source library (see COIN-OR (2004)). However, this component is acknowledged to be rudimentary and does not attempt to encompass many important elements of tabu search.

2. A Simple Tabu Search

In this section we introduce a simple tabu search, based only on short term memory (Algorithm 1). This procedure has one parameter: the number of iterations, N , which is used as the stopping criterion. The other arguments are a seed for initializing the random number generator, and the name of the *MPS* benchmark file.

The part of the MIP solution that is determined by tabu search is the subset of integer variables x in Equation 11.1. The data structure representing a solution is therefore an n -dimensional vector of integers, $\bar{x} = (\bar{x}_1 \dots \bar{x}_n)$.

Algorithm 1: Simple tabu search.

```
SIMPLETABU( $N$ , seed, MPSfile)
(1)   read global data  $A$ ,  $G$ ,  $b$ ,  $c$ , and  $h$  from MPSfile
(2)   initialize random number generator with seed
(3)    $\bar{x} := \text{CONSTRUCT}()$ 
(4)    $\bar{x}^* := \bar{x}$ 
(5)    $t := (-n, \dots, -n)$ 
(6)   for  $i = 1$  to  $N$ 
(7)        $\bar{x} := \text{TABUMOVE}(\bar{x}, \bar{x}^*, i, t)$ 
(8)       if  $\bar{x}$  is better than  $\bar{x}^*$ 
(9)            $\bar{x}^* := \bar{x}$ 
(10)      return  $\bar{x}^*$ 
```

2.1 Construction

Construction is based on the solution of the LP relaxation with a set of variables \mathcal{F} fixed, as stated in equation 11.3 (empty \mathcal{F} leads to the LP relaxation of the initial problem).

$$\min_{x,y} \{cx + hy : Ax + Gy \geq b, x \in \mathbb{R}_+^n, y \in \mathbb{R}_+^p, x_i = \bar{x}_i \forall i \in \mathcal{F}\} \quad (11.3)$$

The solution of this problem is denoted by $x^{LP} = (x_1^{LP}, \dots, x_n^{LP})$; these values are rounded up or down with some probabilities and fixed, as shown in Algorithm 2, were we denote by r a continuous random variable with uniform distribution within $[0, 1]$.

Algorithm 2: Semi-greedy solution construction.

CONSTRUCT()

- (1) $\mathcal{F} := \{\}$
- (2) $\mathcal{C} := \{1, \dots, n\}$
- (3) **for** $j = 1$ **to** n
 - (4) **solve** Equation 11.3
 - (5) **randomly select index** i **from** \mathcal{C}
 - (6) **if** $r < x_i^{LP} - \lfloor x_i^{LP} \rfloor$
 - (7) $\bar{x}_i := \lceil x_i^{LP} \rceil$
 - (8) **else**
 - (9) $\bar{x}_i := \lfloor x_i^{LP} \rfloor$
 - (10) $\mathcal{F} := \mathcal{F} \cup \{i\}$
 - (11) $\mathcal{C} := \mathcal{C} \setminus \{i\}$
 - (12) **return** \bar{x}

This semi-greedy construction is inspired in an algorithm provided in Lengauer (1990). It consists of rounding each variable i to an integer next to its value on the LP relaxation, x_i^{LP} . For all the indices $i \in \{1, \dots, n\}$, the variable \bar{x}_i is equal to the value x_i^{LP} rounded down with probability

$$P(\bar{x}_i = \lfloor x_i^{LP} \rfloor) = \lceil x_i^{LP} \rceil - x_i^{LP},$$

or rounded up with probability $1 - P(\bar{x}_i = \lfloor x_i^{LP} \rfloor)$ (lines 6 to 9 of the Algorithm 2).

2.2 Candidate Selection

At each tabu search iteration, the neighborhood of the current solution is searched and a neighbor solution is selected, as presented in Algorithm 3. The arguments of this algorithm are the current solution \bar{x} , the best solution found \bar{x}^* , the current iteration i , and the tabu vector t . Tabu information is kept in vector t ; t_c holds the iteration at which variable c has been updated.

Lines 1 to 5 prevent the case where the search is blocked, all moves being potentially tabu. In this case a random move is taken: an index is selected randomly, and a value for that variable is drawn within its bounds, with uniform distribution. (We denote by $R[1, n]$ a random integer with uniform distribution within $1, \dots, n$.)

Algorithm 3: Search of a candidate at each tabu search iteration.

```

TABUMOVE( $\bar{x}, \bar{x}^*, k, t$ )
(1)   if  $k - t_i > n \ \forall i$ 
(2)    $c := R[1, n]$ 
(3)    $\bar{x}_c := R[l_c, u_c]$ 
(4)    $t_c := k$ 
(5)   return  $\bar{x}$ 
(6)    $v := \bar{x}$ 
(7)   for  $i = 1$  to  $n$ 
(8)      $s := \bar{x}$ 
(9)      $d := R[1, n]$ 
(10)    foreach  $\delta \in \{-1, +1\}$ 
(11)       $s_i := \bar{x}_i + \delta$ 
(12)      if  $s_i \in [l_i, u_i]$  and  $s$  better than  $v$ 
(13)        if  $k - t_i > d$  or  $s$  better than  $\bar{x}^*$ 
(14)           $v := s$ 
(15)           $c := i$ 
(16)     $\bar{x} := v$ 
(17)     $t_c := k$ 
(18)  return  $\bar{x}$ 
```

The neighborhood is searched by adding a value $\delta = \mp 1$ to each of the variables $1, \dots, n$, as long as they are kept within their bounds. We have tested two search possibilities: breadth first and depth first. With breadth first all the neighbor solutions are checked, and the best is returned (lines 7 to 15). With depth first, as soon as a non-tabu solution better than the current solution is found, it is returned. Results obtained for these two strategies are rather similar, but there seems to be a slight advantage to breadth-first, which is the strategy that adopted in this paper. More sophisticated ways of managing choice rules by means of candidate list strategies are an important topic in tabu search (see, e.g., Glover and Laguna (1997)), and may offer improvements, but we elected to keep this aspect of the method at a simple level.

The tabu tenure (the number of iterations during which a changed variable remains tabu) is generally a parameter of tabu search. In order to simplify the parameterization, we decided to consider it a random value between 1 and the number of integer variables, n . Such value, d , is drawn independently for each variable (line 9); this might additionally lead to different search paths when escaping the same local optimum, in case this situation arises.

2.3 Results

The set of benchmark problems and the statistical measures used to report solutions are presented in appendix 11.A.3.

Results obtained by this simple tabu search, presented in table 11.1 are encouraging, as good solutions were found to problems which could not be easily solved by B&B (please see next section for B&B results). Still, for many problems the optimal solution was not found. This simple tabu search is many times trapped in regions from which it cannot easily escape, wasting large amounts of time. As we will see in the next section, this can be dramatically improved with intensification and diversification procedures.

problem name	best z	best ζ	%above optimum	%feas runs ($E[t^f]$ (s))	Feasibility runs ($E[t^f]$ (s))	%best	Best sol. runs	%opt	Optimality ($E[t^f]$ (s))
bell3a	878430.32	0	0	100	0.23	75	50.32	75	50.32
bell5	9011612.98	0	0.50	30	212.01	5	1688.15	0	$\gg 1764.32$
egout	568.1007	0	0	100	0.47	100	7.56	100	7.56
enigma	0	0.0278	n/a	0	$\gg 1638.69$	5	1559.63	0	$\gg 1638.69$
flugpl	1201500	0	0	30	19.54	15	42.85	15	42.85
gt2	23518	0	11.11	100	0.67	5	4506.53	0	$\gg 4659.14$
lseu	1218	0	8.75	45	95.75	5	1441.24	0	$\gg 1512.53$
mod008	307	0	0	100	0.17	5	10751.22	5	10751.22
modglob	20757757.11	0	0.08	100	0.18	5	26546.63	0	$\gg 27873.20$
noswot	-41	0	4.65	95	13.06	20	928.09	0	$\gg 4667.98$
p0033	3347	0	8.35	20	55.55	15	78.70	0	$\gg 272.96$
pk1	17	0	54.54	100	0.03	5	2009.57	0	$\gg 2046.81$
pp08a	7350	0	0	100	0.11	25	1439.24	25	1439.24
pp08aCUT	7350	0	0	100	0.15	20	3090.42	20	3090.42
rgn	82.1999	0	0	100	0.03	100	6.13	100	6.13
stein27	18	0	0	100	0.01	100	0.05	100	0.05
stein45	30	0	0	100	0.06	80	66.39	80	66.39
vpm1	20	0	0	100	0.52	5	12149.37	5	12149.37

Table 11.1. Simple tabu search: best solution found, percent distance above optimum; expected CPU time required for reaching feasibility, the best solution, and optimality. (Results based on 20 observations of the algorithm running for 5000 iterations.)

3. Intensification and Diversification

We now introduce intensification and diversification procedures to complement the simple tabu search presented in the previous section. These are essential to the performance of the algorithm; in many situations they can save a large amount of computational time, both by speeding up the search in case of being far from a local optima, or by moving the tabu search to different regions when it is trapped somewhere it cannot easily escape.

3.1 Intensification

The complete intensification strategy is presented in Algorithm 4. It consists of fixing all the variables which are tabu (those belonging to set \mathcal{F} , determined in line 1 of Algorithm 4), releasing all the non-tabu variables, and solving the remaining MIP on these:

$$\min_{x,y} \{cx + hy : Ax + Gy \geq b, x \in \mathbb{Z}_+^n, y \in \mathbb{R}_+^p, x_k = \bar{x}_k \forall k \in \mathcal{F}\} \quad (11.4)$$

The rationale behind this procedure is the following: most of the difficult MIP became easy to solve (or at least much easier than the original problem) if a set of important variables are fixed (a fuller development of this type of strategy can be found in Glover (1977), and a related discussion appears in Fischetti and Lodi (2003)). What we are doing here is to say that important variables at a given point of the search are those which are tabu (i.e., those which have been updated recently).

Algorithm 4: Intensification.

INTENSIFY(x, t, i)

- (1) $\mathcal{F} := \{k : i - t_k > n\}$
- (2) **while** Equation 11.3 is not feasible
- (3) randomly select index k from \mathcal{F}
- (4) $\mathcal{F} := \mathcal{F} \setminus \{k\}$
- (5) solve Equation 11.4 (allow search for n seconds, max.)
- (6) **if** no integer solution was found
- (7) **return** x
- (8) let x' be the solution of Equation 11.4
- (9) **return** x'

For some strongly constrained instances, the problem 11.4 might have no feasible LP relaxation. In this case, we randomly remove variables from the set \mathcal{F} , until the LP relaxation becomes feasible (lines 2 to 4). On the other end, the MIP problem of Equation 11.4 might still be very difficult to solve; in order to avoid wasting too much time on its solution, we limit the time spent on it. This limit could be parameterized; but, in order to keep the discussion of results free of parameters, we decided to allow an amount of time equal to the number of integer variables, n , in seconds. (Actually, this is a rather poor choice: in our machine, in most of the cases either a good solution is found in about one second, or it is not found in n seconds; but let us keep it as proposed for the sake of simplicity. Notice also that a way of improving this parameterization would be to use a dynamic strategy, allowing more time to intensification when it appears to be rewarding, as is current practice in tabu search.)

3.2 Diversification

The diversification procedure is similar to construction, but it keeps a part of the current solution structure unchanged. This procedure, presented in Algorithm 5 starts by drawing a random integer l , between 1 and the number of integer variables n . It will then randomly select l variables to remove (lines 4 to 7), and fix them (in a random order), by means of the rounding technique described in section 2.1 (lines 8 to 16).

Algorithm 5: Diversification: partial solution destruction and reconstruction.

```

DIVERSIFY( $x$ )
(1)    $\mathcal{F} := \{1, \dots, n\}$ 
(2)    $\mathcal{C} := \{\}$ 
(3)    $l = R[1, n]$ 
(4)   for  $k = 1$  to  $l$ 
(5)     randomly select index  $k$  from  $\mathcal{F}$ 
(6)      $\mathcal{F} := \mathcal{F} \setminus \{k\}$ 
(7)      $\mathcal{C} := \mathcal{C} \cup \{k\}$ 
(8)   for  $k = 1$  to  $l$ 
(9)     solve Equation 11.3
(10)    randomly select index  $k$  from  $\mathcal{C}$ 
(11)    if  $r < x_k^{LP} - \lfloor x_k^{LP} \rfloor$ 
(12)       $x_k := \lceil x_k^{LP} \rceil$ 
(13)    else
(14)       $x_k := \lfloor x_k^{LP} \rfloor$ 
(15)     $\mathcal{F} := \mathcal{F} \cup \{k\}$ 
(16)     $\mathcal{C} := \mathcal{C} \setminus \{k\}$ 
(17)   return  $\bar{x}$ 
```

With this procedure, on average 50% of the current solution structure will be kept after diversification; additionally, the reconstructed part will still have the high quality provided by the rounding procedure.

3.3 The Complete Tabu Search

Diversification and intensification have to be carefully combined in order to do a good team work inside tabu search. The main procedure, presented in Algorithm 6, starts with a construction (as in the case of simple tabu). This initiates the first “diversification stream”. A simple, short term memory tabu search starts with this solution (lines 15, 16), and pursues until, at a certain point of the search on this stream, there will be an intensification (lines 9, 10). After doing at least one intensification, and having no improvements for a certain

number of tabu search iterations, there will be a diversification (lines 11 to 14); this starts the next diversification stream.

Algorithm 6: A complete tabu search.

TABUSEARCH($N, seed, MPSfile$)

- (1) read global data A, G, b, c , and h from $MPSfile$
- (2) initialize random number generator with $seed$
- (3) $\bar{x} := \text{CONSTRUCT}()$
- (4) $\bar{x}^* := \bar{x}$
- (5) $\bar{x}' := \bar{x}$
- (6) $q = 0$
- (7) $t := (-n, \dots, -n)$
- (8) **for** $i = 1$ **to** N
 - (9) **if** $q = n$
 - (10) $\bar{x} := \text{INTENSIFY}(\bar{x}, t, i)$
 - (11) **else if** $q > n$
 - (12) $\bar{x} := \text{DIVERSIFY}(\bar{x})$
 - (13) $\bar{x}' := \bar{x}$
 - (14) $q = 0$
 - (15) **else**
 - (16) $\bar{x} := \text{TABUMOVE}(\bar{x}, \bar{x}^*, i, t)$
 - (17) **if** \bar{x} is better than \bar{x}^*
 - (18) $\bar{x}^* := \bar{x}$
 - (19) **if** \bar{x} is better than \bar{x}'
 - (20) $\bar{x}' := \bar{x}$
 - (21) $q = 0$
 - (22) **else**
 - (23) $q := q + 1$
- (24) **return** \bar{x}^*

In order to do this search in a “parameter-free” fashion, we propose the following: do an intensification after n (the number of integer variables) iterations with no improvement on the best solution found on the current diversification stream. An intensification starts with the best solution found on the current diversification stream, not with the current solution. After $n + 1$ non-improving tabu search iterations (hence after doing at least one intensification), do a diversification. On Algorithm 6, the number of iterations with no improvement on the current stream’s best solution is computed as variable q , in lines 14 and 17 to 23.

3.4 Results

We present results obtained by the complete tabu search algorithm, in table 11.2. Comparing these with the results of simple tabu the importance of intensification and diversification becomes clear; the solution quality is considerably improved, and the CPU time required to solving the problems is much reduced.

problem name	best z	best ζ	%above optim.	%feas runs	Feasibility $E[t^f](s)$	%best runs	Best sol. $E[t^f](s)$	%opt runs	Optimality $E[t^f](s)$
bell3a	878430.32	0	0	100	0.24	100	4.38	100	4.38
bell5	8966406.49	0	0	100	8.60	100	38.24	100	38.24
egout	568.1007	0	0	100	0.47	100	6.76	100	6.76
enigma	0	0	0	35	187.51	20	376.66	20	376.66
flugpl	1201500	0	0	100	1.52	100	1.55	100	1.55
gt2	21166	0	0	100	0.65	15	2216.95	15	2216.95
lseu	1120	0	0	100	1.47	10	770.45	10	770.45
mod008	307	0	0	100	0.17	40	1119.57	40	1119.57
modglob	20740508.1	0	0	100	0.16	100	1404.29	100	1404.29
noswot	-41	0	4.65	100	8.38	95	239.96	0	$\gg 15653.99$
p0033	3089	0	0	100	0.17	90	4.10	90	4.10
pk1	15	0	36.36	100	0.03	10	1111.96	0	$\gg 2357.09$
pp08a	7350	0	0	100	0.11	45	4316.38	45	4316.38
pp08aCUT	7350	0	0	100	0.15	70	766.59	70	766.59
rgn	82.1999	0	0	100	0.03	100	0.73	100	0.73
stein27	18	0	0	100	0.02	100	0.05	100	0.05
stein45	30	0	0	100	0.06	80	66.65	80	66.65
vpm1	20	0	0	100	0.48	95	393.73	95	393.73

Table 11.2. Complete tabu search: best solution found, percent distance above optimum; expected CPU time required for reaching feasibility, the best solution, and optimality. (Results based on 20 observations of the algorithm running for 5000 iterations.)

The results obtained utilizing the B&B implementation of *GLPK* on the series of benchmark problems selected are provided in the Table 11.3. The maximum CPU time allowed is 24 hours; in case this limit was exceeded, the best solution found within the limit is reported. *GLPK* uses a heuristic by Driebeck and Tomlin to choose a variable for branching, and the best projection heuristic for backtracking (see Makhorin (2004) for further details).

The analysis of tables 11.2 and 11.3 shows that for most of the benchmark instances, tabu search requires substantially less CPU for obtaining the optimal solution than B&B (though the times reported for B&B are for the complete solution of the problem, not only finding the optimal solution). Two of the problems for which B&B could not find an optimal solution in 24 hours of CPU time (gt2 and modglob) were solved by tabu search in a reasonable amount of time.

problem name	best z	CPU time (s)	remarks
bell3a	878430.32	134.7	
bell5	8966406.49	143.3	
egout	568.1007	3.6	
enigma	0	14.1	
flugpl	1201500	1.3	
gt2	30161*	93822.3	stopped, >24h CPU time
lseu	1120	96.6	
mod008	307	51.0	
modglob	20815372.17*	93839.7	stopped, >24h CPU time
noswot	-41*	137.9	stopped, numerical instability
p0033	3089	1.1	
pk1	11	49713.9	
pp08a	7350	93823.4	stopped, >24h CPU time
pp08aCUT	7350	93822.3	stopped, >24h CPU time
rgn	82.12	4.1	
stein27	18	3.9	
stein45	30	269.3	
vpm1	20	10261.8	

Table 11.3. Results obtained by branch-and-bound, using *GLPK - version 4.4*: solution found and CPU time. (*) indicates non-optimal solutions.)

We also present the results obtained utilizing the commercial solver *Xpress-MP Optimizer, Release 13.02*, again limiting the CPU time to 24 hours maximum. Although there are some exceptions, this solver is generally much faster than our implementation of tabu search, but, as the code is not open, we do not know why. A part of the differences could be explained by the quality of the LP solver. Another part could be due to the use of branch-and-cut. Finally, some differences could be due to preprocessing; in our opinion, this is probably the improvement on tabu search that could bring more important rewards.

4. Conclusion

The literature in meta-heuristics reports many tabu search applications to specific problems. In this paper we present a version to solve general integer linear problems. In this domain, the most commonly used algorithm is branch-and-bound. This algorithm converges to the optimal solution, but might be unusable in practical situations due to the large amounts of time or memory required for solving some problems.

The tabu search proposed in this paper provides a way for quickly solving to optimality most of the problems analyzed. In terms of time required for reaching the optimal (or a good) solution, comparison with the branch-and-bound algorithm implemented in the *GLPK* favors tabu search. Comparison with the commercial solver *Xpress-MP Optimizer* in general is not favorable

problem name	best z	CPU time (s)	remarks
bell3a	878430.32	87	
bell5*	8988042.65*	>24h	stopped, >24h CPU time
egout	568.1007	0	
enigma	0	0	
flugpl	1201500	0	
gt2	21166	0	
lseu	1120	0	
mod008	307	0	
modglob	20740508.1	0	
noswot*	-41*	>24h	stopped, >24h CPU time
p0033	3089	0	
pk1	11	937	
pp08a	7350	31	
pp08aCUT	7350	5	
rgn	82.1999	0	
stein27	18	1	
stein45	30	142	
vpm1	20	0	

Table 11.4. Results obtained by the commercial *Xpress-MP Optimizer, Release 13.02*: solution found and CPU time reported by the solver. (* indicates non-optimal solutions.)

to tabu search, although there are some exceptions. Probably, preprocessing is playing an important role in *Xpress-MP*'s performance; actually, we believe that preprocessing is the most promising research direction for improving tabu search performance.

Our tabu search implementation, utilizing the *GLPK* routines, is publicly available Pedroso (2004); the reader is kindly invited to use it. The implementation with *GLPK* is very simple: tabu search routines are just a few hundreds of C programming lines, which can be easily adapted to more specific situations.

The strategy proposed in this work makes it straightforward to apply tabu search to any model that can be specified in mathematical programming, and thus opens a wide range of applications for tabu search within a single framework. Tabu search can be used to provide an initial solution for starting a branch-and-bound process; it can also be used for improving an integer solution found by a branch-and-bound which had to be interrupted due to computational time limitations.

We emphasize our implementation is not the only one possible. Three types of procedures for applying tabu search to MIP problems are discussed in Glover (1990), utilizing strategies somewhat different than those proposed here. To our knowledge, none of these alternatives has been implemented and tested. Additional considerations for applying tabu search to mixed integer programming are discussed in Glover and Laguna (1997), including tabu branching procedures and associated ideas of branching on created variables that provide an

opportunity to generate stronger branches than those traditionally employed. It is hoped that the present work, which appears to mark the first effort to investigate tabu search in the general MIP setting, will spur additional explorations of this topic.

Appendix

1. Benchmark Problems

The instances of MIP and IP problems used as benchmarks are defined in the Bixby et al. (1998) and are presented in Table 11.A.1. They were chosen to provide an assortment of MIP structures, with instances coming from different applications.

Problem name	Application	Number of variables			Number of constraints	Optimal solution
		total	integer	binary		
bell3a	fiber optic net. design	133	71	39	123	878430.32
bell5	fiber optic net. design	104	58	30	91	8966406.49
egout	drainage syst. design	141	55	55	98	568.101
enigma	unknown	100	100	100	21	0
flugpl	airline model	18	11	0	18	1201500
gt2	truck routing	188	188	24	29	21166
lseu	unknown	89	89	89	28	1120
mod008	machine load	319	319	319	6	307
modglob	heating syst. design	422	98	98	291	20740508
noswot	unknown	128	100	75	182	-43
p0033	unknown	33	33	33	16	3089
pk1	unknown	86	55	55	45	11
pp08a	unknown	240	64	64	136	7350
pp08acut	unknown	240	64	64	246	7350
rgn	unknown	180	100	100	24	82.1999
stein27	unknown	27	27	27	118	18
stein45	unknown	45	45	45	331	30
vpm1	unknown	378	168	168	234	20

Table 11.A.1. Set of benchmark problems used: application, number of constraints, number of variables and optimal solutions as reported in MIPLIB.

2. Computational Environment

The computer environment used on this experiment is the following: a Linux Debian operating system running on a machine with an AMD Athlon processor at 1.0 GHz, with 512 Gb of RAM. Both tabu search and *GLPK* were implemented on the C programming language.

3. Statistics Used

In order to assess the empirical efficiency of tabu search, we provide measures of the expectation of the CPU time required for finding a feasible solution, the best solution found, and the optimal solution, for each of the selected MIP problems.

Let t_k^f be the CPU time required for obtaining a feasible solution in iteration k , or the total CPU time in that iteration if no feasible solution was found. Let t_k^o and t_k^b be identical measures for reaching optimality, and the best solution found by tabu search, respectively. The number of independent tabu search runs observed for each benchmark is denoted by K . Then, the expected CPU time required for reaching feasibility, based on these K iterations, is:

$$E[t^f] = \sum_{k=1}^K \frac{t_k^f}{r_f},$$

while

$$E[t^b] = \sum_{k=1}^K \frac{t_k^b}{r_b}$$

is the expected CPU time for finding the best tabu search solution, and the expected CPU time required for reaching optimality is

$$E[t^o] = \sum_{k=1}^K \frac{t_k^o}{r_o}.$$

For $r^f = 0$ and $r^o = 0$, the sums provide respectively a lower bound on the expectations of CPU time required for feasibility and optimality.

References

- Bixby, R. E., S. Ceria, C. M. McZeal and M. Savelsbergh (1998) An Updated Mixed Integer Programming Library, Technical report, Rice University, TR98-03.
- COIN-OR (2004) COmputational INfrastructure for Operations Research, Internet repository, version 1.0, www.coin-or.org.
- Fischetti, M. and A. Lodi (2003) Local Branching, *Mathematical Programming*, 98:23–47.
- Glover, F. and M. Laguna (1997) *Tabu Search*, Kluwer Academic Publishers, Boston.
- Glover, F. (1977) Heuristics for Integer Programming using Surrogate Constraints, *Decision Sciences*, 8(1):156–166.
- Glover, F. (1989) Tabu search—Part I, *ORSA Journal on Computing*, 1:190–206.

- Glover, F. (1990) Tabu search—Part II, *ORSA Journal on Computing*, 2:4–32.
- Lengauer, T. (1990) *Combinatorial Algorithms for Integrated Circuit Layout*, Applicable Theory in Computer Science. John Wiley and Sons, 427–446.
- Makhorin, A. (2004) *GLPK – GNU Linear Programming Kit*, Free Software Foundation, version 4.4.
- MPLIB (1996) Internet repository, v. 3.0, www.caam.rice.edu/~bixby/mplib.
- Neto, T. and J. P. Pedroso (2003) Grasp for Linear Integer Programming, In Sousa, Jorge P. and Resende, Mauricio G. C., editors, *METAHEURISTICS: Computer Decision-Making*, Combinatorial Optimization Book Series, Kluwer Academic Publishers, 545–574.
- Pedroso, J. P. (1998) An Evolutionary Solver for Linear Integer Programming, BSIS Technical Report 98-7, Riken Brain Science Institute, Wako-shi, Saitama, Japan.
- Pedroso, J. P. (2002) An Evolutionary Solver for Pure Integer Linear Programming. *International Transactions in Operational Research*, 9(3):337–352.
- Pedroso, J. P. (2004) Tabu search for MIP: An Implementation in the C Programming Language, Internet repository, version 1.0, www.ncc.up.pt/~jpp/mipts.

Chapter 12

SCATTER SEARCH VS. GENETIC ALGORITHMS

An Experimental Evaluation with Permutation Problems

Rafael Martí¹, Manuel Laguna² and Vicente Campos¹

¹Departamento de Estadística e I.O., Facultad de Matemáticas, Universitat de Valencia,
Dr. Moliner 50, 46100 Burjassot, Valencia, Spain, {rafael.marti, vicente.campos}@uv.es

²Leads School of Business, University of Colorado, Boulder, CO 80309-0419, USA,
laguna@colorado.edu

Abstract: The purpose of this work is to compare the performance of a scatter search (SS) implementation and an implementation of a genetic algorithm (GA) in the context of searching for optimal solutions to permutation problems. Scatter search and genetic algorithms are members of the evolutionary computation family. That is, they are both based on maintaining a population of solutions for the purpose of generating new trial solutions. Our computational experiments with four well-known permutation problems reveal that in general a GA with local search outperforms one without it. Using the same problem instances, we observed that our specific scatter search implementation found solutions of a higher average quality earlier during the search than the GA variants.

Keywords: Scatter Search, Genetic Algorithms, Combinatorial Optimization, Permutation Problems

1. Introduction

Scatter search (SS) and genetic algorithms (GA) were both introduced in the seventies. While Holland (1975) introduced genetic algorithms and the notion of imitating nature and the “survival of the fittest” paradigm, Glover (1977) introduced scatter search as a heuristic for integer programming that expanded on the concept of surrogate constraints. Both methods fall in the category of evolutionary optimization procedures, from the point of view that they build, maintain and evolve a set (population) of solutions throughout the search. Although the population-based approach makes SS and GA part of the so-called evolutionary methods, there are fundamental differences between these two methodologies. One main difference is that genetic algorithms were initially proposed as a mechanism to perform hyperplane sampling rather than optimization. Over the years, however, GAs have morphed into a methodology whose primary concern is the solution of optimization problems (Glover 1994a).

In contrast, scatter search was conceived as an extension of a heuristic in the area of mathematical relaxation, which was designed for the solution of integer programming problems: surrogate constraint relaxation. The following three operations come from the area of mathematical relaxation and they are the core of most evolutionary optimization methods including SS and GAs:

1. Building, maintaining and working with a population of elements (coded as vectors)
2. Creating new elements by combining existing elements.
3. Determining which elements are retained based on a measure of quality

Two of the best-known mathematical relaxation procedures are Lagrangean relaxation (Everett 1963) and surrogate constraint relaxation (Glover 1965). While Lagrangean approaches absorb “difficult” constraints into the objective function by creating linear combinations of them, surrogate constraint relaxation generate new constraints to replace those considered problematic. The generation of surrogate constraints also involves the combination of existing constraints using a vector of weights. In both cases, these relaxation procedures search for the best combination in an iterative manner. In Lagrangean relaxation, for example, the goal is to find the “best” combination, which, for a minimization problem, is the one that results in the smallest underestimation of the true objective function value. Since there is no systematic way of finding such weights (or so-called Lagrangean multipliers) in order to produce the smallest (possibly zero) duality gap, Lagrangean heuristics iteratively change the weights according to the degree of violation of the constraints that have been “brought up” to the objective function.

Scatter search is more intimately related to surrogate relaxation procedures, because not only surrogate relaxation includes the three operations outlined above but also has the goal of generating information from the application of these operations. In the case of surrogate relaxation, the goal is to generate information that cannot be extracted from the “parent constraints”. Scatter search takes on the same approach, by generating information through combination of two or more solutions.

Similarities and differences between SS and GAs have been previously discussed in the literature (Glover 1994b, 1995). Hence, our goal is not to elaborate on those discussions and further analyze the fundamental differences of these two approaches. Instead, our main goal is to provide a direct performance comparison in the context of a class of combinatorial optimization problems. Specifically, we make our comparison employing four classes of problems whose solutions can be represented with a permutation. Our SS and GA implementations are based on a model that treats the objective function evaluation as a black box, making the search procedures context-independent. This means that neither implementation takes advantage of the structural details of the tests problems. We base our comparisons on experimental testing with four well-known problems: linear ordering, traveling salesperson, matrix bandwidth reduction and a job-sequencing problem. The only information that both the SS solver and the GA solver have with respect to these problems is the nature of the objective function evaluation with regard to the “absolute” or “relative” positioning of the elements in the permutation. In other words, we differentiate between two classes of problems:

A-permutation problems for which absolute positioning of the elements is more important (e.g., linear ordering problem)

R-permutation problems for which relative positioning of the elements is more important (e.g., traveling salesperson problem)

We will see later that not all problems can be fully characterized as “absolute” or “relative”, however, this does not render our implementations useless.

2. Scatter Search Implementation

Our scatter search implementation is summarized in Figure 1 and operates as follows. A generator of permutations, which focuses on diversification and not on the quality of the resulting solutions, is used at the beginning of the search to build a set P of $PopSize$ solutions (step 1). The generator, proposed by Glover (1998), uses a systematic approach for creating a diverse set of permutations. As is customary in scatter search, an improvement

method is applied to the solutions in P in order to obtain a set of solutions of reasonable quality and diversity. The improvement method consists of the local search (LS) procedure described in the following section.

The reference set, $RefSet$, is a collection of b solutions that are used to generate new solutions by way of applying a solution combination method. The construction of the initial reference set in step 3 starts with the selection of the best $b/2$ solutions from P . These solutions are added to $RefSet$ and

$$d(p, q) = \sum_{i=1}^n |p_i - q_i|$$

deleted from P . The minimum distance from each improved solution in $P-RefSet$ to the solutions in $RefSet$ is computed. Then, the solution with the maximum of these minimum distances is selected. This solution is added to $RefSet$ and deleted from P and the minimum distances are updated. This process is repeated $b/2$ times. The resulting reference set has $b/2$ high-quality solutions and $b/2$ diverse solutions. The distance between two permutations $p = (p_1, p_2, \dots, p_n)$ and $q = (q_1, q_2, \dots, q_n)$ depends on the type of problem being solved. For A-permutation problems, the distance is given by:

The distance for R-permutation problems is defined as:

$d(p, q) = \text{number of times } p_{i+1} \text{ does not immediately follow } p_i \text{ in } q,$
for $i = 1, \dots, n-1$

The combination procedure is applied in step 5 to all pairs of solutions in the current $RefSet$. Since the reference set consists of b solutions, the number of trial solutions generated with the combination method is $b(b-1)/2$ when applied to the initial reference set. Note that only pairs with at least one new solution are combined in subsequent executions of this step and therefore the number of combinations varies after the initial reference set. The combined solutions are improved in the same way as mentioned above, that is, with the application of the LS procedure. The reference set is then updated by selecting the best b solutions from the union of $RefSet$ and the improved trial solutions. Steps 5, 6 and 7 in the outline of Figure 12.1 are performed as long as at least one new trial solution is admitted in the reference set.

1. **Generate solutions** — Apply the diversification generation method to generate a set of $PopSize$ solutions.
2. **Improve solutions** — Apply the LS method to improve solutions generated in Step 1.
3. **Build the reference set** — Choose the “best” b solutions to build the initial $RefSet$.
4. **Initialize** — Make $BestSol$ the best solution in the current $RefSet$

do {

- while** (new solutions in $RefSet$ and objective function evaluations < $MaxEval$) **do {**
- 5. **Combine solutions** — Generate trial solutions from pairs of reference solutions where at least one solution in the pair is new.
- 6. **Improve solutions** — Apply the local search method to improve the solutions generated in step 5.
- 7. **Update reference set** — Choose the best b solutions from the union of the current $RefSet$ and the set of improved trial solutions.
- }**
- 8. **Update the best** — Set $CurrentBest$ as the best solution in the $RefSet$.
- if** ($CurrentBest$ improves $BestSol$)
 $BestSol = CurrentBest$
- 9. **Rebuild RefSet** — Remove the worst $b/2$ solutions from the $RefSet$. Generate $PopSize$ improved solutions applying steps 1 and 2. Choose $b/2$ “diverse” solutions and add them to $RefSet$.

} while (objective function evaluations < $MaxEval$)

Figure 12.1 . Scatter Search outline

When no new solutions qualify to be added to the $RefSet$, step 9 performs a partial rebuilding of the reference set. We keep the best $b/2$ solutions in the $RefSet$ and delete the other $b/2$. As in step 1, a set P of $PopSize$ improved solutions is generated and the $b/2$ with maximum diversity are added to complete the $RefSet$. The procedure stops when $MaxEval$ objective function evaluations have been performed.

Since the SS and GA implementations will share the combination and improvement methods, we first provide the details for the GA procedure and then describe the mechanisms to combine solutions and the local search used for improving trial solutions.

3. GA Implementation

Just as in the case of the scatter search, we have implemented a standard genetic algorithm for the purpose of comparing performance. Our description of scatter search in the previous section is more detailed, because SS is not as known in the literature as GAs, even though they both date back to the mid

seventies. The GA implementation follows the scheme of Michalewicz (1996) and is summarized in Figure 12.2.

-
1. **Generate solutions** — Build P by randomly generating $PopSize$ permutations.
 2. **Improve solutions** — Apply the LS method to improve solutions generated in Step 1.
 - while** (objective function evaluations < $MaxEval$) **do**
 - {
 3. **Evaluate solutions** — Evaluate the solutions in P and update the best solution found if necessary.
 4. **Survival of the fittest** — Calculate the probability of surviving based on solution quality. Evolve P by choosing $PopSize$ solutions according to their probability of surviving.
 5. **Combine solutions** — Select a fraction p_c of the solutions in P to be combined. Selection is at random with the same probability for each element of P . The selected elements are randomly paired for combination, with each pair generating two offspring that replace their parents in P .
 6. **Mutate solutions** — A fraction p_m of the solutions in P is selected for mutation. The mutated solution replaces the original in P .
-

Figure 12.2. GA outline

Offspring generated with the combination procedure in step 5 and mutations generated in step 6 are subjected to the improvement method referred to in step 2. The improvement method is the same as the one used in the scatter search implementation and described in the following section.

4. Improvement Method

$$p' = \begin{cases} (p_1, \dots, p_{i-1}, p_j, p_i, \dots, p_{j-1}, p_{j+1}, \dots, p_n) & \text{for } i < j \\ (p_1, \dots, p_{j-1}, p_{j+1}, \dots, p_i, p_j, p_{i+1}, \dots, p_n) & \text{for } i > j \end{cases}$$

Insertions are used as the primary mechanism to move from one solution to another in our improvement method. We define $MOVE(p_j, i)$ to consist of deleting p_j from its current position j in p to be inserted in position i (i.e., between the current elements p_{i-1} and p_i if $i < j$ and between the current elements p_i and p_{i+1} if $i > j$). This operation results in the ordering p' as follows:

Since the local search method is context independent, the only available mechanism for computing the move value is submitting p' for evaluation and comparing its value with the value of p . In order to reduce the computational

effort associated with evaluating moves for possible selection and to increase the search efficiency, we define $INSERT(p_j)$ as the set of promising insert positions for p_j . We consider inserting p_j only in those positions in $INSERT(p_j)$. Then, the neighborhood N of the current solution is given as:

$$N = \{p' : MOVE(p_j, i), \text{ for } j = 1, \dots, n \text{ and } i \in INSERT(p_j)\}$$

We partition N into n sub-neighborhoods N_j associated with each element p_j as:

$$N_j = \{p' : MOVE(p_j, i), i \in INSERT(p_j)\}$$

The set $INSERT(p_j)$ depends on whether the problem is an A-permutation or a R-permutation problem. In A-permutation problems we accumulate in $FreqIns(i,j)$ the number of times that element i has been inserted in position j improving the current solution. Then, given an element i , we compute $m(i)$ as the position j where the value of $FreqIns(i,j)$ is maximum. We consider that $m(i)$ and the positions around it are desirable positions for inserting element i . This information is used to assign $INSERT(p_j)$ the following values:

$$INSERT(p_j) = [m(p_j) - RANGE, m(p_j) + RANGE]$$

The value of $RANGE$ is an additional search parameter. In R-permutation problems we accumulate in $FreqIns(i,j)$ the number of times that element i has been inserted in the position immediately preceding element j . Then we compute $m(i)$ as the element j with maximum $FreqIns(i,j)$ value. We define $pp(e)$ as the previous position of element e in the current solution. Then we can consider that $pp(m(i))$ is a desirable position for inserting element i . In this case, $INSERT(p_j)$ is assigned the following values:

$$INSERT(p_j) = \{ pp(e) / FreqIns(p_j, e) \geq \alpha m(p_j) \}$$

where the value of α is dynamically adjusted to obtain a set with $2*RANGE$ elements. The implementation is such that we avoid ordering all the elements in $FreqIns(i,j)$, so we are able to construct the set $INSERT(p_j)$ with a low computational effort.

The rule for selecting an element for insertion is based on frequency information. Specifically, the number of times that element j has been moved resulting on an improved solution is accumulated in $freq(j)$. The probability of selecting element j is proportional to its frequency value $freq(j)$.

Starting from a trial solution constructed with either the diversification generator or any of the combination methods, the **local search (LS)**

procedure chooses the best insertion associated with a given element. At each iteration, an element p_j in the current solution p is probabilistically selected according its $\text{freq}(j)$ value. The solution p' with the lowest value in N_j is selected. The LS procedure executes only improving moves. An improving move is one for which the objective function value of p' is better (strictly smaller for minimization problems or strictly larger for maximization problems) than the objective function value of p . The LS procedure terminates when $NTrials$ elements are consecutively selected and the exploration of their neighborhood fails to find an improving move.

5. Combination Methods

The combination methods, as referred to in scatter search, or operators, as referred to in genetic algorithms, are key components in the implementation of these optimization procedures. Combination methods are typically adapted to the problem context. For example, linear combinations of solution vectors have been shown to yield improved outcomes in the context of nonlinear optimization (Laguna and Martí 2000). An adaptive structured combination that focuses on absolute position of the elements in solutions to the linear ordering problem was shown effective in Campos et al. (2001). (This combination method is labeled #7 below.) In order to design a context-independent combination methodology that performs well across a wide collection of different permutation problems, we propose a set of 10 combination methods from which one is probabilistically selected according to its performance in previous iterations during the search.

In our implementation of scatter search, solutions in the *RefSet* are ordered according to their objective function value. So, the best solution is the first one in *RefSet* and the worst is the last one. When a solution obtained with the combination method i (referred to as cm_i) qualifies to be the j^{th} member of the current *RefSet*, we add $b-j+1$ to $\text{score}(\text{cm}_i)$. Therefore, combination methods that generate good solutions accumulate higher scores and increase their probability of being selected. To avoid initial biases, this mechanism is activated after the first *InitIter* combinations, and before this point the selection of the combination method is made completely at random.

In the GA implementation, when a solution obtained with combination method cm_i is better than its parent solutions, $\text{score}(\text{cm}_i)$ is increased by one. If the combination method is a mutation operator, then the score is increased by one when the mutated solution is better than the original solution. Preliminary experiments showed that there was no significant difference between using this scheme from the beginning of the search and waiting *InitIter* combinations to activate it. Therefore, the probabilistic selection

procedure is activated from the beginning of the GA search. A description of the ten combination methods follows, where we refer to the solutions being combined as “reference solutions” and to the resulting solution as the “trial solution” (although in the GA literature reference solutions are known as “parents” and trial solutions as “offspring”).

Combination Method 1

This is an implementation of a classical GA crossover operator. The method randomly selects a position k to be the crossing point from the range $[1, n/2]$. The first k elements are copied from one reference solution while the remaining elements are randomly selected from both reference solutions. For each position i ($i = k+1, \dots, n$) the method randomly selects one reference solution and copies the first element that is still not included in the new trial solution.

Combination Method 2

This method is a special case of 1, where the crossing point k is always fixed to one.

Combination Method 3

This is an implementation of what is known in the GA literature as the partially matched crossover. The method randomly chooses two crossover points in one reference solution and copies the partial permutation between them into the new trial solution. The remaining elements are copied from the other reference solution preserving their relative ordering.

Combination Method 4

This method is a case of what is a mutation operator referred to in the GA literature as partial inversion. The method selects two random points in a chosen reference solution and inverts the partial permutation between them. The inverted partial permutation is copied into the new trial solution. The remaining elements are directly copied from the reference solution preserving their relative order.

Combination Method 5

This combination method also operates on a single reference solution and is known as scramble sublist. The method scrambles a sublist of elements randomly selected in the reference solution. The remaining elements are directly copied from the reference solution into the new trial solution.

Combination Method 6

This is a special case of combination method 5 where the sublist always starts in position 1 and the length is randomly selected in the range $[2, n/2]$. This method is known as scramble initial sublist.

Combination Method 7

The method scans (from left to right) both reference solutions, and uses the rule that each reference solution votes for its first element that is still not included in the new trial solution (referred to as the “incipient element”). The voting determines the next element to enter the first still unassigned position of the trial solution. This is a min-max rule in the sense that if any element of the reference solution is chosen other than the incipient element, then it would increase the deviation between the reference and the trial solutions. Similarly, if the incipient element were placed later in the trial solution than its next available position, this deviation would also increase. So the rule attempts to minimize the maximum deviation of the trial solution from the reference solution under consideration, subject to the fact that other reference solution is also competing to contribute. A bias factor that gives more weight to the vote of the reference solution with higher quality is also implemented for tie breaking. This rule is used when more than one element receives the same votes. Then the element with highest weighted vote is selected, where the weight of a vote is directly proportional to the objective function value of the corresponding reference solution.

Combination Method 8

In this method the two reference solutions vote for their incipient element to be included in the first still unassigned position of the trial solution. If both solutions vote for the same element, the element is assigned. If the reference solutions vote for different elements but these elements occupy the same position in both reference permutations, then the element from the permutation with the better objective function is chosen. Finally, if the elements are different and occupy different positions, then the one in the lower position is selected.

Combination Method 9

Given two reference solutions p and q , this method probabilistically selects the first element from one of these solutions. The selection is biased by the objective function value corresponding to p and q . Let e be the last element added to the new trial solution. Then, p votes for the first unassigned element that is position after e in the permutation p . Similarly, q votes for the first unassigned element that is position after e in q . If both reference solutions vote for the same element, the element is assigned to the next

position in the new trial solution. If the elements are different then the selection is probabilistically biased by the objective function values of p and q .

Combination Method 10

This is a deterministic version of combination method 9. The first element is chosen from the reference solution with the better objective function value. Then reference solutions vote for the first unassigned successor of the last element assigned to the new trial solution. If both solutions vote for the same element, then the element is assigned to the new trial solution. Other wise, the “winner” element is determined with a score, which is updated separately for each reference solution in the combination. The score values attempt to keep the proportion of times that a reference solution “wins” close to its relative importance, where the importance is measured by the value of the objective function. The scores are calculated to minimize the deviation between the “winning rate” and the “relative importance”. For example, if two reference solutions p and q have objective function values of $\text{value}(p) = 40$ and $\text{value}(q) = 60$, then p should contribute with 40% of the elements in the new trial solution and q with the remaining 60% in a maximization problem. The scores are updated so after all the assignments are made; the relative contribution from each reference solution approximates the target proportion. More details about this combination method can be found in Glover (1994b).

6. Test Problems

We have used four combinatorial optimization problems to compare the performance of the scatter search and GA implementations. Solutions to these problems are naturally represented as permutations:

- the bandwidth reduction problem
- the linear ordering problem
- the traveling salesman problem
- a single machine-sequencing problem.

We target these problems because they are well known, they are different among themselves and problem instances with known optimal or high-quality solutions are readily available. Existing methods to solve these problems range from construction heuristics and metaheuristics to exact procedures. We now provide a brief description of each problem class.

The bandwidth reduction problem (BRP) refers to finding a configuration of a matrix that minimizes its bandwidth. The bandwidth of a matrix $A = \{a_{ij}\}$ is defined as the maximum absolute difference between i and j for

which $a_{ij} \neq 0$. The BRP consists of finding a permutation of the rows and columns that keeps the nonzero elements in a band that is as close as possible to the main diagonal of the matrix; the objective is to minimize the bandwidth. This NP-hard problem can also be formulated as a labeling of vertices on a graph, where edges are the nonzero elements of the corresponding symmetrical matrix. Metaheuristics proposed for this problem include a simulating annealing implementation by Dueck and Jeffs (1995) and a tabu search approach by Martí et al. (2001).

The Linear Ordering Problem (LOP) is also a NP-hard problem that has a significant number of applications. This problem is equivalent to the so-called triangulation problem for input-output tables in economics and has generated a considerable amount of research interest over the years, as documented in Grötschel, Jünger and Reinelt (1984), Chanas and Kobylanski (1996), Laguna, Martí and Campos (1999) and Campos et al. (2001). Given a matrix of weights the LOP consists of finding a permutation of the columns (and simultaneously the rows) in order to maximize the sum of the weights in the upper triangle, the equivalent problem in graphs is that of finding, in a complete weighted graph, an acyclic tournament with a maximal sum of arc weights. For a complete description of this problem, its properties and applications see Reinelt (1985).

The Traveling Salesman Problem (TSP) consists of finding a tour (cyclic permutation) visiting a set of cities that minimizes the total travel distance. A incredibly large amount of research has been devoted to this problem, which would be impossible and impractical to summarize here. However, a couple of valuable references about the TSP are Lawler et al. (1985) and Reinelt (1994).

Finally, the fourth problem is a single machine-sequencing problem (SMSP) with delay penalties and setup costs. At time zero, n jobs arrive at a continuously available machine. Each job requires a specified number of time units on the machine and a penalty (job dependent) is charged for each unit that job commencement is delayed after time zero. In addition, there is a setup cost s_{ij} charged for scheduling job j immediately after job i . The objective is to find the schedule that minimizes the sum of the delay and setup costs for all jobs. Note that if delay penalties are ignored, the problem becomes an asymmetric traveling salesman problem. Barnes and Vanston (1981) reported results on three branch and bound algorithms of instances with up 20 jobs and Laguna, Barnes and Glover (1993) developed a TS method that is tested in a set of instances whose size ranges between 20 and 35 jobs.

7. Computational Experiments

For our computational testing, we have employed the following problem instances:

- 37 BRP instances from the Harwell-Boeing Sparse Matrix Collection found in <http://math.nist.gov/MatrixMarket/data/Harwell-Boeing>. This collection consists of a set of standard test matrices arising from problems in linear systems, least squares, and eigenvalue calculations from a wide variety of scientific and engineering disciplines. The size of these instances ranges between 54 and 685 rows with an average of 242.9 rows (and columns).
- 49 LOP instances from the public-domain library LOLIB (1997) found in <http://www.iwr.uni-heidelberg.de/groups/comopt/software/LOLIB>. These instances consist of input-output tables from economic sectors in the European community and their size ranges between 44 and 60 rows with an average of 48.5 rows (and columns).
- 31 TSP instances from the public-domain library TSPLIB (1995) found in <http://elib.zib.de/pub/Packages/mp-testdata/tsp/tsplib/tsp.lib>. These instances range in size between 51 and 575 cities with an average of 159.6 cities.
- 40 SMSPI instances from Laguna, Barnes and Glover (1993) available upon request from laguna@colorado.edu. The best solutions available for these instances are not proven optimal, however they are the best upper bounds ever found. These instances range in size between 20 and 35 jobs with an average of 26.0 jobs.

In order to apply the different strategies described above, we have classified the BRP and SMSPI as A-permutation problems and the LOP and TSP as R-permutation problems. Note that the objective function in the SMSPI is influenced by both the absolute position of the jobs (due to the delay penalties) and the relative position of the jobs (due to the setup costs). Our classification is based on the knowledge that the delay penalties in the tested instances are relatively larger when compare to the setup cost. In cases when this knowledge is not available, it would be recommended to run the procedure twice on a sample set of problems in order to establish the relative importance of the positioning of elements in the permutation.

The solution procedures were implemented in C++ and compiled with Microsoft Visual C++ 6.0, optimized for maximum speed. All experiments were performed on a Pentium III at 800 MHz. The scatter search parameters *PopSize*, *MaxIter*, *b* and *InitIter* were set to 100, 2, 10 and 50 respectively, as recommended in Campos et al. (1999). The parameters associated with the local search procedure were set after some preliminary experimentation (*RANGE* = 3 and *Ntrials* = 25). The GA parameters were set to *PopSize* =

100 , $p_c = 0.25$ and $p_m = 0.01$, as recommended in Michalewicz (1996). Both procedures used the stopping criterion of 1 million objective function evaluations.

In our experiments we compare the SS implementation with two versions of the GA procedure: 1) without local search (GA) and 2) with local search (GALS). SS always uses local search by design. The two GA versions complement each other in that one uses part of its objective function evaluation “budget” to perform local searches and the other uses its entire budget to apply the combination operators and evolve the population of solutions. In our first experiment we restrict the use of combination methods in such a way that both GA versions use combination methods 1-6 and SS uses combination methods 7-10. Recall that combination methods 1-3 are crossover operators and combination methods 4-6 are mutation operators. Combination methods 7-10 are more strategic in nature and one is completely deterministic.

Table 12.1 shows the average percent deviation from the best-known solution to each problem. The procedures were executed once with a fixed random seed and the average is over all instances. It should be mentioned that in the case of LOP and TSP the best solutions considered are the optimal solutions as given in the public libraries. In the case of the BRP the best solutions are from Martí et al. (2001) and the best solutions for the SMSP instances are due to Laguna et al. (1993).

Table 12.1. Percent deviation from best

Method	BRP	LOP	SMSP	TSP
GA	59.124%	6.081%	4.895%	95.753%
GALS	55.500%	0.005%	0.832%	143.881%
SS	58.642%	0.000%	0.291%	43.275%

Table 12.1 shows that SS has a better average performance than GA in all problem types but is inferior to GALS when solving BRP instances. The three methods are able to obtain high quality results for the LOP and the SMSP instances. The results for the TSP are less desirable, but it should be pointed out that these problems are on average larger than the LOP and SMSP instances. Table 12.2 presents the average percent improvement of SS over the two GA versions. The largest improvement occurs when solving TSP instances and the negative improvement associated with GALS shows the better average performance of this method when compared to SS.

Table 12.2. Percent improvement of SS over GA and GALS

Method	BRP	LOP	SMSP	TSP
GA	0.3%	6.5%	4.4%	26.8%
GALS	-2.0%	0.0%	0.5%	41.3%

Table 12.2 shows that in general the use of local search within the GA framework results in improved outcomes, with the TSP as a notable exception to this general rule. For the TSP, the performance of the GA implementation deteriorates when coupled with the local search procedure. The SS vs. GALS in the case of BRP instances deserves a closer examination, because it is the only case in which either GA implementation outperforms SS. Figure 12.3 shows the trajectory of the average percent deviation from the best-known solution as the SS and GALS searches progress.

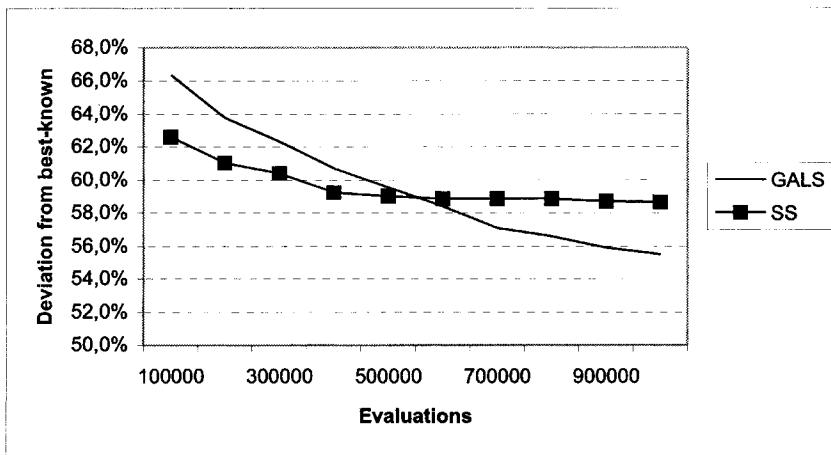


Figure 12.3. Percent deviation from best known solutions on BRP

The percent deviation value trajectory in Figure 12.3 shows that while SS results are better than or at least as good as GALS results before 600 thousand evaluations, GALS keeps improving while SS stagnates. Our explanation for this behavior is as follows. In the BRP, the change in the objective function value from one solution to another does not represent a meaningful guidance for the search. In particular, the objective is a min-max function that in many cases results in the same evaluation for all the solutions in the neighborhood of a given solution. Since SS relies on strategic choices, it is unable to obtain information from the evaluation of this “flat landscape” function to direct the search. GALS, heavily relying on randomization, is able to more effectively probe the solution space presented by BRP instances.

Figure 12.4 depicts the trajectory of the percent deviation from the best-known solution to the SMSP as the search progress. The average values of the three methods under consideration are shown.

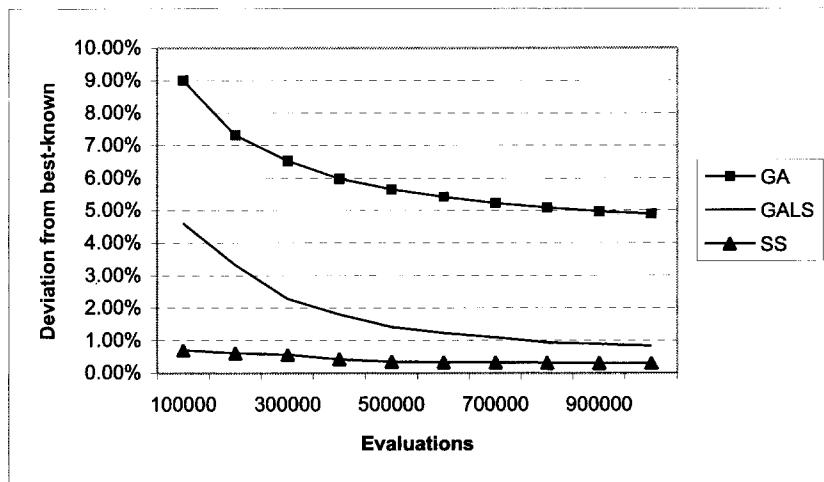


Figure 12.4. Percent deviation trajectory on SMSP

Figure 12.4 shows that the SS procedure is able to obtain high quality solutions from the very beginning of the search. The use of local search coupled with GA gives this procedure a significant advantage. At the end of the search, the percent deviation associated with GA is not as low as the deviation achieved by GALS before 100,000 evaluations. Also note that after 100,000 objective function evaluations, the percent deviation from the best-known solutions for the SS method is 0.7 while after 1 million evaluations the GA and the GALS methods are still at 4.8 and 0.8, respectively.

In our second experiment, we compare SS, GA and GALS when allowed to use all the combination methods described in section 3. The results of these experiments are summarized in Tables 12.3 and 12.4. These tables are equivalent to Tables 12.1 and 12.2 in that they show the percent deviation from the best known solution values and the percent improvement of SS over GA and GALS, respectively.

Table 12.3. Percent deviation from best

Method	BRP	LOP	SMSP	TSP
GA	58.244%	6.722%	4.940%	101.689%
GALS	55.102%	0.004%	0.268%	133.792%
SS	52.587%	0.000%	0.207%	54.321%

Table 12.4. Percent improvement of SS over GA and GALS

Method	BRP	LOP	SMSP	TSP
GA	3.57%	7.21%	4.51%	23.49%
GALS	1.62%	0.00%	0.06%	33.99%

A direct comparison of tables 12.1 and 12.3 shows the advantage of using all combination methods within both GA and SS. Tables 12.3 and 12.4 indicate that when all combination methods are used, SS has a superior average performance than GA and GALS. The performance is only marginally better in the case of LOP and SMSP, but it continues to be significantly better in the case of TSP. When using all operators, including those that incorporate a fair amount of randomization, SS is able to outperform GALS in the BRP instances, as shown in Figure 12.5.

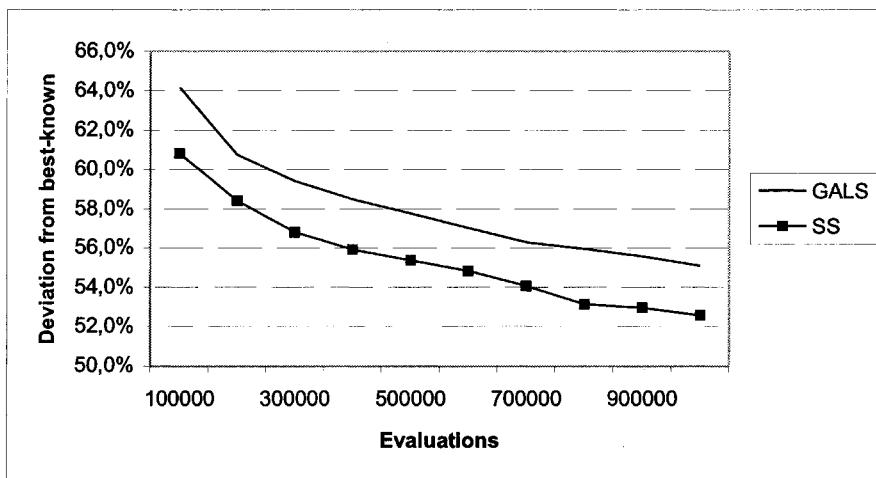


Figure 12.5. Percent deviation trajectory for SS and GALS on BRP

Figure 12.6 shows the trajectory of the percent deviation from the best-known solutions to the SMSP as the search progress. The average values shown in Figure 12.6 correspond to the second experiment, where all combination methods are made available to SS, GA and GALS.

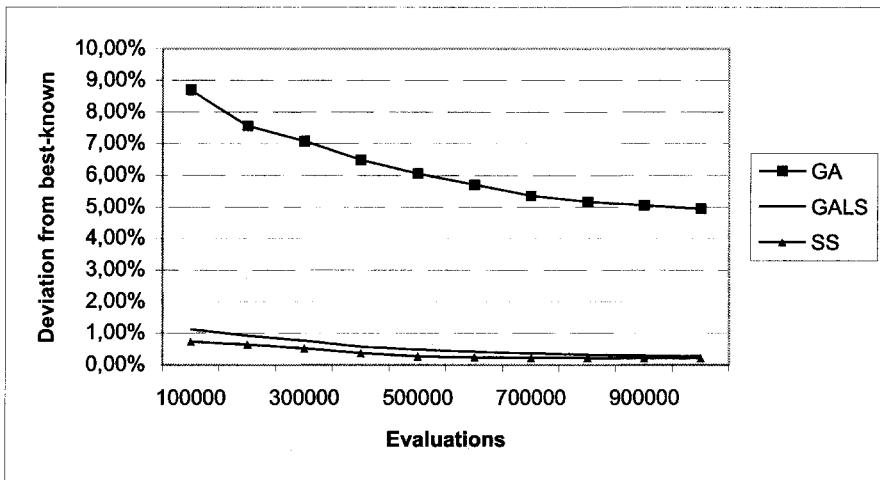


Figure 12.6. Percent deviation trajectory on SMSP

A comparison of figures 12.4 and 12.6 reveals that while SS shows a minor improvement when solving SMSP with using all combination methods, the improvement associated with GALS is significant. GALS achieves a deviation of 1.14% at the 100,000 evaluation mark when using the 10 combination methods, which compares quite favorably with a deviation of 4.8% at the same stage of the search when using a fraction of the available combination methods.

8. Conclusions

We have implemented a scatter search and a GA procedure for a class of combinatorial problems whose solutions can be represented as permutations with the purpose of comparing performance of these competing metaheuristics. The implementations are standard and share 10 combination methods. They also share the improvement method based on a simple hill-climbing procedure. The procedures are context-independent in the sense that they treat the objective function evaluation as a black box. To allow for the use of key search strategies, both implementations require that the problems being solved be classified as either “absolute” or “relative” in terms of the relevant factor for positioning elements in the permutation.

The performance of the procedures was assessed using 157 instances of four different permutations problems. SS can claim superiority when solving

TSP instances but only a modest improvement over GAs with local search when solving LOP, BRP or SMSP instances. We are certain that this won't be the last time SS will be compared against GAs and that in any given setting one could outperform the other. Our main finding is that both methodologies are capable of balancing search diversification and intensification when given the same tools for combining and improving solutions.

Acknowledgements

Research partially supported by the Ministerio de Ciencia y Tecnología of Spain: TIC2000-1750-C06-01.

References

- Campos, V., M. Laguna and R. Martí (1999) "Scatter Search for the Linear Ordering Problem," Corne, Dorigo and Glover (Eds.) *New Ideas in Optimization*, McGraw-Hill, UK.
- Campos, V., F. Glover, M. Laguna and R. Martí (2001) "An Experimental Evaluation of a Scatter Search for the Linear Ordering Problem," *Journal of Global Optimization*, 21:397–414.
- Chanas, S. and P. Kobyłanski (1996) "A New Heuristic Algorithm Solving the Linear Ordering Problem," *Computational Optimization and Applications*, 6:191–205.
- Dueck, G.H. and J. Jeffs (1995) "A Heuristic Bandwidth Reduction Algorithm," *J. of Combinatorial Math. And Comp.*, 18:97–108.
- Everett, H. (1963) "Generalized Lagrangean Multiplier Method for Solving Problems of Optimal Allocation of Resources," *Operations Research*, 11: 399–417.
- Glover, F. (1965) "A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problem," *Operations Research*, 13:879–919.
- Glover, F. (1977) "Heuristics for Integer Programming Using Surrogate Constraints," *Decision Sciences*, 8(7):156–166.
- Glover, F. (1994a) "Genetic Algorithms and Scatter Search: Unsuspected Potentials," *Statistics and Computing*, 4:131–140.
- Glover, F. (1994b) "Tabu Search for Nonlinear and Parametric Optimization with Links to Genetic Algorithms," *Discrete Applied Mathematics*, 49: 231–255.
- Glover, F. (1995) "Scatter Search and Star-Paths: Beyond the Genetic Metaphor," *OR Spektrum*, 17(2–3):125–138.
- Glover, F. (1998) "A Template for Scatter Search and Path Relinking," in *Artificial Evolution, Lecture Notes in Computer Science* 1363, J.K. Hao, E. Lutton, E. Ronald , M. Schoenauer and D. Snyers (Eds.), Springer, 13–54.

- Grötschel, M., M. Jünger and G. Reinelt (1984), "A Cutting Plane Algorithm for the Linear Ordering Problem," *Operations Research*, 32(6):1195–1220.
- Holland, J.H. (1975) *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI.
- Laguna, M., J.W. Barnes and F. Glover (1993), "Intelligent Scheduling with Tabu Search: An Application to Jobs With Linear Delay Penalties and Sequence-Dependent Setup Costs and Times," *Journal of Applied Intelligence*, 3:159–172.
- Laguna, M. and R. Martí (2000) "Experimental Testing of Advanced Scatter Search Designs for Global Optimization of Multimodal Functions," Technical Report TR11-2000, Dpto de Estadística e I.O., University of Valencia.
- Laguna, M., R. Martí and V. Campos (1999) "Intensification and Diversification with Elite Tabu Search Solutions for the Linear Ordering Problem," *Computers and Operations Research*, 26:1217–1230.
- Lawler, L., R. Kan and Shmoys (1985) *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, John Wiley and Sons.
- Martí, R., M. Laguna, F. Glover and V. Campos (2001) "Reducing the Bandwidth of a Sparse Matrix with Tabu Search," *European Journal of Operational Research*, 135:450–459.
- Michalewicz, Z. (1996) *Genetic Algorithms + Data Structures = Evolution Programs*, 3rd edition, Springer-Verlag, Berlin.
- Reinelt, G. (1985) "The Linear Ordering Problem: Algorithm and Applications," *Research and Exposition in Mathematics*, 8, H.H. Hofman and R. Wille (eds.), Heldermann Verlag, Berlin.
- Reinelt, G. (1994) "The Traveling Salesman: Computational Solutions for TSP applications," *Lecture Notes in Computer Science*, Springer Verlag, Berlin.

Chapter 13

PARALLEL COMPUTATION, CO-OPERATION, TABU SEARCH

Teodor Gabriel Crainic

Département de management et technologie, École des Sciences de la Gestion, Université du Québec à Montréal and Centre de recherche sur les transports, Université de Montréal, theo@crt.umontreal.ca

Abstract We present strategies to design parallel tabu search algorithms and survey developments and results in the area. In the second part of the paper, we focus on multi-search strategies. We discuss general design and implementation principles, point out a number of challenges and pitfalls, and identify trends and promising research directions.

Keywords: Parallel Computation, Parallelization Strategies, Co-operative Search, Tabu Search, Scatter Search, Path Relinking

1. Introduction

Parallel computation is increasingly acknowledged as a natural complement to problem solving strategies, particularly when applied to large-scale, hard formulations and instances. The field of meta-heuristics is no exception to this trend and the last ten to fifteen years have witnessed a continuously stronger stream of important developments, most of which targeted tabu search, genetic algorithms, and simulated annealing methods (e.g., the surveys of Crainic and Toulouse 1998, 2003, Cung *et al.* 2002, Holmqvist, Migdalas, and Pardalos 1997, Pardalos *et al.* 1995, and Verhoeven and Aarts 1995). While a number of these efforts addressed particular problem instances and yielded tailor-made procedures, most proposed strategies of a more general nature, at least within the framework of the underlying sequential search methodology. These strategies are the object of this paper.

Parallel computing methods aim to solve a given problem quicker than the corresponding sequential method. In meta-heuristic terms, this goal may be stated as either “accelerate the search for a comparable solution quality” or

“broaden the search”, that is, obtain better solutions for a comparable computation effort or, at least, for the same wall-clock time. Parallel meta-heuristic search methods should also be more robust than sequential methods with respect to the characteristics of the problem instances they encounter. Parallel implementations are thus expected not only to identify better solutions but also to do it consistently over diverse sets of problem instances without excessive calibration efforts.

The review of the literature reveals that, beyond the idiosyncrasies of each implementation that follow from the particular problem instance addressed, there are only a limited number of general strategies that may be used to build parallel meta-heuristics. A number of these strategies are well-understood by now, others are still a fertile ground for research. Together, they enhance our ability to address hard and large-scale problems. In this paper we focus on tabu search. The objectives are to briefly review parallel strategies and developments for tabu search, to discuss in somewhat more details co-operative methods that are the subject of most contemporary efforts, and to identify promising research directions.

The paper is organized as follows. Section 2 describes parallel strategies and reviews parallel tabu search methods proposed in the literature. Section 3 describes the main multi-search strategies and discusses the challenges associated to the development of successful co-operative methods. It also presents a number of ideas on how parallel strategies could be applied to some advanced tabu search methods, particularly path relinking and scatter search. Section 4 sums up the paper with a number of interesting challenges and research directions for parallel tabu search.

2. Parallel Tabu Search

Meta-heuristics have been defined as master strategies (heuristics) to guide and modify other heuristics to avoid getting trapped in local optima or sequences of visited solutions (*cycling*), produce solutions beyond those normally identified by local search heuristics, and provide reasonable assurance that the search has not overlooked promising regions (Glover 1986, Glover and Laguna 1993).

Tabu search, similarly to most meta-heuristics, is an *improving* iterative procedure that *moves* from a given solution to a solution in its *neighbourhood* that is better in terms of the objective function value (or some other measure based on the solution characteristics). Thus, at each iteration, a *local search* procedure (the “original” heuristic) identifies and evaluates solutions in the neighbourhood of the current solution, selects the best one relative to given criteria, and implements the transformations required to establish the selected solution as the current one. Inferior quality solutions encountered during neighbourhood explorations may be accepted as a strategy to move away from local optima. In

theory, the procedure iterates until no further improvement is possible. In actual implementations, a more restrictive stopping criteria is used: total number of iterations, relative improvement over a number of iterations, etc.

Memory and *memory hierarchy* are major concepts in tabu search, as memory-based strategies are used to guide the procedure into various search phases, possibly exploring different neighbourhoods. Short-term *tabu status* memories record recent visited solutions or their attributes to avoid cycling due to the repetition or inversion of recent actions. The tabu status of a move may be lifted if testing it against an *aspiration criterion* signals the discovery of a high quality solution (typically, the best one encountered so far). Medium to long-term memory structures record various informations and statistics relative to the solutions already encountered (e.g., frequency of certain attributes in the best solutions) to “learn” about the solution space and guide the search. *Intensification* of the search around good solutions and its *diversification* towards regions of the solution space not yet explored are two main ingredients of tabu search. These two types of moves are based on medium and long-term memories, are implemented using specific neighbourhoods, and may involve quite complex solution transformations. More details on the basic and advanced features of tabu search may be found in Glover (1986, 1989, 1990, 1996) and Glover and Laguna (1993, 1997).

Tabu search has proved a fertile ground for innovation and experimentation in the area of parallel meta-heuristics. Most parallel strategies encountered in the literature have been applied to tabu search for a variety of applications and a number of interesting parallelization concepts have been introduced while developing parallel tabu search methods (Crainic and Toulouse 2003).

Crainic, Toulouse, and Gendreau (1997) introduced in 1993 what is still the most comprehensive taxonomy of parallel tabu search methods. The classification has three dimensions (Figure 13.1). The first dimension, *Search Control Cardinality*, explicitly examines how the global search is controlled: either by a single process (as in master-slave implementations) or collegially by several processes that may collaborate or not. The two alternatives are identified as *1-control (1C)* and *p-control (pC)*, respectively. The classes of the second dimension indicate the *Search Differentiation*: do search threads start from the same or different solutions and do they make use of the same or different search strategies? The four cases considered are: *SPSS, Same initial Point, Same search Strategy*; *SPDS, Same initial Point, Different search Strategies*; *MPSS, Multiple initial Points, Same search Strategies*; *MPDS, Multiple initial Points, Different search Strategies* (see also Voß 1993). Finally, the dimension relative to the type of *Search Control and Communications* addresses the issue of how information is exchanged.

In parallel computing, one generally refers to *synchronous* and *asynchronous* communications. In the former case, all processes have to stop and engage in

some form of communication and information exchange at moments (number of iterations, time intervals, specified algorithmic stages, etc.) exogenously determined, either hard-coded or determined by a control (master) process. In the latter case, each process is in charge of its own search, as well as of establishing communications with the other processes, and the global search terminates once each individual search stops. To reflect more adequately the quantity and quality of the information exchanged and shared, as well as the additional knowledge derived from these exchanges (if any), we refine these notions and define four classes of Search Control and Communication strategies, *Rigid (RS)* and *Knowledge Synchronization (KS)* and, symmetrically, *Collegial (C)* and *Knowledge Collegial (KC)*. Crainic, Toulouse, and Gendreau (1997) detail the taxonomy and use it to analyze the parallel methods proposed in the literature up to that time.

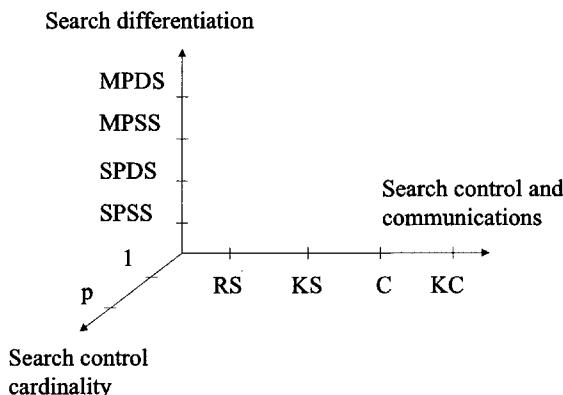


Figure 13.1. Method-oriented Taxonomy

Typically, 1-control strategies implement a classical master-slave approach that aims solely to accelerate the search. Here, a “master” processor executes a synchronous tabu search procedure but dispatches computing-intensive tasks to be executed in parallel by “slave” processes. The master receives and processes the information resulting from the slave operations, selects and implements moves, gathers all the information generated during the tabu exploration, updates the memories, and decides whether to activate different search strategies or stop the search.

In the context of tabu search, the operation most widely targeted in such approaches is the neighbourhood evaluation. At each iteration, the possible moves in the neighbourhood of the current solution are partitioned into as many sets as the number of available processors and the evaluation is carried out in parallel by slave processes. This 1C/RS/SPSS strategy yielded very interesting results for problems with large neighbourhoods and relatively small computing efforts required to evaluate and perform a given move, such as the *quadratic assignment (QAP)* (Chakrapani and Skorin-Kapov 1992, 1993b, 1995, Taillard 1991, 1993b), *travelling salesman (TSP)* (Chakrapani and Skorin-Kapov 1993a) and *vehicle routing (VRP)* (Garcia, Potvin, and Rousseau 1994) problems. For the same quality of solution, near-linear speedups are reported using a relatively small number of processors. Moreover, historically, this approach permitted improvements to the best-known solutions to several problem instances proposed in the literature.

Implementations of the *sequential fan candidate list* strategy (Glover, Taillard, and de Werra 1993, Glover and Laguna 1997), also known as *look ahead* or *probing* approaches, allow slave processes to perform a number of iterations before synchronization and the selection of the best neighbouring solution from which the next iteration is initiated. Probing strategies thus belong to the 1-control, knowledge synchronous (1C/KS) class and may use any of the four search differentiation models identified earlier on. The only parallel implementation of this strategy the author is aware of is to be found in the comparative study of several synchronous tabu search parallelizations performed by Crainic, Toulouse, and Gendreau (1995a) for the location-allocation problem with balancing requirements. In this study, the authors implemented and compared a 1C/RS/SPSS and a 1C/KS/SPSS method. The second performed marginally better. However, both methods were outperformed by p-control implementations that attempt a more thorough exploration of the solution space. These results also emphasize that performance is less interesting for the low-level parallelism offered by single control, synchronous parallel strategies when the time required by one serial iteration is relatively important compared to the total solution time and only a few hundred moves may be executed in a reasonable computing time (compared to the tens of thousands performed by a typical VRP tabu search procedure).

Domain decomposition is another major parallelization strategy. The fundamental idea is simple: Partition the feasible domain of the problem in several (disjoint) sets and execute the contemplated meta-heuristic on each subset, thus accelerating the global search. Two cases may then be encountered. Either the solution of each search thread is a complete solution to the problem in hand or a partial solution only, in which case a complete solution has to be reconstructed. In both cases, however, each search process has had access to a part of the domain only (especially when the partition is strictly enforced during

the search). Then, to somewhat increase the thoroughness of the search, the partition is modified and the search is started anew. This process is repeated a number of times.

The “natural” implementation of domain decomposition is through a 1C/KS master-slave scheme with any of the search differentiation strategy. (Notice that different meta-heuristics may be applied to different sets and that, by definition, each search starts from a different initial point.) The master determines the partition, synchronizes slave processes, reconstructs solutions (when required), and determines stopping conditions. Slave processes perform the search on their assigned partitions. Such approaches have proved successful for problems for which numerous iterations may be performed in a relatively short time and restarting the method with several different partitions does not require unreasonable computational efforts (e.g., Fiechter 1994 for the *TSP*, Porto and Ribeiro 1995, 1996, and Porto, Kitajima, and Ribeiro 2000 for the task scheduling problem on heterogeneous systems).

The same idea may also be implemented in a pC/KS framework, however. In this case, processors stop at pre-determined moments to exchange partial solutions and build whole ones, modify the partition, verify the stopping conditions. From a computer programming point of view, one of the processors may assume co-ordination tasks, but this does not change the particular nature of the algorithmic design. Taillard’s (1993a) early tabu search for VRP follows this pC/KS/MPSS paradigm. The domain is partitioned and vehicles are allocated to the resulting regions. Once the initial partition is performed, each subproblem is solved by an independent tabu search. All processors stop after a number of iterations that varies according to the total number of iterations already performed. The partition is then modified by an information exchange phase, during which tours, undelivered cities, and empty vehicles are exchanged between adjacent processors (corresponding to neighbouring regions).

Partition approaches did allow to address successfully a number of problem instances. The synchronization inherent in the design of the strategy hinder its performance, however. Indeed, in most cases, results obtained by using partition strategies have been improved by methods that launch several search threads to simultaneously explore the solution space with various degrees of co-operation. In fact, enjoying the benefit of hindsight, the main contribution of the Taillard (1993a) paper is to mark the evolution towards one of the most successful sequential meta-heuristics for the VRP, the *adaptive memory* tabu search (Rochat and Taillard 1995, Glover 1996).

According to an adaptive memory approach applied to the VRP, cities are initially separated into several subsets, and routes are built using a construction heuristic. Initial routes are then stored in a structure called an *adaptive memory*. Then, a combination procedure builds a complete solution using the routes in the memory and the solution is further improved using a tabu search method.

The routes of “good” solutions are then deposited into the same memory, which thus adapts to reflect the current state of knowledge of the search. The process then re-starts with a new solution built from the routes stored in the adaptive memory. The method stops when a pre-specified number of calls to the adaptive memory have been performed. This approach clearly implements the principles of decomposition using a serial procedure and has now been successfully applied to other problem classes. However, interestingly, most parallel applications of this approach are now found in co-operative multi-thread strategies.

Multi-thread or *multi-search* parallelizations for tabu search follow the same basic pattern: p threads search through the same solution space, starting from possibly different initial solutions and using possibly different tabu (or other) search strategies. Historically, independent and synchronous co-operative multi-thread methods were proposed first. Currently, asynchronous procedures are being generally developed. One also observes an increased interest in issues related to the definition and modelling of co-operation (Section 3).

Independent multi-searches belong to the pC/RS class of the taxonomy. Most implementations start several independent search processes from different, randomly generated, initial configurations. No attempt is made to take advantage of the multiple threads running in parallel other than to identify the best overall solution once all processes stop. This definitively earns independent search strategies their Rigid Synchronization classification. (Note that, in general, the implementations designate a processor to collect the information and verify stopping criteria.) Battiti and Tecchiolli (1992, for the QAP), Taillard (the main study is found in his 1994 paper on parallel tabu methods for job shop scheduling problems), and others studied and empirically established the efficiency of such procedures when compared to the best heuristics proposed at the time for their respective problems. This parallelization of the classic sequential multi-start heuristic is easy to implement and may offer satisfactory results. Co-operative strategies often offer superior performance, however (e.g., Crainic, Toulouse, and Gendreau 1995b, Crainic and Gendreau 2002).

pC/KS strategies are also generally implemented in a master-slave setting but attempt to take advantage of the parallel exploration by synchronizing processors at pre-determined intervals. The master process than collects information and usually restarts the search from the best solution (Malek *et al.* 1989 for the TSP, Rego and Roucairol 1996 for the VRP using ejection chains). De Falco *et al.* (1994 for the QAP), and De Falco, Del Balio, and Tarantino (1995 for the mapping problem) attempted to overcome the limitations of the master-slave setting. When each search thread terminates its local search, they synchronize and best solutions are exchanged between processes that run on neighbouring processes. Good performance and results were reported by all authors.

Asynchronous co-operative multi-thread search methods belong to the pC/C or pC/KC classes of the taxonomy according to the quantity and quality of

the information exchanged and, eventually, on the “new” knowledge inferred based on these exchanges. Most such developments use some form of *memory* for inter-thread communications (the term *blackboard* is also used sometimes). Each individual search thread starts from (usually) a different initial solution and generally follows a different search strategy. Exchanges are performed asynchronously and through the memory. Memories recording the performance of individual solutions, solution components, or even search threads may be added to the pool and statics may be gradually built. Moreover, various procedures may also be added to the pool to attempt to extract information or to create new informations and solutions based on the solutions exchanged. Co-operative multi-thread strategies implementing such methods belong to the pC/KC class.

One may classify co-operative multi-thread search methods according to the type of information stored in the central memory: complete or partial solutions. In the latter case, one often refers to *adaptive memory* strategies, while *central memory*, *pool of solutions*, *solution warehouse* or, even, *reference set* are terms used for the former. These methods have proved extremely successful on a broad range of problem types: real-time routing and vehicle dispatching (Gendreau *et al.* 1999), VRP with time windows (Taillard *et al.* 1997, Badeau *et al.* 1997, Schulze and Fahle 1999, Le Bouthiller and Crainic 2004), multicommodity location with balancing requirements (Crainic, Toulouse, and Gendreau 1995b), fixed cost, capacitated, multicommodity network design (Crainic and Gendreau 2002), and partitioning of integrated circuits for logical testing (Andreatta and Ribeiro 1994, Aiex *et al.* 1996, 1998, and Martins, Ribeiro, and Rodriguez 1996). One must notice, however, that the selection and the utilization of the information exchanged significantly impacts the performance of co-operating procedures. Co-operation mechanisms have to be carefully designed (Toulouse, Crainic, and Gendreau 1996, Toulouse, Thulasiraman, and Glover 1999). We return to these issues in the next section.

Although introduced for tabu search, the classification used in this paper applies to many other classes of meta-heuristics and refines the criteria based on the impact of parallelization on the search trajectory (e.g., Crainic and Toulouse 2003). Thus, it may form the basis for a comprehensive taxonomy of parallel meta-heuristics. This task, however, is beyond the scope of this paper.

3. Co-operation, Path Relinking, Scatter Search

A trend emerges from the collective efforts dedicated to parallel meta-heuristics in general and tabu search in particular. The main methodological focus moved from the low-level parallelism (e.g., the 1C/RS methods) of the initial developments, through a phase of domain decomposition approaches, on to independent search methods and co-operation (and hybridization). Of course, each type of parallelization strategy may play an important role given the particular problem

class and instance. In this section, we focus on co-operation, however, since it appears to offer both the most promising avenues for superior performances and the biggest challenges in terms of algorithm design.

Co-operative multi-thread tabu search methods launch several independent searches and implement information-exchange mechanisms among these threads. The general objective is to exchange *meaningful* information in a *timely* manner such that the global parallel search achieves a better performance than the simple concatenation of the results of the individual methods. Performance is measured in terms of computing time and solution quality (Barr and Hickman 1993, Crainic and Toulouse 2003).

As mentioned already, independent search methods implement multi-start sequential heuristics and, therefore, cannot achieve both performance objectives. Computation time is indeed reduced, but the same solution is reached. To overcome this limitation, Knowledge Synchronous pC strategies have been implemented where information is exchanged once all threads have completed their searches. The best overall solution is the only information exchanged and it serves as departure point for a new round of independent searches. The method is simple to implement and authors report good results. Computing times soar, however, and it would be interesting to thoroughly compare solution quality performance with co-operative methods. The limited evidence to this effect comes from applications to vehicle routing formulations and seem to indicate that co-operative strategies perform better in general.

Two other issues should be investigated in relation to these pC/KS strategies. First, the implementations reported in the literature start *new* searches following synchronization, cleaning up all memories. Given the importance of memories for tabu search, one wonders whether precious information is lost in the process. An investigation into what information to pass from one independent search phase to the next, how to use it, and its impact on performance might prove very interesting. The second issue has to do with the fact that, up to now, the only information exchanged when processes synchronize is the best solution. Yet, it has been shown that for co-operative methods this strategy may be counter-productive by concentrating most (all) threads on the same region of the solution space. This is an issue for further study as well.

Co-operation strategies require careful design as well as resources to manage the exchanges and process the data. A number of fundamental issues have to be addressed when designing co-operative parallel strategies for meta-heuristics (Toulouse, Crainic, and Gendreau 1996):

- What information is exchanged?
- Between what processes it is exchanged?
- When is information exchanged?

- How is it exchanged?
- How is the imported data used?

It is beyond the scope of this paper to address in detail all these issues. The reader may refer to the papers mentioned in this section for a more thorough description. It may be of interest, however, to sum up some of the pitfalls and less-than-winning strategies that have been identified in the literature so far.

Synchronous co-operative implementations tend to show a poorer performance compared to asynchronous and independent searches, especially when synchronization points are pre-determined (as in most current implementations). Such strategies display large computation overheads and a loss of efficiency. This is due to the obligation to wait for all processes to reach the synchronization point. The pre-definition of synchronization points also makes these strategies much less reactive to the particular problem instance and the progress of the search than asynchronous approaches and methods.

Asynchronous strategies are increasingly being proposed. Not all approaches offer the potential for success, however. To illustrate some of the issues, consider the following generic “broadcast and replace” strategy. When a search thread improves its local best solution, a message is sent to all other processes to stop their own exploration and import the broadcast solution. Then, if the imported solution is better than the local one, the search is restarted with the new solution. Several variants of this approach may be found in the literature. It has been observed, however, that interrupting the “local search” phase does not yield good results. The main reason is that one no longer performs a thorough search based on the local observation of the solution space. Rather, one transforms the tabu search into random search. This phenomenon is particularly disruptive if one permits the initial local search phase to be interrupted.

Stated in more general terms, the previous observation says that co-operative meta-heuristics with unrestricted access to shared knowledge may experience serious premature “convergence” difficulties, especially when the shared knowledge reduces to one solution only (the overall best or the new best from a given thread). This is due to a combination of factors: There is no global history or knowledge relative to the trajectory of the parallel search and thus each process has a (very) partial view of the entire search; Threads often use similar criteria to access the (same) “best” solution; Each process may broadcast a new best solution after a few moves only and thus disseminate information where only part of the solution has been properly evaluated. The contents of shared data tend then to become stable and biased by the best moves of the most performing threads and one observes a premature convergence of the dynamic search process. Moreover, the phenomenon may be observed whether one initializes the local memories following the import of an external solution or not (Toulouse

et al. 1998, Toulouse, Crainic, and Sansó 1999, 2004, Toulouse, Crainic, and Thulasiraman 2000).

Toulouse, Thulasiraman, and Glover (1999; see also Toulouse, Glover, and Thulasiraman 1998) proposed a new co-operation mechanism that attempts to address these challenges. The mechanism is called *multi-level co-operative search* and is based on the principle of controlled diffusion of information. Each search process works at a different level of aggregation of the original problem (one processor works on the original problem; the aggregation scheme ensures that a feasible solution at a level is feasible at the more disaggregated levels). Each search communicates exclusively with the processes working on the immediate higher and lower aggregation levels. Improved solutions are exchanged asynchronously at various moments dynamically determined by each process according to its own logic, status, and search history. An incoming solution will not be transmitted further until a number of iterations have been performed. The approach has proven very successful for graph partitioning problems (Ouyang *et al.* 2000, 2000a). Significant research is needed, however, to explore the various possibilities of the mechanisms, including the role of local and global memories.

Adaptive and *central* memory strategies also aim to overcome the limitations of straightforward co-operation by implementing mechanisms to build knowledge regarding the global search. This knowledge is then used to guide the exchange of information and, thus, to impact the trajectory of each individual process. The two approaches have much in common. In both approaches, there is no communication among the individual search processes: communication takes place exclusively between the memory process and individual search threads. Both approaches control when communication takes place (not very often and not during local search phases) and assign this responsibility to the individual searches. Based on the implementations reported in the literature, differences are mainly at the level of the information stored in the memory and its use.

As indicated in the previous section, adaptive memory implementations follow the general principles of the sequential adaptive memory strategy (Rochat and Taillard 1995) and store the elements of the best solutions found by the individual threads together with, eventually, memories counting the frequency of each element in the best solutions encountered so far. Processes communicate their best solutions at the normal end of their search. New starting solutions are built out of the elements in the memory. When frequency memories are implemented, they are used to bias the selection of the elements during the construction phase.

The central memory approach has less limitations. In theory at least since, in our opinion, the potential of this method has not been fully explored yet. As far as we can tell, Crainic, Toulouse, and Gendreau (1997) proposed the first

central memory strategy for tabu search. A more fully developed implementation is reported by Crainic and Gendreau (2002) for the fixed cost, capacitated, multicommodity network design problem. In both studies, the individual tabu search threads communicate the vector of design variables each time the local solution is improved, but import a solution from the central memory only before undertaking a diversification phase. Then, if the imported solution is better than the current best, the diversification proceeds from the new solution. Otherwise, the imported solution is discarded and the procedure proceeds as usual. Memories are never re-initialized. The authors compared five strategies of retrieving a solution from the pool when requested by an individual thread. The strategy that always returns the overall best solution displayed the best performance when few (4) processors were used. When the number of processors was increased, a probabilistic procedure, based on the rank of the solution in the pool, appears to offer the best performance. The parallel procedure improves the quality of the solution and also requires less (wall clock) computing time compared to the sequential version, particularly for large problems with many commodities.

Recently, Le Bouthiller and Crainic (2004) took this approach one step further and proposed a central memory parallel meta-heuristic for the VRP with time windows where several tabu search and genetic algorithm threads co-operate. In this model, the central memory constitutes the population common to all genetic threads. Each genetic algorithm has its own parent selection and crossover operators. The offspring are returned to the pool to be enhanced by a tabu search procedure. The tabu search threads and the central memory follow the same rules as in the work of Crainic and Gendreau (2002). Experimental results show that without any particular calibration, the parallel meta-heuristic obtains solutions whose quality is comparable to the best meta-heuristics available, and demonstrates almost linear speedups. These results indicate that central memory approaches apply equally well whether complex constraint structures complement the combinatorial characteristics of the problems studied or not.

Many enhancements could be added to these co-operation mechanisms and should be studied further. One such enhancement, proposed but not yet tested in the literature, concerns performance measures that could be attached to each individual search. Then, processes that do not contribute significantly neither to the quality of the solutions in the pool, nor to their diversity could be discarded. The computing resources thus recuperated could either be simply released or assigned to more productive duties (e.g., speed up computations of some critical search steps or initiate new search threads modeled on the most successful ones). A second possibility concerns performance statistics attached to individual solutions (or parts thereof) measuring, for example, the relative improvement of solutions generated by individual threads when starting from them. Another open question concerns the global search memories that could

be inferred from the records built by individual searches and how the global search trajectory could then be inflected to yield superior results.

It has been noted that the pool of solutions constitutes a *population*. Then, other than the usual construction heuristics, population-based methods may be used to generate new solutions and improve the pool. Crainic and Gendreau (1999) report the development of such a hybrid search strategy combining their co-operative multi-thread parallel tabu search method with a genetic engine. The genetic algorithm initiates its population with the first elements from the central memory of the parallel tabu search. Asynchronous migration (migration rate = 1) subsequently transfers the best solution of the genetic pool to the parallel tabu search central memory, as well as solutions of the central memory towards the genetic population. Even though the strategy is not very elaborate, the hybrid appears to perform well, especially on larger problems. It is noteworthy that the genetic algorithm alone was not performing well and that it was the parallel tabu search procedure that identified the best results once the genetic method contributed to the quality of the central memory.

This combination of individual tabu search threads and population-based methods that run on the set of solutions in the central memory appears very promising and should be investigated in depth. (In more general terms, this strategy applies equally well to many meta-heuristic classes and is the object of research in the genetic and simulated annealing communities.) Different population-based methods could be used, *path relinking* and *scatter search* (Glover 1997, Glover and Laguna 1997, Glover, Laguna, and Martí 2000, Laguna and Martí 2003) being prime candidates. Both methods have proved highly successful on a wide spectrum of difficult problems. Moreover, the close relationships displayed by the two methods would facilitate the exchanges with co-operating tabu search threads. This topic must be thoroughly investigated.

This brings forth the issue of the parallelization of path relinking and scatter search. In their sequential versions, both methods make use of an initialization phase to generate "good" solutions, as well as of a *reference* or *elite* set of solutions continuously updated during the search. A path relinking method will select two solutions, a starting solution and a target one, and use a tabu search to move from the starting to the target solution. Moves are biased to favor gradually entering "good" attributes of the target solution into the current one. Scatter search, selects a number of solutions from the reference set and combines them to yield a new solution that may be improved by a local or tabu search algorithm. Solutions with "good" attributes, including overall best ones, are included by each method in the corresponding reference set.

Research on parallel path relinking and parallel scatter search is very scarce, yet. Of course, the strategies already presented for tabu search may be applied to path relinking and scatter search. Low level, 1-control, strategies, as well as multi-search approaches involving several path relinking or scatter search

threads, are straightforward. Co-operative strategies offer a greater challenge. Indeed, the reference set corresponds to a central memory co-operation mechanism that may be used by several path relinking or scatter search threads. Issues to be studied concern the initialization of the reference set and of each thread, as well as the particular definition of each thread. Thus, for example, one could consider the main algorithmic parts of scatter search – the selection of candidates, the generation of new solutions (different searches could combine a different number of solutions), and their possible improvement by a tabu search procedure – as “separate” procedures. A large number of co-operation designs become possible and should be investigated thoroughly. Also worthy of serious research is the role of memories in co-operation settings involving path relinking and scatter search methods, as well as the strategies that have the two methods co-operate through a unique reference set.

4. Perspectives and Research Directions

We have presented strategies to design parallel tabu search algorithms and surveyed developments and results in the area. Parallel tabu search methods offer the possibility to address problems more efficiently, both in terms of computing efficiency and solution quality. Low-level parallelization strategies appear often beneficial to speed up computation-intensive tasks, such as the evaluation of potential moves in the neighbourhood of a given solution. Moreover, such strategies may be advantageously incorporated into hierarchical parallel schemes where the higher level method either explores partitions of the solution domain or implements a co-operating multi-thread search.

Co-operation and multi-thread parallelization appear to offer the most interesting perspectives for tabu search, as for meta-heuristics in general. Asynchronous co-operative multi-thread strategies constitute probably the strongest trend in parallel meta-heuristics, the results reported in the literature indicating that they offer better results than synchronous and independent searches. They also seem to offer the most interesting development avenue for advanced tabu search methods, including path relinking and scatter search. More theoretical and empirical work is still required in this field, however. Consequently, in the second part of the paper, we focussed on multi-search strategies. We discussed general design and implementation principles, pointed out a number of challenges and pitfalls, and identified trends and promising research directions.

Hybridization is another important trend in sequential and parallel meta-heuristics and recent studies tend to demonstrate that combining different meta-heuristics yields superior results. This opens up an exciting field of enquiry. What meta-heuristics to combine with tabu search? What role can each type of meta-heuristic play? What information is exchanged and how it is used in this context? How important a role path relinking and scatter search may play for

parallel meta-heuristics in general? These are only a few of the questions that need to be explored.

It has been often noticed that co-operating parallel mechanisms bear little, if any, resemblance to the initial meta-heuristic one attempts to parallelize. This remark is true whether the individual search threads belong to the same class of meta-heuristics or not. A natural question then is whether co-operating multi-thread parallel methods should form a "new", very broadly defined, class of meta-heuristics.

Is co-operative multi-thread parallelization a "new" meta-heuristic? Almost certainly, even though we face the challenge to properly define it. Is it a "new" tabu search class? Not as currently used. Indeed, the method misses a global search control mechanism and, especially, the memories that would guide this mechanism are a central component of tabu search. The development of such memories and mechanisms is one of the most interesting challenges we face.

Acknowledgments

Funding for this project has been provided by the Natural Sciences and Engineering Council of Canada and by the Fonds F.C.A.R. of the Province of Québec. While working on the project, Dr. T.G. Crainic was Adjunct Professor at the Département d'Informatique et de Recherche Opérationnelle of the Université de Montréal (Canada).

References

- Aiex, R.M., S.L. Martins, C.C. Ribeiro and N.R. Rodriguez (1996) Asynchronous Parallel Strategies for Tabu Search Applied to the Partitioning of VLSI Circuits, *Monografias em ciéncia da computação*, Pontifícia Universidade Católica de Rio de Janeiro.
- Aiex, R.M., S.L. Martins, C.C. Ribeiro and N.R. Rodriguez (1998) "Cooperative Multi-Thread Parallel Tabu Search with an Application to Circuit Partitioning," In *Proceedings of IRREGULAR'98 - 5th International Symposium on Solving Irregularly Structured Problems in Parallel, Lecture Notes in Computer Science*, Springer-Verlag, 1457:310–331.
- Andreatta, A.A. and C.C. Ribeiro (1994) "A Graph Partitioning Heuristic for the Parallel Pseudo-Exhaustive Logical Test of VLSI Combinational Circuits," *Annals of Operations Research*, 50:1–36.
- Badeau, P., F. Guertin, M. Gendreau, J.Y. Potvin and É.D. Taillard (1997) "A Parallel Tabu Search Heuristic for the Vehicle Routing Problem with Time Windows," *Transportation Research C: Emerging Technologies*, 5(2):109–122.

- Barr, R.S. and B.L. Hickman (1993) "Reporting Computational Experiments with Parallel Algorithms: Issues, Measures, and Experts Opinions," *ORSA Journal on Computing*, 5(1):2–18.
- Battiti, R. and G. Tecchiolli (1992) "Parallel Based Search for Combinatorial Optimization: Genetic Algorithms and TABU," *Microprocessors and Microsystems*, 16(7):351–367.
- Chakrapani, J. and J. Skorin-Kapov (1992) "A Connectionist Approach to the Quadratic Assignment Problem," *Computers & Operations Research*, 19(3/4):287–295.
- Chakrapani, J. and J. Skorin-Kapov (1993) "Connection Machine Implementation of a Tabu Search Algorithm for the Traveling Salesman Problem," *Journal of Computing and Information Technology*, 1(1):29–36.
- Chakrapani, J. and J. Skorin-Kapov (1993). "Massively Parallel Tabu Search for the Quadratic Assignment Problem," *Annals of Operations Research*, 41:327–341.
- Chakrapani, J. and J. Skorin-Kapov (1995) "Mapping Tasks to Processors to Minimize Communication Time in a Multiprocessor System," In *The Impact of Emerging Technologies of Computer Science and Operations Research*, Kluwer Academic Publishers, Norwell, MA, 45–64.
- Crainic, T.G. and M. Gendreau (1999) "Towards an Evolutionary Method - Co-operating Multi-Thread Parallel Tabu Search Hybrid," In S. Voß, S. Martello, C. Roucairol and I.H. Osman, Editors, *Meta-Heuristics 98: Theory & Applications*, Kluwer Academic Publishers, Norwell, MA, 331–344.
- Crainic, T.G. and M. Gendreau (2002) "Cooperative Parallel Tabu Search for Capacitated Network Design," *Journal of Heuristics*, 8(6):601–627.
- Crainic, T.G. and M. Toulouse (1998) *Parallel Metaheuristics*, In T.G. Crainic and G. Laporte, Editors, *Fleet Management and Logistics*, Kluwer Academic Publishers, Norwell, MA, 205–251.
- Crainic, T.G. and M. Toulouse (2003) *Parallel Strategies for Meta-heuristics*, In F. Glover and G. Kochenberger, Editors, *State-of-the-Art Handbook in Metaheuristics*, Kluwer Academic Publishers, Norwell, MA, 475–513.
- Crainic, T.G., M. Toulouse and M. Gendreau (1995) "Parallel Asynchronous Tabu Search for Multicommodity Location-Allocation with Balancing Requirements," *Annals of Operations Research*, 63:277–299.
- Crainic, T.G., M. Toulouse and M. Gendreau (1995) "Synchronous Tabu Search Parallelization Strategies for Multicommodity Location-Allocation with Balancing Requirements," *OR Spektrum*, 17(2/3):113–123.
- Crainic, T.G., M. Toulouse and M. Gendreau (1997) "Towards a Taxonomy of Parallel Tabu Search Algorithms," *INFORMS Journal on Computing*, 9(1):61–72.
- Cung, V.D., S.L. Martins, C.C. Ribeiro and C. Roucairol (2002) "Strategies for the Parallel Implementations of Metaheuristics," In C. Ribeiro and P.

- Hansen, Editors, *Essays and Surveys in Metaheuristics*, Kluwer Academic Publishers, Norwell, MA, 263–308.
- De Falco, I., R. Del Balio and E. Tarantino (1995) Solving the Mapping Problem by Parallel Tabu Search. Report, Istituto per la Ricerca sui Sistemi Informatici Paralleli-CNR.
- De Falco, I., R. Del Balio, E. Tarantino and R. Vaccaro (1994) "Improving Search by Incorporating Evolution Principles in Parallel Tabu Search," In *Proceedings International Conference on Machine Learning*, 823–828.
- Fiechter, C.N. (1994) "A Parallel Tabu Search Algorithm for Large Travelling Salesman Problems," *Discrete Applied Mathematics*, 51(3):243–267.
- Garcia, B.L., J.Y. Potvin and J.M. Rousseau (1994) "A Parallel Implementation of the Tabu Search Heuristic for Vehicle Routing Problems with Time Window Constraints," *Computers & Operations Research*, 21(9):1025–1033.
- Gendreau, M., F. Guertin, J.Y. Potvin and É.D. Taillard (1999) "Tabu Search for Real-Time Vehicle Routing and Dispatching," *Transportation Science*, 33(4):381–390.
- Glover, F. (1986) "Future Paths for Integer Programming and Links to Artificial Intelligence," *Computers & Operations Research*, 1(3):533–549.
- Glover, F. (1989) "Tabu Search – Part I," *ORSA Journal on Computing*, 1(3):190–206.
- Glover, F. (1990) "Tabu Search – Part II," *ORSA Journal on Computing*, 2(1):4–32.
- Glover, F. (1996) "Tabu Search and Adaptive Memory Programming – Advances, Applications and Challenges," In R. Barr, R. Helgason and J. Kennington, Editors, *Interfaces in Computer Science and Operations Research*, Kluwer Academic Publishers, Norwell, MA, 1–75.
- Glover, F. (1997) "A Template for Scatter Search and Path Relinking," In J. Hao, E. Lutton, E. Ronald, M. Schoenauer and D. Snyers, Editors, *Artificial Evolution, Lecture Notes in Computer Science*, Springer Verlag, Berlin, 1363:13–54.
- Glover, F. and M. Laguna (1993) "Tabu Search," In C. Reeves, Editor, *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Publications, Oxford, 70–150.
- Glover, F. and M. Laguna (1997) *Tabu Search*. Kluwer Academic Publishers, Norwell, MA.
- Glover, F., M. Laguna and R. Martí (2000) "Fundamentals of Scatter Search and Path Relinking," *Control and Cybernetics*, 39(3):653–684.
- Glover, F., É.D. Taillard and D. de Werra (1993) "A User's Guide to Tabu Search," *Annals of Operations Research*, 41:3–28.
- Holmqvist, K., A. Migdalas and P.M. Pardalos (1997) "Parallelized Heuristics for Combinatorial Search," In A. Migdalas, P. Pardalos and S. Storoy, Editors,

- Parallel Computing in Optimization*, Kluwer Academic Publishers, Norwell, MA, 269–294.
- Laguna, M. and R. Martí (2003) *Scatter Search: Methodology and IMplementations in C*. Kluwer Academic Publishers, Norwell, MA.
- Le Bouthillier, A. and T.G. Crainic (2004) "A Cooperative Parallel Meta-Heuristic for the Vehicle Routing Problem with Time Windows," *Computers & Operations Research*.
- Malek, M., M. Guruswamy, M. Pandya and H. Owens (1989) "Serial and Parallel Simulated Annealing and Tabu Search Algorithms for the Traveling Salesman Problem," *Annals of Operations Research*, 21:59–84.
- Martins, S.L., C.C. Ribeiro and N.R. Rodriguez (1996) Parallel Programming Tools for Distributed Memory Environments, Monografias em Ciéncia da Computação , Pontifícia Universidade Católica de Rio de Janeiro.
- Ouyang, M., M. Toulouse, K. Thulasiraman, F. Glover and J.S. Deogun (2000) "Multi-Level Cooperative Search: Application to the Netlist/Hypergraph Partitioning Problem," In *Proceedings of International Symposium on Physical Design*, ACM Press, 192–198.
- Ouyang, M., M. Toulouse, K. Thulasiraman, F. Glover and J.S. Deogun (2002) "Multilevel Cooperative Search for the Circuit/Hypergraph Partitioning Problem," *IEEE Transactions on Computer-Aided Design*, 21(6):685–693.
- Pardalos, P.M., L. Pitsoulis, T. Mavridou and M.G.C. Resende (1995) "Parallel Search for Combinatorial Optimization: Genetic Algorithms, Simulated Annealing, Tabu Search and GRASP," In A. Ferreira and J. Rolim, Editors, *Proceedings of Workshop on Parallel Algorithms for Irregularly Structured Problems, Lecture Notes in Computer Science 980*, Springer-Verlag, Berlin, 317–331.
- Porto, S.C.S., J.P.F.W. Kitajima and C.C. Ribeiro (2000) "Performance Evaluation of a Parallel Tabu Search Task Scheduling Algorithm," *Parallel Computing*, 26:73–90.
- Porto, S.C.S. and C.C. Ribeiro (1995) "A Tabu Search Approach to Task Scheduling on Heterogenous Processors Under Precedence Constraints," *International Journal of High-Speed Computing*, 7:45–71.
- Porto, S.C.S. and C.C. Ribeiro (1996) "Parallel Tabu Search Message-Passing Synchronous Strategies for Task Scheduling Under Precedence Constraints," *Journal of Heuristics*, 1(2):207–223.
- Rego, C. and C. Roucairol (1996) "A Parallel Tabu Search Algorithm Using Ejection Chains for the VRP," In I. Osman, and J. Kelly, Editors, *Meta-Heuristics: Theory & Applications*, Kluwer Academic Publishers, Norwell, MA, 253–295.
- Rochat, Y. and É.D. Taillard (1995) "Probabilistic Diversification and Intensification in Local Search for Vehicle Routing," *Journal of Heuristics*, 1(1):147–167.

- Schulze, J. and T. Fahle (1999) "A Parallel Algorithm for the Vehicle Routing Problem with Time Window Constraints," *Annals of Operations Research*, 86:585–607.
- Taillard, É.D. (1991) "Robust Taboo Search for the Quadratic Assignment Problem," *Parallel Computing*, 17:443–455.
- Taillard, É.D. (1993) "Parallel Iterative Search Methods for Vehicle Routing Problems," *Networks*, 23:661–673.
- Taillard, É.D. (1993) *Recherches itératives dirigées parallèles*. PhD thesis, École Polytechnique Fédérale de Lausanne.
- Taillard, É.D. (1994) "Parallel Taboo Search Techniques for the Job Shop Scheduling Problem," *ORSA Journal on Computing*, 6(2):108–117.
- Taillard, É.D., P. Badeau, M. Gendreau, F. Guertin and J.Y. Potvin (1997) "A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows," *Transportation Science*, 31(2):170–186.
- Toulouse, M., T.G. Crainic and M. Gendreau (1996) "Communication Issues in Designing Cooperative Multi Thread Parallel Searches," In I.H. Osman and J.P. Kelly, Editors, *Meta-Heuristics: Theory & Applications*, Kluwer Academic Publishers, Norwell, MA, 501–522.
- Toulouse, M., T.G. Crainic and B. Sansó (1999) "An Experimental Study of Systemic Behavior of Cooperative Search Algorithms," In S. Voß, S. Martello, C. Roucairol and I.H. Osman, Editors, *Meta-Heuristics 98: Theory & Applications*, Kluwer Academic Publishers, Norwell, MA, 373–392.
- Toulouse, M., T.G. Crainic and B. Sansó (2004) "Systemic Behavior of Cooperative Search Algorithms," *Parallel Computing*, 21(1):57–79.
- Toulouse, M., T.G. Crainic, B. Sansó and K. Thulasiraman (1998). "Self-Organization in Cooperative Search Algorithms," In *Proceedings of the 1998 IEEE International Conference on Systems, Man, and Cybernetics*, Omni-press, Madison, Wisconsin, 2379–2385.
- Toulouse, M., T.G. Crainic and K. Thulasiraman (2000) "Global Optimization Properties of Parallel Cooperative Search Algorithms: A Simulation Study," *Parallel Computing*, 26(1):91–112.
- Toulouse, M., F. Glover and K. Thulasiraman (1998) "A Multi-Scale Cooperative Search with an Application to Graph Partitioning," Report, School of Computer Science, University of Oklahoma, Norman, OK.
- Toulouse, M., K. Thulasiraman and F. Glover (1999). "Multi-Level Cooperative Search," In P. Amestoy, P. Berger, M. Daydé, I. Duff, V. Frayssé, L. Giraud and D. Ruiz, Editors, *5th International Euro-Par Parallel Processing Conference, Lecture Notes in Computer Science*, Springer-Verlag, Berlin, 1685:533–542.
- Verhoeven, M.G.A. and E.H.L. Aarts (1995) "Parallel Local Search," *Journal of Heuristics*, 1(1):43–65.

Voß, S. (1993) "Tabu Search: Applications and Prospects," In D.Z. Du and P. Pardalos, Editors, *Network Optimization Problems*, World Scientific Publishing Co., Singapore, 333–353.

Chapter 14

USING GROUP THEORY TO CONSTRUCT AND CHARACTERIZE METAHEURISTIC SEARCH NEIGHBORHOODS

Bruce W. Colletti and J. Wesley Barnes

*Graduate Program in Operations Research and Industrial Engineering, The University of Texas at Austin, Austin TX 78712, bcolletti@compuserve.com,
wbarnes@mail.utexas.edu*

Abstract: Over the past four years, a Consortium composed of The University of Texas at Austin, the Air Force Institute of Technology and the US Air Force Air Mobility Command has been remarkably successful in developing the theoretical foundations of group theoretic tabu search (GTTS) and applying it to a set of complex military logistics problems. This paper briefly recounts some of those accomplishments and describes the underlying mathematical framework that supported them. The symmetric group on n letters, S_n , can be used to build and solve equations whose coefficients and variables are permutations, i.e., elements of S_n . These equations efficiently describe search neighborhoods for combinatorial optimization problems (COPs) whose solutions can be modeled as partitions, orderings, or both (P|O problems). Following the introduction, the second section describes neighborhoods that preserve a cycle solution structure and recounts examples of how group theory has been used to provide new and useful insights into search neighborhoods. The third section describes neighborhoods for multiple cyclic solution structures. Section 4 considers a hybrid move neighborhood that borrows from Sections 2 and 3. The fifth section overviews some highly successful applications of the GTTS approach. The final section contains concluding remarks and offers suggestions for future work. The paper's goals are: (1) to illustrate that move neighborhoods can be cast into a group theoretic context and that new powerful neighborhoods can be easily "hand tailored" using group theory; and (2) to motivate others to consider how the unifying mathematical framework of group theory could be further exploited to gain powerful new understandings of direct search move neighborhoods.

Keywords: Metaheuristics, Group Theory, Traveling Salesman Problem (TSP), Tabu Search, Combinatorial Optimization

1. Introduction

Group theory, the "algebra of permutations", can powerfully enhance the study, understanding and application of metaheuristic search neighborhoods. This statement is abundantly illustrated in the documents referenced in Table 14.1. In the following sections, each of these papers will be summarized in appropriate detail.

Table 14.1 Papers associated with group theory & metaheuristic neighborhoods

Colletti and Barnes (1999)	Group Theory & Metaheuristic Neighborhoods
Colletti, Barnes and Dokov (1999)	Characterizing the k-OPT Neighborhood Via Group Theory
Colletti and Barnes (2000)	Linearity in the Traveling Salesman Problem
Colletti and Barnes (2001)	Local Search Structure in the Symmetric TSP
Colletti, Barnes and Neuway (2000)	Group theory & Transition Matrices of Metaheuristic Neighborhoods
Barnes, Wiley, Moore and Ryer (2002)	Solving the Aerial Fleet Refueling Problem using GTTS
Crino, Moore, Barnes and Nanry (2002)	Solving The Military Theater Distribution Problem Using GTTS

While the final two papers listed in Table 14.1 present the use of GTTS in the efficient and effective solution of much more complex COPs occurring in compelling and important *real-world* scenarios, for simplicity and clarity of explanation, we will initially focus our consideration on the special case of the classical m-TSP where the agents do not share a common base or depot city (Gilmore, Lawler, Shmoys 1985). Rather, each agent is based at one of the cities in the subtour, or cycle, assigned to that agent. Let us stipulate an associated arbitrary zero-diagonal real distance matrix and denote this problem as the "cyclic" n-city, m-agent traveling salesperson problem (m-CTSP). (When $m = 1$, and presuming the depot city is one of the n cities, the m-CTSP becomes identical to the classical 1-TSP.) Some agents may be idle but no more than m cycles are allowed. In this paper, we will use this problem as our illustrative foundation. Colletti and Barnes (1999) show how the m-CTSP results may easily be adapted for use with more general P|O problems such as the classical m-TSP where all agents have a common base.

Although group theory is the foundation of several exact methods of integer and mixed-integer programming (Glover 1969; Gomory 1958, 1965, 1969; Salkin 1975; White 1966; Wolsey 1969), it has found limited use in heuristic and metaheuristic methods. In classical integer programming approaches, the parent group is abelian (commutative). The flexibility of that parent group's many associated factor groups provided a readily available path for the construction of such things as

Gomory cuts. However, the parent group for heuristics applied to P|O problems, the symmetric group on n letters, S_n , is nonabelian possessing a single unprofitably small factor group. This lack of factor groups associated with S_n forces markedly different group theoretic approaches from those employed for exact techniques. (For completeness, an appendix briefly sketches the group theoretic concepts essential to the understanding of the body of the paper.)

Section 2 provides a general description of the class of *conjugative rearrangement* neighborhoods that preserve the *cycle structure* of the incumbent solution, i.e., all candidate neighbor solutions must possess the same number of cycles and each cycle must possess the same number of cities as in the incumbent solution. This description is followed by an overview of Colletti and Barnes (2000, 2001) and Colletti, Barnes and Neuway (2000) to illustrate how group theory can be used to provide new and useful insights into the character of metaheuristic neighborhoods. In the remainder of Section 2, three other specific types of rearrangement neighborhoods are presented in their equivalent group theoretic form.

Section 3 considers neighborhoods that do not preserve the cycle structure of the incumbent solution and presents a general class of group theoretic search neighborhoods achievable through the application of splitting and welding *templates*. The k-Or neighborhood is given as a specific example of this class and two other representative move neighborhood types are presented in their equivalent group theoretic form.

Section 4 uses group theory to present a hybrid neighborhood that uses ideas from Sections 2 and 3. This hybrid neighborhood reveals a very compact representation of a k-arc exchange neighborhood for arbitrary values of k .

Section 5 presents overviews of Barnes et al. (2002) and Crino et al. (2002) where the problems attacked, the neighborhoods developed and used, and the relative effectiveness and efficiencies of the solution methodologies are discussed. The sixth and final Section provides concluding remarks and a glimpse of future research goals.

2. Neighborhoods that Preserve Cycle Structure

Group theory may be directly applied to P|O problems by using S_n , whose elements make up all possible partitions and orderings of the n objects composing the solution representation. For example, one group element, or *permutation*, of S_{11} is the partition of the 11 objects onto four subsets, $p = (2,3,7)(1,6,5,4)(9,8,11)(10) = (2,3,7)(1,6,5,4)(9,8,11)$, where, by convention, unit cycles are not explicitly written. p has four subtours or *cycles* in which each letter is mapped into its successor.

- Swap cities 2 & 4 in $p = (1,2,3,4,5,6)$ to yield $q = p^m = p^{(2,4)} = (1,4,3,2,5,6)$
- Cyclically rearrange 2,4,& 6 in $p = (1,2,3,4,5,6)$ to yield

- Similarly, $q = p^m = p^{(2,6,4)} = (1,6,3,2,5,4)$

Figure 14.1. Three examples of rearrangement moves

Letter rearrangement moves are often used to build TSP neighborhoods. For example, the 2-letter swap neighborhood of tour p is composed of all tours obtained by swapping all letter pairs in p . Such moves are described via conjugation, i.e., if $p, q \in$ group G , then $p^q \equiv q^{-1}pq$ is the *conjugate of p by q* . Conjugation partitions a group into mutually exclusive and exhaustive *conjugacy classes*. For S_n , a conjugacy class consists of all permutations that share a common cycle structure. Figure 14.1 gives three examples of simple rearrangement moves for a six city TSP.

When $G = S_n$, p and p^q have the same cycle structure. In addition to computing $p^q = q^{-1}pq$, p^q may also be found by replacing each letter in p by its image under q . Hence conjugation corresponds to letter rearrangement moves. For example, $[(1, 2) (3, 4)]^{(3, 7)(6, 9, 1)(4, 5)} = (6, 2) (7, 5)$.

2.1 Letter Rearrangement Moves

The neighborhood of all possible k -letter rearrangements on derangement $p \in S_n$ is given by p^C , where C is the union of all conjugacy classes in S_n whose cycle structures move k letters (any permutation in S_n that moves all n letters is called a *derangement*). For example, all letter pair swaps correspond to $C \equiv$ the conjugacy class of all transpositions. All 6-letter rearrangements correspond to $C \equiv$ the union of conjugacy classes whose cycle structures are (x, x, x, x, x, x) , $(x, x)(x, x, x, x)$ and $(x, x, x)(x, x, x)$.

We now recount the results of some recent research associated with letter rearrangement moves (Colletti and Barnes 2000, 2001; Colletti, Barnes and Neuway 2000) to provide examples that substantiate the relevance of applying group theory to the study of metaheuristic neighborhoods.

Colletti and Barnes (2000) study the class of general letter rearrangement neighborhoods for the asymmetric TSP under conjugation by a union of conjugacy classes. They use S_n to show that the incumbent tour's neighborhood *weight* (the neighbors' summed tourlengths) is linear in the tourlengths of the incumbent and its inverse (it is important to note that any duplicate neighbors are retained in these

computations). If one is concerned about the symmetric TSP, this result confirms that the summed tourlengths of neighborhood solutions is a linear function of the tourlength of the incumbent tour. The coefficients of the tourlengths of the incumbent and its inverse are only dependent upon the elements of C and value of n . The fact that the linear coefficients are not dependent upon the particular distance matrix at hand is enticing. There may very well be deeper knowledge present in this observation that will yield even more powerful general structure between search neighborhoods and metaheuristic methods. Many search strategies are based on the presumption that an incumbent tour's length will directly reflect its neighborhood weight, i.e., a good tour will be found in a neighborhood of good tours. The above relationship brings to light a counterintuitive fact. Under some letter rearrangement neighborhood types, the neighborhood's weight is inversely related (has a negative slope) to the incumbent tourlength, i.e., a very inferior solution (with a relatively large tourlength) could have a small neighborhood weight, which implies the existence of short tours in the neighborhood. Interestingly, the swap neighborhood possesses the most positive slope of all such rearrangement neighborhoods. This may explain why so much success has been seen in the application of the swap neighborhood in the literature. The ramifications of this linearity property remain largely unexplored. (One associated result, as described later, is that this property has led to the confirmation of an extended conjecture regarding the quality of local optima in rearrangement neighborhoods (Colletti and Barnes 2001). It is now known that for the general asymmetric m -TSP over derangements, linear coefficients are unaffected by the cycle structure of the solution space, a somewhat surprising result. When solutions are not derangements, the linearity is "setwise linear" over solutions that move common letters (regardless of sequencing and cycle structure).

There are other observations. First, with respect to the symmetric m -TSP over derangements, when two rearrangement move methods yield very different neighborhoods, each may still "reach" the global optimum at the same rate. That is, if the methods' linear equations share a common slope, then both have the same rate of change with respect to incumbent tourlength. If so, the larger neighborhood method may offer little advantage.

The second observation is that it is highly likely that other fundamental properties of rearrangement move methods remain to be found using group theory. For example, does the linearity property persist when the rearrangement move is defined in terms of a conjugacy class of a subgroup of S_n instead of an entire conjugacy class of S_n ? If so, what is the advantage of using such a smaller conjugacy class? This line of questioning has already led to the study of neighborhoods defined as orbits of group actions, some of whose results appear herein.

A final observation is that the weight of a rearrangement neighborhood can now be swiftly computed (directly) without having to add neighbor tourlengths. In turn, this suggests that group theory may reveal a move neighborhood's macroscopic properties free from the piecemeal evaluation of individual neighbors.

Codenotti and Margara (1992) and Grover (1992) reveal four specific simple symmetric 1-TSP neighborhoods that satisfy Grover's homogeneous linear difference wave equation and so have no arbitrarily poor local optima, i.e., no local optimum will exceed the average tourlength of all feasible solutions.

Colletti and Barnes (2001) use the perspective of group theory to extend these results to a general theory that embraces the above four symmetric 1-TSP neighborhoods as special cases. Indeed, Colletti and Barnes (2001) use group theory to prove that any conjugative letter rearrangement neighborhood for the symmetric 1-TSP has the property of no arbitrarily poor local optima. Colletti (1999) generalizes the above 1-TSP results to the symmetric m-TSP whose solution space is any conjugacy class of derangements (n-cycles are one type of derangement).

This property is as surprising as it is useful: surprising because it simply depends upon the move definition – distance matrix and feasible solution space properties have no influence. The property *always exists* for symmetric m-TSP rearrangement neighborhoods over derangements, an unintuitive result. Instead, intuition wrongly suggests that there may well be some extremely lengthy incumbent whose neighbors' tourlengths are well above average. Thus, if a move strategy ends up with an incumbent whose tourlength is longer than the average, then an improving tour will *always* be found among the 2-city swap neighbors. Finally, as noted before, this property leads to questions about what other fundamental "invariant" properties are possessed by rearrangement neighborhoods and how they can be revealed by group theory.

Without the unifying overview structure provided by group theory, this same result could only be provided by deriving the result *one neighborhood at a time*. Thus, in Colletti and Barnes (2000, 2001), the use of group theory reveals two insights into metaheuristic neighborhoods that would be difficult, if not impossible, to obtain in other ways.

Colletti, Barnes and Neuway (2000) study rearrangement neighborhood transition matrices. For example, consider swap moves (all possible rearrangements of two cities) on a 5 city 1-TSP. In this case solutions are made up of the conjugacy class of 5-cycles, i.e., single agent tours, where the agent visits all five cities (and we assume the agent is based at one of the cities). In this case there are $4! = 24$ tours indexed in dictionary order as:

(1, 2, 3, 4, 5), (1, 2, 3, 5, 4), (1, 2, 4, 3, 5), (1, 2, 4, 5, 3), (1, 2, 5, 3, 4), (1, 2, 5, 4, 3), (1, 3, 2, 4, 5), (1, 3, 2, 5, 4), (1, 3, 4, 2, 5), (1, 3, 4, 5, 2), (1, 3, 5, 2, 4), (1, 3, 5, 4, 2), (1, 4, 2, 3, 5), (1, 4, 2, 5, 3), (1, 4, 3, 2, 5), (1, 4, 3, 5, 2), (1, 4, 5, 2, 3), (1, 4, 5, 3, 2), (1, 5, 2, 3, 4), (1, 5, 2, 4, 3), (1, 5, 3, 2, 4), (1, 5, 3, 4, 2), (1, 5, 4, 2, 3), (1, 5, 4, 3, 2)

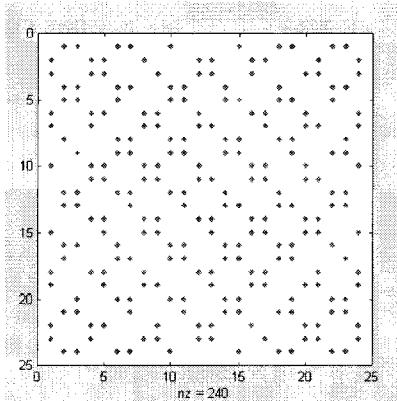


Figure 14.2. The “Raw” swap neighborhood for 5 city 1-TSP

Each of the 24 tours has ten one step neighbors, as pictured in the uninformative schematic transition matrix given in Figure 14.2. However, by using group theory to find the orbits of a group action of the alternating group, A_n , acting upon all 24 tours, we can use the orbits to partition the tours into two essential classes of the same size. Identically permuting the rows and columns of the transition matrix in accordance with this partition (with some minor reordering) yields the periodic transition matrix of order 2 presented in Figure 14.3.

This means that a swap move neighborhood on a 5 city TSP “communicates”, i.e., we can get to any other tour starting at an arbitrary tour. However, in one step you can move only from one essential class to the other.

Figure 14.4 presents the transition matrix, viewed through the perspective of group theory, for the rearrangement neighborhood of order 3, i.e., all possible 3-city rearrangements. This nonintuitive result shows that the 3-city rearrangement neighborhood does not communicate! You can never depart the essential class in which you started. Half the tours are unreachable from any specific tour. If n is an odd number, this surprising structure is present for any n city 1-TSP when the 3-city rearrangement neighborhood is used!

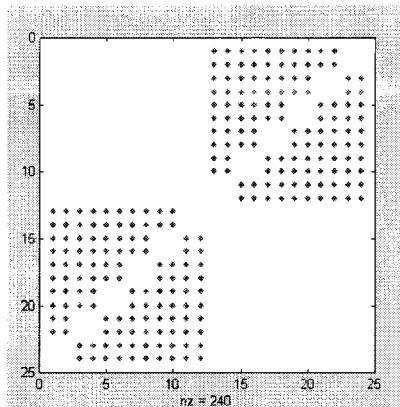


Figure 14.3. The transition matrix revealed!

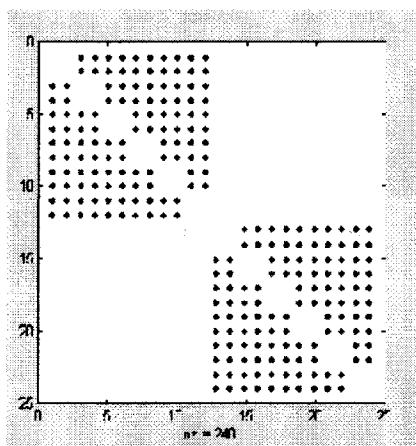


Figure 14.4. The transition matrix for the order 3 rearrangement neighborhood

If your starting solution is not in the same essential class with an optimal tour, you can *not* find an optimal solution. However, we may view this structure from a more favorable perspective. If we are aware of this structure, the search for the optimum may be attacked as two independent subproblems with communicating solution neighborhoods of dimensionality $\frac{(n-1)!}{2}$ by $\frac{(n-1)!}{2}$. For larger values of n , this is considerably less formidable than the original solution neighborhood with dimension $(n-1)!$ by $(n-1)!$!

Using group theory, Colletti, Barnes and Neuway (2000) prove that for any n-city 1-TSP, if you are using conjugative rearrangement moves with a single conjugacy class defining the move, you either will have a single communicating essential class, or 2 essential classes. If you have 2 classes they will either communicate with period 2 or they will form two noncommunicating classes. They also derive the detailed, specific conditions on the defining conjugacy class and the values of n that lead to each of these structures. (It may be shown that very similar structures are present for the m-TSP with n cities.)

Using group theory greatly facilitated both the direct discovery of these general structures and proof of their validity for the general n city problem.

2.2 Special Rearrangement Neighborhoods

We now turn our attention to the development of other specific rearrangement moves from the group theory perspective.

2.2.1 Reversal Neighborhoods

Given an m-CTSP tour p , suppose there are specified disjoint subpaths to reverse, e.g., $[2, 3, 4, 5, 6]$ and $[9, 10, 11]$ in $p = (1, 2, 3, 4, 5, 6, 7)(8, 9, 10, 11, 12)$ to obtain $(1, 6, 5, 4, 3, 2, 7)(8, 11, 10, 9, 12)$. Lin and Kernighan (1973) consider such reversals in their classical heuristic for the symmetric 1-TSP. Group theory allows us to algebraically describe the neighborhood of p , which reverses all combinations of specified subpaths.

Reversing the p -subpath $[a_1, \dots, a_m]$ requires two steps: first, construct the *contracting endpoints permutation*, $\tau = (a_1, a_m)(a_2, a_{m-1})(a_3, a_{m-2}) \dots (a_i, a_{m-i-1}) \dots$, so named because we start at the extreme ends of the subpath, form a transposition on these letters, and then repeat as we move towards the center of the subpath. Second, compute $q = p^\tau$.

Let R be the set of disjoint subpaths to reverse and let $\tau(i)$ be the contracting endpoints permutation of subpath(i). The R -reversal neighborhood of p are the tours that reverse all subpath combinations in p . For example, $p^{\tau(2)\tau(4)}$ simultaneously reverses subpaths 2 and 4, and in general $p^{\prod_{i \in R} \tau(i)}$ simultaneously reverses the indexed disjoint subpaths.

By construction, the disjoint $\tau(i)$'s commute, and each $\tau(i)$ is an involution. Thus if T is the set of all $\tau(i)$'s, then the subgroup generated by T , denoted $\langle T \rangle$, consists of all their possible products. In turn, the reversal neighborhood of p is $p^{\langle T \rangle}$.

2.2.2 Conjugative Elite Paired Solution Neighborhoods

Suppose the feasible solutions of an n-city m-CTSP must satisfy a specific cycle structure, i.e., feasible permutations are elements of a single

conjugacy class in S_n . Let p and q be elite feasible tours, where $\text{tourlength}(q) < \text{tourlength}(p)$. The conjugative equation $p^x = q$ has solution space $X = [\text{Centralizer}(S_n, p)]r$, where r is any specific solution (note that X is a right coset). This fact suggests two neighborhoods which may have superior solutions to both p and q .

The first is $p^{S(\text{mov}(\alpha))}$, where α is any element of X . Since $q = p^\alpha$, we know the neighborhood has at least one shorter tour than p and perhaps others as well. Note that the α with fewest letters yields the most easily built symmetric group.

For the second neighborhood, q^x , a small $\text{mov}(qp^{-1})$ is preferred. In this case, p and q will be very similar and since $p^x = q$ is a shorter tour than p , q^x may include other attractive tours.

2.2.3 Conjugative Path Relinking

Conjugative moves provide a useful way to represent the path relinking metastrategy used to intensify and diversify a tabu search. Glover, Kelly and Laguna (1995) say each intermediate solution in the path from p to q arises from its predecessor in a way that leaves fewest remaining moves to q . As before, we presume p and q share a common cycle structure.

Each solution in S_n to $q = p^x$ represents a "direct move" from p to q . We break down a direct move in order to build the diversification paths of path relinking. That is, if x has multiple disjoint cyclic factors $\{\lambda(i)\}_{i=1..m}$, then one path is $\{p, p^{\lambda(1)}, p^{\lambda(1)\lambda(2)}, \dots, p^{\lambda(1)\dots\lambda(m-1)}, p^x = q\}$. However, the disjoint $\lambda(i)$ commute and so there are at most $(m-1)!$ such paths. The above remarks by Glover, Kelly and Laguna favor the solution x with fewest cyclic factors.

When x is a k -cycle, then express it as a product of transpositions, $x = \prod_i \tau_i$, where i takes on values from 1 to $k-1$. The associated path $\{p, p^{\tau(1)}, p^{\tau(1)\tau(2)}, \dots, p^{\tau(1)\dots\tau(k-2)}, p^x = q\}$ is strictly determined since the $\tau(i)$ don't commute. However, each of the k equivalent representations of x determines a different path.

Thus, either use the x with fewest cyclic factors or the k -cycle x with fewest letters, comparing $(m-1)!$ and k . Although we have assumed the solution space is a conjugacy class, Colletti (1999) describes template-based path relinking when p and q do not share a common cycle structure.

3. Template Based Neighborhoods

In Section 3.1, we describe a group theoretic process that can transform any permutation in S_n into any other permutation. Specializing the results of Section 3.1, Section 3.2 presents a compact and efficient method to construct neighborhoods that preserve stipulated directed subpaths between cities. A special case of that group theoretic method is then shown to yield a very simple construction of the classical k -Or neighborhood for arbitrary k . Section 3.3 proposes a new

neighborhood which seeks to preserve the common properties of two elite solutions.

3.1 Templates

Templates are elements of S_n . Postmultiplying a permutation, p , with the inverse of a *splitting template* τ will partition p into *fragmentary cycles*. The resulting permutation, $q = pt^{-1}$, will have more cycles than p . Postmultiplying q by a welding template, ω , will recombine 2 or more of q 's cycles to form another permutation, r , with fewer cycles than q . Templates algebraically describe and generalize the classical cross exchange (Taillard et al. 1997) and subtour patching methods (Gilmore, Lawler and Shimoys 1985; Karp 1979; Bartalos, Dudas and Imreh 1995).

For example, consider the 12-cycle, $p = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12) \in S_{12}$, and the splitting template $\tau = (1, 3, 6, 10, 12)$. $q = pt^{-1}$ yields $q = (1, 2)(3, 4, 5)(6, 7, 8, 9)(10, 11)$. This is the same as removing the arcs $\{[2, 3], [5, 6], [9, 10], [11, 12], [12, 1]\}$ from p and then looping each resulting *fragmentary subpath* upon itself. τ is the cycle formed from the p -ordered tails of the subpaths of p that survive the cuts. Alternatively, τ is the cycle formed by the p -ordered heads of arcs deleted in p .

In general, if the $\{\lambda_i\}$ are the disjoint cyclic factors of p , and if τ_i is a splitting template of λ_i , then

$$q = \prod \lambda_i \tau_i^{-1} = \left[\prod \lambda_i \right] \left[\prod \tau_i^{-1} \right] = p \prod \tau_i^{-1} = p \left[\prod \tau_i \right]^{-1}$$

Thus, if τ is the product of the splitting templates, then $q = pt^{-1}$. The above steps use the fact that disjoint permutations commute: the τ_i 's are disjoint and when $k > i$, τ_i and λ_k are also disjoint. We also know that for any pair of group elements, $x^{-1}y^{-1} = (yx)^{-1}$.

A *welding template*, ω , is an m -cycle that unites m disjoint cycles according to the template's letter sequence. Template letters come from distinct cycles, which may include 1-cycles. For example, $\{(1, 2, 3), (4, 5, 6, 7), (8, 9)\}$ are united by $\omega = (1, 4, 8)$ to create $(1, 2, 3)(4, 5, 6, 7)(8, 9)\omega = (1, 2, 3, 4, 5, 6, 7, 8, 9)$. If $\omega = (5, 2, 9)$ then $(1, 2, 3)(4, 5, 6, 7)(8, 9)\omega = (5, 6, 7, 4, 2, 3, 1, 9, 8)$. Note that in the resulting cycle, each factor appears as a subpath with the tail specified in the template.

A *joining template* is a product of disjoint welding templates. Using p , λ_i and τ_i defined earlier, suppose $\{\omega_k\}$ are disjoint welding templates on the fragmentary cycles. If τ is the product of the τ_i 's and if ω is the product of the ω_i 's, then the m -CTSP tour q created by fragmenting p and then uniting specified fragments is $q = pt^{-1}\omega$.

As a special case whose derivation involves splitting and welding templates, consider the classical k -Or neighborhood (Carlton and Barnes 1996). In any m -CTSP tour $p \in S_n$, the k -Or move repositions a k -letter

subpath $[t, \dots, h]$ after letter x to obtain the new tour q . If $\{t, h^p, x^p\}$ are distinct (where y^p denotes the p -image of letter y) and x is not in subpath $[t, \dots, h]$, then $q = pr$ where r is the 3-cycle (t, h^p, x^p) . The k -Or neighborhood is obtained by computing all such q where x satisfies the stated conditions.

For example, consider $p = (1, 2, 3, 4, 5, 6, 7, 8) (9, 10, 11)$ and its subpath $[3, 4, 5, 6]$. Since $r \in R = \{(3, 7, 1), (3, 7, 2), (3, 7, 8), (3, 7, 9), (3, 7, 10), (3, 7, 11)\}$, the full 4-Or neighborhood obtained by repositioning the subpath throughout p is pR :

$$\{(1, 2, 7, 8, 3, 4, 5, 6) (9, 10, 11), (1, 3, 4, 5, 6, 2, 7, 8) (9, 10, 11), (1, 2, 7, 3, 4, 5, 6, 8) (9, 10, 11), (1, 2, 7, 8) (3, 4, 5, 6, 9, 10, 11), (1, 2, 7, 8) (3, 4, 5, 6, 10, 11, 9), (1, 2, 7, 8) (3, 4, 5, 6, 11, 9, 10)\}$$

3.2 Elite Subpaths Neighborhoods

Suppose that an elite set of directed disjoint subpaths, X , have been identified, i.e., these subpaths are considered so favorable that their integrity should be preserved during the metaheuristic search process. As shown below, group theory makes construction of such a neighborhood straightforward, i.e., the neighborhood is simply a *left coset* of a subgroup, $S(T) \leq S_n$.

Suppose $T \subseteq N = \{1, \dots, n\}$, and let $S(T)$ be the symmetric group on the letters in T . It is well-known that $S(T)$ can be generated by any transposition in $S(T)$ and some appropriate $|T|$ -cycle, where $|T|$ denotes cardinality of T . We now tie these concepts to templates and the elite subpath neighborhood construction.

If τ is a splitting template on p and if ω is a joining template on the fragmentary cycles of pt^{-1} , where $mov(\omega) \subseteq mov(\tau)$, then permutation $pt^{-1}\omega$ contains the fragmentary subpaths. For example, if $p = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15) \in S_{15}$ and $\tau = (1, 4, 8, 13)$, then $q = pt^{-1} = (1, 2, 3) (4, 5, 6, 7) (8, 9, 10, 11, 12) (13, 14, 15)$ and $X = \{[1, 2, 3], [4, 5, 6, 7], [8, 9, 10, 11, 12], [13, 14, 15]\}$.

If ω is $(1, 8)$ or $(4, 13, 1)$, then $pt^{-1}\omega$ is respectively $(1, 2, 3, 8, 9, 10, 11, 12) (4, 5, 6, 7) (13, 14, 15)$ and $(4, 5, 6, 7, 13, 14, 15, 1, 2, 3) (8, 9, 10, 11, 12)$. If $\omega = (4, 8) (13, 1)$, then $pt^{-1}\omega = (4, 5, 6, 7, 8, 9, 10, 11, 12) (13, 14, 15, 1, 2, 3)$. Thus, any ω whose letters come from the fragmentary tails, $T = \{1, 4, 8, 13\}$, is a joining template on the cycles of pt^{-1} .

In general, given a specific set of disjoint subpaths X , let T be the fragmentary tails of the subpaths in X and let q be the product of the fragmentary cycles. Then $S(T)$ consists of joining templates (on these fragmentary cycles) which preserve the fragmentary subpaths. In turn, all permutations in S_n which preserve the subpaths in X are given by the left coset $qS(T \cup [N - mov(q)])$. The reason for $N - mov(q)$ is that q may not move all letters in N , e.g., when a splitting template isolates cities by cutting adjacent arcs.

For our current example, the neighborhood of p that preserves subpaths in X is the left coset $qS(T)$:

$$\{ (1, 2, 3)(4, 5, 6, 7)(8, 9, 10, 11, 12)(13, 14, 15), \\ (1, 2, 3)(4, 5, 6, 7)(8, 9, 10, 11, 12, 13, 14, 15), \\ (1, 2, 3)(4, 5, 6, 7, 8, 9, 10, 11, 12)(13, 14, 15), \\ (1, 2, 3)(4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15), \\ (1, 2, 3)(4, 5, 6, 7, 13, 14, 15, 8, 9, 10, 11, 12), \\ (1, 2, 3)(4, 5, 6, 7, 13, 14, 15)(8, 9, 10, 11, 12), \\ (1, 2, 3, 4, 5, 6, 7)(8, 9, 10, 11, 12)(13, 14, 15), \\ (1, 2, 3, 4, 5, 6, 7)(8, 9, 10, 11, 12, 13, 14, 15), \\ (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)(13, 14, 15), \\ (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15), \\ (1, 2, 3, 4, 5, 6, 7, 13, 14, 15, 8, 9, 10, 11, 12), \\ (1, 2, 3, 4, 5, 6, 7, 13, 14, 15)(8, 9, 10, 11, 12), \\ (1, 2, 3, 8, 9, 10, 11, 12, 4, 5, 6, 7)(13, 14, 15), \\ (1, 2, 3, 8, 9, 10, 11, 12, 13, 14, 15, 4, 5, 6, 7), \\ (1, 2, 3, 8, 9, 10, 11, 12)(4, 5, 6, 7)(13, 14, 15), \\ (1, 2, 3, 8, 9, 10, 11, 12, 13, 14, 15)(4, 5, 6, 7), \\ (1, 2, 3, 8, 9, 10, 11, 12)(4, 5, 6, 7, 13, 14, 15), \\ (1, 2, 3, 8, 9, 10, 11, 12, 4, 5, 6, 7, 13, 14, 15), \\ (1, 2, 3, 13, 14, 15, 8, 9, 10, 11, 12, 4, 5, 6, 7), \\ (1, 2, 3, 13, 14, 15, 4, 5, 6, 7)(8, 9, 10, 11, 12), \\ (1, 2, 3, 13, 14, 15, 8, 9, 10, 11, 12)(4, 5, 6, 7), \\ (1, 2, 3, 13, 14, 15)(4, 5, 6, 7)(8, 9, 10, 11, 12), \\ (1, 2, 3, 13, 14, 15, 4, 5, 6, 7, 8, 9, 10, 11, 12), \\ (1, 2, 3, 13, 14, 15)(4, 5, 6, 7, 8, 9, 10, 11, 12) \}$$

In summary, Section 3.2 algebraically characterizes the neighborhood that results when we "build all tours preserving given subpaths." The resulting equations can be combined with those representing other metaheuristic concepts and by so doing, reveal insights difficult to obtain via narrative descriptions of algorithms.

3.3 Template-Based Elite Paired Solution Neighborhoods

Suppose that a metaheuristic search has produced elite m-CTSP tours p and q , i.e., p and q are among the most attractive yet found. In the spirit of the path relinking metastrategy (Glover, Kelly and Laguna 1995), intuition suggests p and q may guide us to other attractive tours. One method is to use cosets to mix the inherent properties of the attractive solutions.

Without loss of generality, let $\text{tourlength}(q) < \text{tourlength}(p)$. It is straightforward to show that $M = \text{mov}(qp^{-1})$ are the tails of the non common arcs between p and q , a result used to efficiently compute m-TSP tour length difference (Colletti and Barnes 1999). Thus $qp^{-1} \in S(M)$ and so $q \in S(M)p$, a right coset which may contain other tours shorter

than p . Since $|S(M)| = |M|!$, this ancillary neighbor may often merit investigation when M is acceptably small.

As fully detailed in Colletti (1999), group theory may be used to describe many other useful neighborhood construction methods that do not necessarily preserve the cycle structure of the incumbent solution.

4. A Hybrid Move

Arc exchange moves combine template and conjugative moves to create 1-TSP neighborhoods. In the k -arc exchange considered here, k arcs are cut from a 1-TSP tour p and replaced by k arcs to recover a new 1-TSP tour q . All subpaths in p that survive the cut will appear unchanged in q . A neighborhood of p on a specific arcs cut set consists of all tours that can be recovered in this way.

Although conceptually straightforward, building such neighborhoods becomes more challenging as k grows modestly, both in programming effort and run time. Using group theory greatly simplifies the description of this neighborhood construction process. The insights provided by group theory, joined with the tools of computational group theory (Schönert et al. 1995), make neighborhood construction straightforward. Although Colletti, Barnes and Dokov (1999) describe this exchange method using a dihedral group action, here we describe it more elegantly using templates and the simpler cyclic group action.

Let the splitting template τ represent the given arc cut set, where τ^{-1} is the product of the fragmentary cycles. Although $S(\text{mov}(\tau))$ are welding templates which preserve the fragmentary subpaths, not all return a 1-TSP tour. The welding templates that do yield a tour are in $\tau^{S''(\text{mov}(\tau))}$, where $S''(\text{mov}(\tau))$ is the symmetric group on any $|\text{mov}(\tau)| - 1$ letters of $\text{mov}(\tau)$. Thus, the k -arc exchange neighborhood of p determined by the splitting template τ is $E_\tau = \tau^{-1}[\tau^{S''(\text{mov}(\tau))}]$.

Now let splitting template λ represent a different k -arc cut. Since λ and τ have the same cycle structure, there is an $r \in S_r$ where $\lambda = \tau^r$. Colletti (1999) shows that the k -arc exchange neighborhood given by λ is

$$E_\lambda = \left[(r^{-1})^{(p^{-1})} r \right] E_\tau^r$$

This states E_λ in terms of the *already known* E_τ . We may obtain any E_λ (given E_τ has been obtained) while avoiding construction of $S''(\text{mov}(\lambda))$. Furthermore, r can be chosen to have the fewest letters of all elements in the right coset that is the solution space of $\lambda = \tau^x$. These useful but non-intuitive results are possible by viewing these metaheuristic concepts from the group theory viewpoint.

We could build all possible k -arc exchanges by taking the union of all splitting template neighborhoods. However, there is a more effective way

that reduces the overlap among neighborhoods. Let $W[\tau] \equiv \tau^{-1} \cdot \tau^{S^0(\text{mov}(\tau))}$ and let T be all nC_k k-cycle splitting templates on p . Let V be a transversal on the orbits of the group action $\langle p \rangle T$, and let R be a transversal on the solution spaces of $\tau^x = v, \forall v \in V$. Thus, p 's full k-arc exchange neighborhood is

$$N_p = p \left(\bigcup_{r \in R} (W^r[\tau]) \right)^{\langle p \rangle}$$

Finally, suppose we seek the full k-arc exchange neighborhood for a new 1-TSP tour q . Since p and q have the same cycle structure, there exists $z \in S_n$ where $p^z = q$ (as before, choose the element (of the right coset solution space) having the fewest letters). Colletti (1999) proves $N_q = N_p^z$. This expresses the full neighborhood for q in terms of the *already computed* full neighborhood for p . Thus, the effort expended to build a full k-arc exchange neighborhood need only be done once during the metaheuristic search. All subsequent neighborhoods can be directly computed via conjugation on the initial tour's neighborhood. Furthermore, if the full neighborhood is built for $p = (1, \dots, n)$, then this canonical neighborhood can be stored for use in all subsequent searches which build k-arc exchange neighborhoods for the n -city 1-TSP. This reduction in effort holds great promise for improved computational efficiency in evaluating such neighborhoods. Initial *very compelling* indications of the fulfillment of this promise are demonstrated by Jones (2002).

5. Example Applications of GTTS

The last two papers referenced in Table 14.1 were outgrowths of the research proposed by Barnes et al. (1999) and are representative of past and current work in employing GTTS to large complex COPs.

Barnes et al. (2002) describe a GTTS approach to the Aerial Fleet Refueling Problem (AFRP). A much more detailed treatment is given by Wiley (2001). The AFRP is concerned with the efficient and effective use of a heterogeneous set of tanker (refueling) aircraft, located at various geographical locations, in the required operations associated with the deployment of a diverse fleet of military aircraft (originating in the continental United States) to a foreign location or *theater* of activity. Typically, the “receiving” aircraft must traverse great distances over large bodies of water and/or over other inhospitable environs where no ground based refueling resources exist. Lacking the ability to complete their flights without refueling, each receiving aircraft must be serviced one or more times during their deployment flights by means of in-flight refueling provided by one of the available tanker aircraft. The receiving aircraft,

aggregated into receiver groups (RGs) that fly together, have stipulated departure and destination bases and each RG's arrival time is bounded by a stated desired earliest and latest time. The excellence of a suggested solution to this very challenging decision making problem is measured relative to a rigorously defined hierarchical multicriteria objective function.

Given a deployment scenario, examples of overview questions that require answers are (1) How many tankers are required to meet the air refueling requirements? (2) How quickly can all the receivers be deployed to their final destinations? (3) How far do the tankers and receiver aircraft have to travel? (4) How much fuel is burned by both tankers and by receiver aircraft?

In order to meaningfully answer upper level questions like these, as well as more probing questions relating to efficient planning operations, a great many detailed operational aspects must be addressed.

The following information is assumed to be given: (1) a known set of tankers and their associated original beddown (starting) bases, (2) a known set of RG's, (3) a known set of RG starting and ending bases and tanker beddown bases, (4) a known set of flight characteristics for each aircraft including flight speed, altitude, take-off weight, fuel capacity and fuel-burn rates, and (5) a known set of tanker specific characteristics including fuel-offload capacity and fuel-offload rates.

For a given deployment, the following decisions compose the solution to the AFRP: (1) the waypoints (WPTs), i.e., the physical locations (latitude, longitude and altitude) and start times where each refueling of all RGs will take place, (2) the tanker(s) that will serve each WPT, and (3) how much fuel the assigned tanker(s) should deliver to a WPT. We further assume that the decision maker has the authority to (a) stipulate the departure times of all RGs and tankers and (b) to require both tankers and RGs to "orbit" at specified locations to satisfy WPT requirements in terms of timing and location.

As discussed in detail in Barnes, Wiley, Moore and Ryer (2002), the AFRP objective function is multicriteria and hierarchical where the hierarchical ordering of the criteria depends on mission priorities. The solution to the AFRP is composed of a complex set of interrelated decisions involving time, space and amounts of fuel. Usually, the effect of changing any individual decision will "ripple" through the AFRP structure forcing multiple associated changes in the solution.

The AFRP solution is constrained by a large number of limiting factors. The safety of the crews and aircraft associated with the deployment is dominant, i.e., no tanker or receiver aircraft should have to divert from its flight path due to lack of fuel. Many other constraints must also be satisfied. For example, a tanker has limited fuel capacity and its crew has flight duration restrictions which affect the crew-tanker ability to travel long distances and to provide fuel. A tanker requires time to fly between any two locations and time to perform midair refueling. Hence,

all tanker WPT assignments must be limited to those where the tanker is physically capable of being present at the specified WPT time.

The AFRP is unique, complicated and extremely difficult to model and solve when viewed in its full practical context. As is discussed in Wiley (2001), the AFRP can be related to other classical combinatorial optimization problems by relaxing specific constraints or by fixing selected decision variables. Perhaps the most closely associated classical problem is a variation of the multi-vehicle, multi-depot Vehicle Routing Problem (VRP).

However, there are several additional considerations present in the AFRP that are not present in VRP. Among these added considerations are that the AFRP "customers" (RGs) are not fixed in space but rather have multiple time dynamic locations. These time and locations and the amount of product to be delivered must all be determined as part of the solution. In addition, all RGs must be supplied with fuel in a timely manner to prevent "untimely" flight diversions. Directly associated with the WPT decisions are the decisions on the takeoff time of each RG and the possibly multiple takeoff times of each tanker.

The primary objective of the research documented in Barnes, Wiley, Moore and Ryer (2002) was to develop methods for producing an "ensemble" of excellent solutions to any instance of the AFRP. To achieve this objective a Group Theoretic Tabu Search (GTTS) approach was created. A solution to the AFRP was represented as an element of S_n and move neighborhoods were also represented by elements of S_n acting under conjugation or multiplication. To address the issue of reusability and portability, the GTTS approach was implemented using the Java programming language. The implementation makes extensive use of Wiley's (2000) Java class library for S_n .

In the S_n solution depiction, each tanker's deployment activity was represented as a single cycle in the associated symmetric group permutation. Letter or "node" types were associated with tankers, WPTs and airbases in such a way that 8 *specifically tailored* dynamic neighborhoods could be used to search the complex solution space. The construction of such dynamic neighborhoods, which were *essential* to the efficient and effective solution of the AFRP, would not have been practical without the group theoretic perspective that was employed.

Using this GTTS-AFRP methodology, a number of deployment scenarios were attacked. The computational results show that the GTTS-AFRP is effective, providing good results with no parameter tuning. The solutions that were obtained were superior to all known benchmark problem solutions and the procedures were sufficiently robust to allow diverse objective functions and constraints to be considered with no additional difficulty. In particular, a practical sized (typical) Middle East deployment scenario was attacked. This problem consisted of 99 aircraft within 26 RGs originating at bases within the Continental US utilizing 6 tanker origination/beddown bases with 120 available tankers.

The best solution, found in two hours and 16 minutes (on a AMD Athlon 950 Mhz machine with 256 MB RAM), employed 95 tankers flying 326,968 miles and allowed the latest RG to arrive in 55 hours. The tanker assignments in this solution were obtained at the *detailed asset operational level* (individual tankers or receivers) and yielded excellent assignments in terms of such metrics as minimizing tanker usage, tanker travel and total deployment time. In general, depending on the desire of the user, the GTTS-AFRP method can provide either a single "best" solution to a particular problem or a set of robust, comparable solutions to a problem instance. The fact that the best previous method available to the US Air Force would require several analysts weeks, or months, to achieve a single feasible solution strongly underscores the power of the GTTS-AFRP. Thus, with this new tool, analysts and decision makers not only can have the flexibility and added insight provided by multiple solutions, they also can make critical decisions in a tremendously shortened planning horizon.

The last paper referenced in Table 14.1 (Crino et al. 2002) presents a GTTS approach to the Theater Distribution Vehicle Routing and Scheduling Problem (TDVRSP). The TDVRSP is associated with determining superior allocations of required flows of personnel and materiel within a defined geographic area of military operation. A theater distribution system is comprised of facilities, installations, methods and procedures designed to receive, store, maintain, distribute and control the flow of materiel between exogenous inflows to that system and distribution to end user activities and units within the theater. An automated system that can integrate multi-modal transportation assets to improve logistics support at all levels has been characterized as a major priority and immediate need for the US military services.

Crino et al. (2002) describes both the conceptual context, based in a flexible group theoretic tabu search (GTTS) framework, and the software implementation of a robust, efficient and effective generalized theater distribution methodology. The concepts and results are presented in much greater detail in Crino (2002). This unprecedented methodology, for the first time, evaluates and determines the routing and scheduling of multi-modal theater transportation assets at the *individual asset operational level* to provide economically efficient time definite delivery of cargo to customers.

Modeling requirements for the TDVRSP extend well beyond those of the typical VRP. Within a typical theater, there are multi-modal vehicles with differing characteristics such as cruising speed, capacity, cruising length, fixed and variable costs, route type (air, ground, water) and service times. Vehicle modeling requirements include the ability to make one or more trips during the planning horizon, the ability to perform direct delivery from outside the theater, and the ability to refuel at customer locations. Scheduling considerations include vehicle

availability, service times, load times and unload times. All vehicles operate from a depot or a hub.

The theater distribution network nodes are depots, hubs and customers. Depots, aerial ports of debarkation (APODs) and seaports of debarkation (SPODs) are the supply nodes. The hubs, or support areas (SAs) which occur at corps, division and brigade levels, are transshipment nodes, which receive, store and distribute cargo. Customers are sink nodes that receive cargo. All nodes have time window constraints, fuel storage capacity and maximum on the ground (MOG) constraints. Hubs have cargo storage constraints and customers have cargo demand requirements and time definite delivery requirements.

A TDVRSP has three types of time window constraints: early time definite delivery (ETDD), time definite delivery (TDD) and multiple time windows for non-departure and non-arrival times (MTW). An ETDD stringently defines a customer service starting time but does not constrain vehicle arrival or departure times. A TDD defines when a customer service should be complete but does not constrain service occurrence, or vehicle arrival and departure times. MTWs restrict vehicle arrival and departure at a node but do not stipulate when vehicles are loaded or offloaded.

The TDVRSP has a tiered distribution architecture. The first order tier contains the depots and customers/hubs served by the depots. Middle tiers consist of hubs that service customers/hubs. The last order tier consists of end customers served by a hub. Each tier is a self-contained distribution network and lower ordered tiers are dependent on higher ordered tiers.

Hubs distribute cargo after it is received and processed. Cargo, characterized by the amount delivered and time of delivery, is processed and prepared for delivery to its next customer. Cargo is either loaded directly onto available vehicles or stored for later delivery.

The TDVRSP primary objectives are to minimize unmet customer demand (demand shortfall), late deliveries (TDD shortfall), vehicle fixed costs and vehicle variable costs. Late delivery times are weighted by the amount of demand delivered late. A fixed cost is charged for each vehicle used in the solution and variable vehicle costs are associated with vehicle travel.

Applying GTTS to the TDVRSP requires that solutions be represented as elements of S_n . For a TDVRSP solution, each cycle in the solution's disjoint cyclic structure represents a vehicle trip where the first letter in the cycle represents a vehicle. Subsequent letters represent the customers serviced by that vehicle. A unit cycle represents either a vehicle letter not leaving the depot/hub or a customer letter not serviced. Since vehicles can make multiple trips and customers can receive multiple services within a time period, a unique letter is allocated for each vehicle trip and each customer service.

In addition to yielding an efficient and effective solution methodology for the TDVRSP, the research documented in Crino et al. (2002) also provided *significant* advances in the development of GTTS applied to vehicle routing and scheduling problems. These GTTS advances, described in detail in Crino (2002), included, for the *first* time, the formulation of move neighborhoods defined using groups to generate orbits. These orbits were used as a means to efficiently search the solution space. They eliminated cycling, prevented solution reevaluation and avoided entrapment in local optima. This methodology prevents the search from being trapped in a subset of the solution space topology and eliminates the need for a restart mechanism. This technique allowed exhaustive non-repetitive search of each partition and, by placing a utilized orbit on an orbit tabu list, prevented reevaluation.

A unique solution space partition hierarchy was developed using the symmetric group on n -letters. Conjugacy classes, cyclic form structures, orbital planes and orbits were defined that partition the solution space. Solution space partitions were exploited in the diversification and intensification process. In addition, neighborhoods were constructed to intelligently traverse the partitions and enable a potential search of the entire space. Group move neighborhoods steered the search between different orbits. And swap move neighborhoods traversed the search between different orbital planes. Insert and extraction move neighborhoods moved the search to different conjugacy classes and cyclic form structures.

Orbital planes were defined and used as a primary search mechanism of the GTTS. Orbital planes are orbits partitioned by orbits. They provide a more atomic partitioning of the solution space permitting partial or exhaustive search. Using orbital planes is a highly efficient special case of Colletti's (1999) orbital transversal method because the search may re-examine an orbit (orbital plane) without reevaluating solutions within the orbit (orbital plane).

Details on the TDVRSP modeling assumptions and algorithmic solution procedures are discussed in detail in Crino (2002) and Crino et al. (2002). Prior to Crino's (2002) research, no benchmark problems existed for the TDVRSP. Crino (2002) used valid experimental design methodology to construct a set of 39 problems that effectively test the robustness of the GTTS-TDVRSP algorithm. The problems are composed of three TDVRSP types: the Air Force multiple trips multiple services (MTMS) without hub, the Joint MTMS without hub, and the Joint MTMS with hub and other defining constraints. The 39 problems were further categorized by problem size (small, medium and large), delivery restriction density (low, medium and high), demand to capacity ratio (low, medium and high).

Results from Crino's benchmark problem 37, one of the *large* Joint MTMS problems with hub and other defining constraints is indicative of the timely, effective and robust results provided by the GTTS-TDVRSP

approach for all 39 benchmark problems. The total run time was 110 minutes, where the best solution was found at iteration 709 (62.8 minutes). However, competitive satisfactory solutions, where all customer demands were satisfied, were found much earlier in the search process. The GTTS-TDVRSP method has clearly displayed its ability to solve the multiple objective TDVRSP problem.

The development of an automated solution methodology to the TDVRSP problem has been characterized as a major priority and immediate need for the US military services. Many software programs are available that “perform” theater distribution modeling. However, all, but the technique presented in Crino et al. (2002), are simulation models and simulations can provide neither total asset visibility (TAV) nor in-transit visibility (ITV). Therefore, this model is the *first* of its kind to offer this functionality.

In a recent study funded by HQ USAF (HQ USAF/ILA, 2002), a number of recommendations were made to the Air Force on types of models that could support an automated theater distribution management system. The purpose of the system is to “optimize” the entire theater distribution system, from the number of bases and vehicles, down to the vehicle routes and schedules. They concluded that vehicle routing and scheduling was very difficult to optimize, and development of an optimization approach was considered a high-risk methodology. Therefore, they proposed a low risk method using simulation. Unfortunately, they lose the functional requirements of providing TAV and ITV when using simulation. The HQ USAF study validates the importance and magnitude of this research. What was regarded as *too difficult* has been successfully *created, developed and demonstrated* in this TDVRSP research.

6. Concluding Remarks

Group theory provides a unifying mathematical framework in which to study metaheuristic neighborhoods and search methods applied to P|O problems. This algebra allows us to build and solve equations whose coefficients and variables are permutations and sets of permutations. By manipulating these equations using the comprehensive structure provided by group theory, we can obtain significant understanding of important concepts that are more difficult or impossible to glean using more customary approaches.

Of equal or greater importance, the understanding and structure provided by the group theoretic perspective can also lead directly to more powerful solution methodology applied to complex practical COPs.

We are continuing our ongoing investigations into the structure and character of several classes of metaheuristic search neighborhoods. We hope that the first steps described in this paper will motivate other researchers to consider how the unified framework of group theory could

be further exploited to gain powerful new insights into move neighborhoods and other metaheuristic concepts.

We are also continuing to develop approaches to provide solutions to other complex military logistics problems. As documented in Barnes, McKinzie and Gomes (2002) and in the recently funded Phase II continuation proposal (Barnes et al. 2002) of the efforts proposed by Barnes et al. (1999), five additional military logistics problems are being approached using GTTS methodology.

We hope that future researchers will join us in this exciting new area of theoretical and applied work.

Acknowledgements

This research has been supported by three grants from the Air Force Office of Scientific Research.

Appendix

The pure mathematics of group theory (Gaglione 1992, Isaacs 1994) has many practical uses. A group is any set and operation which together satisfy the properties of closure, associativity, right-identity and right-inverse (g^{-1} is the inverse of group element g). For example, the set of all n-by-n invertible real matrices is a group: multiplying any two such matrices produces another matrix (*closure*); multiplication is *associative*; right-multiplying a matrix by the *identity* matrix produces the given matrix; and right-multiplying a matrix by its *inverse* produces the identity matrix. Another familiar group is the integers under addition.

Group theory may be directly applied to the m-CTSP by using S_n , whose elements make up all possible partitions and orderings of the n cities. For example, one group element, or *permutation*, of S_{11} is the partition of the 11 cities onto four agents, $p = (2, 3, 7) (1, 6, 5, 4) (9, 8, 11) (10) \equiv (2, 3, 7) (1, 6, 5, 4) (9, 8, 11)$, where, by convention, unit cycles are not explicitly written. p has four subtours or *cycles* in which each letter is mapped into its successor (denoted $2^p = p(2) = 3$, $3^p = p(3) = 7$, $7^p = p(7) = 2$, $10^p = p(10) = 10$). $\text{mov}(p)$ are the letters contained in the disjoint non-unit cycles of p , i.e., for $p = (2, 3, 7) (1, 6, 5, 4) (9, 8, 11) (10)$, $\text{mov}(p) = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 11\}$. If q is also a permutation then the product pq is the composite function produced by applying p and then q , i.e., for letter x , $x^{pq} = (x^p)^q$. For example, if $q = (3, 7, 8, 10) (4, 9)$ then $3^{pq} = (3^p)^q = 7^q = 8$, and so under pq , 3 is mapped into 8. Permutation multiplication need not be commutative since $3^p = (3^q)^p = 7^p = 2$, and so $pq \neq qp$. Since a permutation represents a 1-1 mapping from the letters $\{1, \dots, n\}$ onto itself, multiplication (i.e., function composition) is associative and closed.

The inverse of a permutation simply reverses each cycle, and the identity permutation is that which maps each letter into itself, i.e., is composed of n unit cycles. Thus, the four properties of a group are

satisfied and the set of all $n!$ permutations under multiplication is called the symmetric group on n -letters, denoted S_n .

The components of a permutation are cycles and an m -cycle has m letters. A 2-cycle is also called a *transposition* and an n -cycle is a permutation with a single cycle equivalent to a 1-TSP tour. Every permutation is a unique product of *disjoint* cycles, i.e., cycles sharing no common cities. The number of cycles and the cycle sizes define the permutation's *cycle structure* and every cycle is a non-unique product of transpositions. For example, $(1, 2, 3, 4) = (2, 3, 4, 1) = (1, 2)(1, 3)(1, 4)$ while $(2, 3, 4, 1) = (2, 3)(2, 4)(2, 1)$. If $p^2 = pp$ is the identity, then p is an *involution*.

Here are other key concepts, where G is an abstract group:

If group $H \subseteq G$ then H is a *subgroup* of G , denoted $H \leq G$

If $H \leq G$ and $g \in G$, then $Hg = \{hg: h \in H\}$ is a *right coset* of H in G , and $gH = \{gh: h \in H\}$ is a *left coset* in G

If $H \subseteq G$ and $g \in G$, then $g^H = \{g^h: h \in H\}$

All left (right) cosets of H exclusively and exhaustively partition G

If $H \leq G$ and $g \in G$, the *Centralizer* of g in H is the set of all elements in H that commute with g , i.e., $\text{Cent}(H, g) = \{h \in H: h^{-1}gh = g\} = \{h \in H: gh = hg\}$

A *transversal* on disjoint sets is a collection of elements, precisely one from each set

The last overview item is the *group action* $_G T$, a concept that uses a group G to partition a set T into mutually exclusive and exhaustive cells called *orbits*. Regarding group elements as "moves" between elements of T , the elements in the partition of $x \in T$ are those reachable from x by any series of moves. If $g \in G$, then x^g denotes the element in T reached in one step from x via g .

The group action *operator* is the rule that assigns value to x^g , e.g., conjugation if $T \subseteq G$, the mapping operator if T are letters and G is a permutation group, or similarity products if T and G are n by n matrices. Finally, a group action must satisfy certain properties in order to be valid (Isaacs 1994), i.e., one cannot freely match any group G with any set T . Colletti (1999) uses group actions to escape the chaotic attractors of reactive tabu search (Battiti and Tecchiolli 1994).

References

- Barnes, J.W., B. Colletti, M. Fricano, R. Brigantic, J. Andrew and T. Bailey (1999) "Advanced Air Mobility Command Operational Airlift Analyses Using Group Theoretic Metaheuristics," A Consortium Proposal," submitted to the Air Force Office of Scientific Research (Funded February 2000).

- Barnes, J.W., V. Wiley, J. Moore and D. Ryer (2002) "Solving the Aerial Fleet Refueling Problem using Group Theoretic Tabu Search," in review, invited paper for a special issue of *Mathematical and Computer Modeling* on Defense Transportation Issues.
- Barnes, J.W., K. McKinzie and C. Gomes (2002) "Achieving Quicker and Better Solutions to USTRANSCOM's Problems with Group Theoretic Tabu Search," A Collaboration between Cornell University and The University of Texas at Austin, submitted to the Air Force Office of Scientific Research (Funded September 2002).
- Barnes, J.W., M. Shirley, T. Irish, J. Andrew and J. Moore (2002) "Advanced Air Mobility Command Operational Airlift Analyses Using Group Theoretic Metaheuristics, Phase II, A Consortium Proposal," submitted to the Air Force Office of Scientific Research, (Funded January 2003)
- Bartalos, I., T. Dudas and B. Imreh (1995) "On a Tour Construction Heuristic for the Asymmetric TSP," *Acta Cybernetica* 12: 209–216.
- Battiti, R. and G. Tecchiolli (1994) "The Reactive Tabu Search," *ORSA Journal on Computing* 6(2):126–140.
- Carlton, W.B. and J.W Barnes (1996) "Solving the Traveling Salesman Problem with Time Windows using Tabu Search," *IIE Transactions* 28(8):617–629.
- Codenotti, B. and L. Margara (1992) Local Properties of Some NP-Complete Problems. TR-92-021, International Computer Science Institute, University of California at Berkeley.
- Colletti, B.W. (1999) *Group Theory and Metaheuristics*. Ph.D. dissertation, The University of Texas at Austin.
- Colletti, B. and J.W. Barnes (1999) "Group Theory and Metaheuristic Neighborhoods," (Graduate Program in Operations Research, Technical Report Series 99-02, The University of Texas at Austin).
- Colletti, B. and J.W. Barnes (2000) "Linearity in the Traveling Salesman Problem," *Applied Mathematics Letters* 13(3):27–32.
- Colletti, B. and J.W. Barnes (2001) "Local Search Structure in the Symmetric Traveling Salesperson Problem under a General Class of Rearrangement Neighborhoods," *Applied Mathematics Letters*, 14(1): 105–108.
- Colletti, B., J.W. Barnes and D.L. Neuway (2000) "Using Group Theory to Study Transition Matrices of Metaheuristic Neighborhoods," Graduate Program in Operations Research, Technical Report Series 2000-03, The University of Texas at Austin, in review, European Journal of Operational Research.
- Colletti, B., J.W. Barnes and S. Dokov (1999) "A Note on Characterizing the k-OPT Neighborhood Via Group Theory," *Journal of Heuristics* 5: 51–55.
- Crino, J.R. (2002) *A Group Theoretic Tabu Search Methodology For Solving Theater Distribution Vehicle Routing and Scheduling Problems*. Ph.D. dissertation, AFIT.
- Crino, J., J. Moore, J.W. Barnes and W. Nanny "Solving The Military Theater Distribution Vehicle Routing and Scheduling Problem Using Group Theoretic Tabu Search", in review, invited paper for a special issue of *Mathematical and Computer Modeling* on Defense Transportation Issues.
- Gaglione, A.M. (1992) *An Introduction to Group Theory* (PU/NRL/5350-92-231). Naval Research Laboratory, Washington D.C (also United States Naval Academy, Department of Mathematics, Annapolis MD).

- Gilmore, P.C., E.L. Lawler and D.B. Shmoys. (1985) "Well-Solved Special Cases." In E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan and D.B. Shmoys (eds.), *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. New York: John Wiley & Sons.
- Glover, F. (1969) "Integer Programming Over a Finite Additive Group," *SIAM Journal on Control*, 7(2):213–231.
- Glover, F., J.P. Kelly and M. Laguna (1995) "Genetic Algorithms and Tabu Search: Hybrids for Optimization," *Computers and Operations Research*, 22(1):111–134.
- Gomory, R.E. (1958) "Outline of an Algorithm for Integer Solutions to Linear Programs," *Bulletin of the American Mathematical Society*, 64: 275–278.
- Gomory, R.E. (1965) "On the Relation Between Integer and Noninteger Solutions to Linear Programs," *Proceedings of the National Academy of Sciences, USA*, 53:260–265.
- Gomory, R.E. (1969) "Some Polyhedra Related to Combinatorial Problems," *Linear Algebra and its Applications*, 2(4):451–558.
- Grover, L.K. (1992) "Local search and the Local Structure of NP-Complete Problems," *Operations Research Letters*, 12((4)):235–243.
- HQ USAF/ILA. "Theater Distribution Management System Requirements and Feasibility Study", 15 January, 2002.
- Isaacs, I. (1994) *Algebra: A Graduate Course*. Pacific Grove: Brooks/Cole Publishing.
- Jones, K. (2002) "A Highly Effective and Efficient Method for Neighborhood Evaluation In Metaheuristic Search," MS Report, The University of Texas at Austin.
- Karp, R.M. (1979) "A Patching Algorithm for the Nonsymmetric Traveling-Salesman Problem," *SIAM Journal of Computing*, 8(4):561–573.
- Lin, S. and B. Kernighan (1973) "An Effective Heuristic Algorithm for the Traveling Salesman Problem," *Operations Research*, 21:498–516.
- Salkin, H.M. (1975) *Integer Programming*. Reading: Addison-Wesley Publishing Company.
- Schönert, Martin, I.M. R. Araújo, H.U. Arthur, T. Besche, Bischofs, O. Bonten, T. Breuer, F. Celler, G. Cooperman, B. Eick, V. Felsch, F. Gähler, G. Gamble, W. de Graaf, B. Höfling, J. Hollmann, D.F. Holt, E. Horvath, A. Hulpke, A. Kaup, S. Keitemeier, S. Linton, F. Lübeck, B. Majewski, J. Meier, T. Merkowitz, W. Merkowitz, J. Mnich, R.F. Morse, S. Murray, M. Neunhöffer, W. Nickel, A. Niemeyer, D. Pasechnik, G. Pfeiffer, U. Polis, F. Rákóczi, S. Rees, U. Schiffer, A. Seress, A. Solomon, H. Theissen, R. Wainwright, A. Wegner, C. Wensley (1995) GAP—Groups, Algorithms and Programming. Lehrstuhl D für Mathematik, Rheinisch Westfälische Technische Hochschule, Aachen, Germany, 5th edition. www.gap-system.org
- Taillard, E., P. Badeau, M. Gendreau, F. Guertin and J. Potvin (1997) "A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows," *Transportation Science*, 31:170–186.
- White, W.W. (1966) *On a Group Theoretic Approach to Linear Integer Programming*. Ph.D. dissertation, University of California at Berkeley.
- Wiley, V. The Symmetric Group Class User's Manual for Partitioning and Ordering Problems, Technical Report Series ORP00-04, The University of Texas at Austin, Graduate Program in Operations Research.

- Wiley, V. (2001) *The Aerial Fleet Refueling Problem*. Ph.D. Dissertation, The University of Texas at Austin.
- Wolsey, L.A. (1969) *Mixed Integer Programming: Discretization and the Group Theoretic Approach*. Ph.D. dissertation, Massachusetts Institute of Technology.

Chapter 15

LOGISTICS MANAGEMENT

An Opportunity for Metaheuristics

Helena R. Lourenço

GREL, DEE, Universitat Pompeu Fabra, Barcelona, Spain, helena.ramalhinho@upf.edu

Abstract: In today's highly competitive global marketplace, the pressure on organizations to find new ways to create value and deliver it to their customers grows ever stronger. In the last two decades, the logistics function has moved to center stage. There has been a growing recognition that effective logistics management throughout the firm and supply chain can greatly assist in the goal of cost reduction and service enhancement. The keys to success in Logistics Management (LM) require heavy emphasis on integration of activities, cooperation, coordination and information sharing throughout the firm and the entire supply chain, from suppliers to customers. To be able to respond to the challenge of integration, modern businesses need sophisticated decision support systems based on powerful mathematical models and solution techniques, together with advances in information and communication technologies. Both industry and academia alike have become increasingly interested in using LM as a means of responding to the problems and issues posed by changes in the logistics function. This paper presents a brief discussion on the important issues in LM and argues that metaheuristics can play an important role in solving complex logistics problems derived from designing and managing logistics activities within the supply chain as a single entity. Among several possible metaheuristic approaches, we will focus particularly on Iterated Local Search, Tabu Search and Scatter Search as the methods with the greatest potential for solving LM related problems. We also briefly present some successful applications of these methods.

Keywords: Logistics Management, Metaheuristics, Iterated Local Search, Tabu Search and Scatter Search

1. Introduction

In today's highly competitive global marketplace, the pressure on organizations to find new ways to create value and deliver it to their customers grows ever stronger. The increasing need for industry to compete with its products in a global market, across cost, quality and service dimensions, has given rise to the need to develop logistic systems that are more efficient than those traditionally employed. Therefore, in the last two decades, logistics has moved from an operational function to the corporate function level. There has been a growing recognition that effective logistics management throughout the firm and supply chain can greatly assist in the goal of cost reduction and service enhancement.

The key to success in Logistics Management (LM) requires heavy emphasis on integration of activities, cooperation, coordination and information sharing throughout the entire supply chain, from suppliers to customers. To be able to respond to the challenge of integration, modern businesses need sophisticated decision support systems (DSS) based on powerful mathematical models and solution techniques, together with advances in information and communication technologies. There is no doubt that quantitative models and computer based tools for decision making have a major role to play in today's business environment. This is especially true in the rapidly growing area of logistics management. These computer-based logistics systems can make a significant impact on the decision process in organizations. That is why both industry and academia alike have become increasingly interested in using LM and logistics DSS as a means of responding to the problems and issues posed by changes in the area.

Many well-known algorithmic advances in optimization have been made, but it turns out that most have not had the expected impact on decisions for designing and optimizing logistics problems. For example, some optimization techniques are of little help in solving complex real logistics problems in the short time needed to make decisions. Also, some techniques are highly problem-dependent and need high expertise. This leads to difficulties in the implementation of the decision support systems which contradicts the trend towards fast implementation in a rapidly changing world. In fact, some of the most popular commercial packages use heuristic methods or rules of thumb. The area of heuristic techniques has been the object of intensive studies in the last few decades, with new and powerful techniques, including many metaheuristic methods, being proposed to solve difficult problems. There is therefore, on the one hand, the need for sophisticated logistics DSS to enable organizations to respond quickly to new issues and problems faced in LM, and, on the other, there are advances in the area of metaheuristics that can provide an effective response to complex problems. This provides a fertile

ground for the application of these techniques in LM and, subsequently, the development of computer-based systems to help logistics decisions.

The objective of this paper is to provide an understanding of the role that metaheuristics can play in solving complex logistics problem in an integrated business processes environment such as optimizing routing distribution, supply chain design, production scheduling and resource allocation.

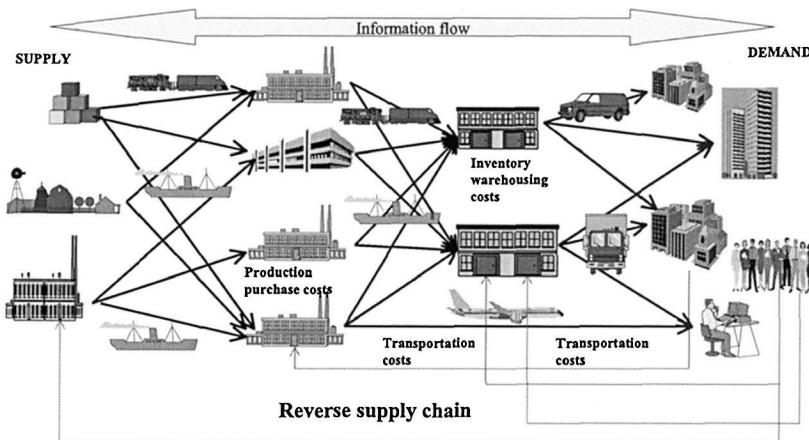


Figure 15.1. An example of a supply chain

In the following section we present a brief discussion on the important issues in LM. Next, we argue that metaheuristics can play an important role in solving complex logistics problems derived from the important need to design and manage the entire supply chain as a single entity. Among several possible metaheuristic approaches, we will focus particularly on Iterated Local Search, Tabu Search and Scatter Search as the methods with the greatest potential for solving LM related problems. In Section 4 we will give a brief presentation of some successful applications of metaheuristics in the solving of real supply chain problems; finally, we present some conclusions and directions for future research.

2. Logistics Management

The Council of Logistics Management defines Logistics as follows: "Logistics is part of the supply chain process that plans, implements, and controls the efficient, effective flow and storage of goods, services, and related information from the point of origin to the point of consumption in order to meet customers' requirements" (<http://www.clim1.org>). However, there is no clear consensus in literature on the definition of LM. Many authors refer to Logistics Management as Supply Chain Management (SCM), i.e. they considered that LM is logistics taken across inter-organizational boundaries; and use these terms interchangeably. Simchi-Levi, Kaminski and Simchi-Levi (2000) gave the following definition: "Supply Chain Management is a set of approaches utilized to efficiently integrated suppliers, manufactures, warehouses, and stores, so that merchandise is produced and distributed at the right quantities, to the right locations, and at the right time, in order to minimize system wide costs while satisfying service level requirements.

Johnson et al. (1999) also presented the following definitions. They maintained that "*Logistics* define the entire process of materials and products moving into, through, and out of a firm. *Inbound logistics* covers the movement of materials received by the suppliers. *Material management* describes the movements of materials and components within a firm. *Physical distribution* refers to the movement of goods outwards from the end of the assembly line to the customer. Finally, *supply-chain management* is a somewhat larger concept than logistics, because it deals with managing both the flow of materials and the relationships among channel intermediaries from the point of origin of raw materials through to the final consumer."

Recently, however, there has been some convergence towards accepting SCM as a larger concept than logistics management. Cooper, Lambert and Pagh (1997) clearly described the differences between the two concepts. They claimed that the integration of business processes across the supply chain is what they called Supply Chain Management, therefore, SCM covers more functions than just logistics being integrated across firms. One of the key components of SCM is, of course, Logistics Management, but it also includes Customer Relationship Management and Product Development and Commercialization.

In this paper, following the above definitions, we define LM as the management of all logistics activities throughout a firm and supply chain. We give special emphasis to relationships with other functions of the organization, such as marketing and finance, and to the integration of the logistics activities in the entire supply chain, including those with suppliers and customers. We consider, like Cooper, Lambert and Pagh (1997), that,

SCM covers a wider area than LM, but that LM is also of major importance to efficient SCM. In LM, the planning, coordinating and controlling of all logistics activities must be done by taking into account the remaining elements of the supply chain. Every firm, whether involved in manufacturing or services, belongs to at least one supply chain. The key success of LM, may lie in the system's integration, i.e. requiring emphasis on integration of logistics activities, cooperation, coordination, and information sharing throughout the entire supply chain.

The supply chain encompasses all activities associated with the flow and transformation of goods from raw material stages to the end users, as well as the associated information flows. Material and information both flow up and down the supply chain. A supply chain consists, basically, of the following elements: suppliers, manufacturing centers, warehouses, distribution centers, transportation systems, retail outlets and customers; raw material, work-in process inventory, finished goods and information that flows between the different elements (see Figure 15.1). One important aspect in a supply chain is the integration and coordination of all logistics activities in the chain, since decisions in one element directly affect the whole supply chain. Firms must avoid sub-optimization by managing the logistics activities on the entire supply chain as a single entity. This integration aspect obviously significantly increments the complexity of any logistics problem. To respond to this challenge there is the need for powerful and robust techniques, as we will discuss in the following section.

We will consider the following key issues in LM:

- Logistics integration.
- Facility location and network design
- Transportation and vehicle routing
- Material handling and order picking
- Customer service
- Product design
- Logistics of production and operations
- Warehouse management and distribution strategies.
- Inventory management.
- Information systems and DSS
- E-commerce and e-logistics
- Reverse and green logistics

These areas interact to a large degree with other functions of the firm, and with other elements of the supply chain. They can therefore benefit a great deal from efficient management based on information and optimization systems. For each issue, we offer a brief description and discuss aspects that can increase the complexity when optimizing the logistics activities within a firm or the entire supply chain. The idea is not to discuss the issues in detail,

interested readers who are referred to Simchi-Levi, Kaminsky and Simchi-Levi (2000), Ballou (1998), Johnson et al. (1999). We also refer to Tayur, Ganeshan and Magazine (1998) where several quantitative models for SCM are presented and a broad taxonomy review research is described.

2.1 Logistics Integration and Coordination

Logistics coordination and integration within a supply chain has become a core issue in LM, not just integration within the organization but integration upstream with suppliers and downstream with distributors and customers. Coordination and integration means many different things, but basically all authors agree that it refers to collaborative working and implies joint planning, joint product development, mutual exchange of information and integrated information systems, cross coordination on several levels in the companies on the network, long term cooperation, fair sharing of risks and benefits, etc., Skoett-Larsen (2000). One enormous advantage of an integrated supply chain is the reduction of the so-called bullwhip-effect, Lee, Padmanabhan and Whang (1997), where small changes or decisions, on one level of the network, may result in large fluctuations, large amounts of stock, and/or increased lead times on other levels of the supply chain. However, as the process becomes more integrated within a supply chain, the complexity of the logistics decisions also increases.

There are two main aspects involved in the integration of logistics decisions. The first of these are the information systems. Without integration of information systems between the different players, there can be no translation or sharing of information, which is the basis for any possible integration between departments or firms. With today's technology, the integration of information systems is possible and has been implemented by many firms. The second aspect is the use of optimization systems to achieve an integrated management of the logistics activities. As more and more industries decide to integrate their information systems, the need for sophisticated tools to help the decision makers to evaluate possible alternatives, decisions and their impact in the whole supply chain also increases.

2.2 Facility Location and Network Design

The firm must balance the costs of opening new warehouses with the advantages of being close to the customer. Warehouse location decisions are crucial determinants of whether the supply chain is an efficient channel for the distribution of the products.

In OR literature, there are several research projects dedicated to location issues, such as warehouse location. See, for example, the web page of the European Working Group on Locational Analysis - EWGLA (<http://www.vub.ac.be/EWGLA/homepage.htm>) and the one for the Section on Location Analysis within INFORMS-SOLA (<http://www.uscolo.edu/sola/sola.html>), as well as the following references Miller (1996), Drezner (1995) and Daskin (1995). It seems that some of these models are quite simple when representing real problems in the design of an actual supply chain. For example, most of them do not take into account warehouse capacity, warehouse handling and operational costs (most of them just take into account the initial fixed cost of the warehouse) or warehouse service level requirements, which can be connected to inventory issues. Also, when designing a supply chain that involves several countries, import and export taxes, different transportation options, cultural and legal issues and several others must be taken into consideration. Another important aspect is the relationship between network design and demand management. Aspects such as the seasonal nature of demand has never been taken into account, as far as we know. However, it could be an interesting research area since many firms are interested in designing their supply networks in partnership with other firms that have products with completely different seasonal behavior, e.g. air conditioning and heating equipment. The incorporation of all the aspects mentioned above into a location or network design model can make a significant difference to the analysis of the logistics on a supply chain and the decisions with respect to location and supply chain design.

2.3 Transportation and Vehicle Routing

One of the central problems of supply chain management is the coordination of product and material flows between locations. A typical problem involves bringing products located at a central facility to geographically dispersed facilities at minimum cost. For example, the product supply is located at a plant, warehouse, cross-docking facility or distribution center and must be distributed to customers or retailers. The task is often performed by a fleet of vehicles under the direct control, or not, of the firm. Transportation is an area that absorbs a significant amount of the cost in most firms. Therefore, methods for dealing with the important issues in transportation, such as mode selection, carrier routing, vehicle scheduling and shipment consolidations are needed in most companies.

One important aspect in transportation management is coordination with the remaining activities in the firm, especially within warehouse and customer service. In some cases transport is the last contact with the customer and companies should therefore take care to meet the customer expectations

and use this relationship to improve their sales. The transport coordination within the different elements of a supply chain, involving different companies, can be of great strategic importance, since all of them most likely benefit by offering fast delivery to a specific customer. Therefore, many issues in the integration of transportation with other activities in the network can be a challenge to academic and industrial communities.

One basic and well-known problem in transportation is vehicle scheduling and routing. A vehicle scheduling system should output a set of instructions telling drivers what to deliver, when and where. An “efficient” solution is one that enables goods to be delivered when and where required, at the lowest possible cost, subject to legal and political constraints. The legal constraints relate to working hours, speed limits, regulations governing vehicle construction and use, restrictions for unloading and so on. With the growth in Internet sales, this problem is gaining momentum, since delivery times are usually very short, customers can be dispersed in a region, every day there is a different set of customers and with very short product delivery time-windows. For a review on the area see Crainic and Laporte (1998).

2.4 Warehouse Management and Distribution Strategies

Warehousing is an integral part of every logistics system and plays a vital role in providing a desired level of customer service. Warehousing can be defined as the part of a supply chain that stores products (raw materials, parts, work-in-process and finished goods) at and between points of production and points of consumption, and also provides information to management on the status and disposition of items being stored. The basic operations at a warehouse are receiving, storage-handling, order picking, consolidation – sorting and shipping. The main objectives are to minimize product handling and movement and store operations as well as maximize the flexibility of operations. Given the actual importance of the activities related to order picking we dedicate a subsection to it.

Traditional warehouses are undergoing enormous transformations due to the introduction of direct shipment and cross-docking strategies. The latter may be more effective in distributing the products among retailers or customers. However, in order to be successful, these strategies require a high level of coordination and information systems integration between all elements in the supply chain: manufacturers, distributors, retailers and customers, a definite volume of goods to be transported and a fast and responsive transportation system, to give just the most important requirements. Deciding which is the best distribution strategy for a particular product of a company can make an enormous impact on the success of that company. Therefore, there is the need for a DSS that helps executive

managers to select the best distribution strategies and, at the warehouse level, to exercise decisions to make the movement and storage operations more efficient.

2.5 Inventory Management

The importance of inventory management and the relationship between inventory and customer service is essential in any company. As for the location issues, inventory management has been well studied in OR literature; however, the use of inventory systems in helping decision-making processes has been less widespread. Most of the well known models in literature are simple and do not, for example, consider multi-product inventory management that requires the same resources, or, in some cases, do not treat all the complexities involved in inventory management such as demand uncertainty, returns and incidents. So far, the better known inventory models and systems consider a single facility managing its inventories in such a way as to minimize its own costs. As we have mentioned, one major challenge in LM is the integration and coordination of all logistics activities in the supply chain, a particularly important issue being inventory management within the whole supply chain in order to minimize systemwide costs. This requires models and DSS that are able to aid decisions and suggest policies for inventory management in the whole supply chain. To solve such a complex issue, we will argue that DSS which combine simulation and metaheuristics techniques can be of great help.

2.6 Product Design

Products are a main element in the supply chain, which should be designed and managed in such a way as to enable efficient flow of these products. This approach is known as “design for supply chain” and is likely to become frequently used in the future. The characteristics of the product, such as weight, volume, parts, value, perishability, etc., influence the decisions made in relation to a supply chain, since the need for warehousing, transportation, material handling and order processing depend on these attributes. Products designed for efficient packaging and storage obviously make an impact on the flow in the supply chain and cost less to transport and store. During the design process of a new product, or changes to an existing one, the requirements of the logistics relating to product movements should be taken into consideration. Also, the need for short lead times and the increased demand from customers for unique and personalized products put pressure on efficient product design, production and distribution. Postponement is one successful technique that can be applied to delay product differentiation and also lead to an improvement in the logistics of the

product, Lee, Billington and Carter (1993). The use of information systems and simulation techniques that help to analyze the impact on the supply chain of a certain design of a specific product can be of great help to managers.

2.7 Material Handling and Order Picking

Material handling is a broad area that basically encompasses all activities relating to the movement of raw material, work in process or finished goods within a plant or warehouse. Moving a product within a warehouse is a no value-added activity but it incurs a cost. Order processing or picking basically includes the filling of a customer order and making it available to the customer. These activities can be quite important since they have an impact on the time that it takes to process customer orders in the distribution channel or to make supplies available to the production function. They are cost absorbing and therefore need attention from the managers. Packaging is valuable both as a form of advertising and marketing, as well as for protection and storage from a logistical perspective. Packaging can ease movements and storage by being properly designed for the warehouse configuration and material handling equipment.

The major decisions in this area include many activities, such as facility configuration, space layout, dock design, material-handling systems selection, stock locator and arrangement, equipment replacement, and order-picking operations. Most of the models and techniques available these days consider the above decision processes as activities independent of the remaining ones in the whole system. Therefore, DDS that analyze the impact of material handling and order picking activities on the logistics system and enable the decision-maker to make the best decision for the whole network, are an important and essential tool.

2.8 Logistics of Production and Scheduling

The most common definition of production and operations management (POM) is as follows: the management of the set of activities that creates goods and services through the transformation of inputs into outputs, Chase, Aquilano and Jacobs (2004), Stevenson (1999). The interaction between POM and LM is enormous, since production needs raw materials and parts to be able to produce a commodity, and then this commodity must be distributed, Graves, Rinnoy Kan and Zipkin (1993). Therefore, coordination between both areas is fundamental to an efficient supply chain. The techniques required to plan and control the production in an integrated supply chain go beyond the MRP (Material Requirement Planning) so popular in industries. The need to take into consideration manufacturing or service capacity, labor and time constraints has given importance to the Scheduling

area. This field is extremely wide; however research at a scientific level has focused mainly on the formalization of specific problem types, leading to standard problems like the flow-shop scheduling problem, job-shop scheduling problems, etc. A significant amount of research has been dedicated to the classification of problem difficulty by deriving complexity results for a large variety of problem variants and the development of efficient solution techniques for standard scheduling problems, Pinedo (1995). Research efforts in the latter area have shown that in the case of many problems, the use of heuristic algorithms, which cannot guarantee optimal solutions, but were able, in a large number of experiments, to find extremely high quality solutions in a short time, are currently the most promising techniques for solving difficult scheduling problems. Despite efforts in academic scheduling research, there is still a considerable gap in the application to practical problems of the techniques developed on the academic side. Scheduling problems are already quite hard to solve per se, and their extension to include aspects of the whole supply chain significantly increases their complexity. Moreover, in many supply chains, the bottleneck activity is production, therefore efficient planning and managing of production and scheduling activities within the coordination of the supply chain is of great importance to an efficient supply chain. The development of production and scheduling models and solving techniques that consider the logistics activities related are a challenge for both academia and industry. Some ERP providers have already incorporated metaheuristics for solving complex scheduling problems, such as SAP (www.sap.com) with its product APS (Advanced Planning and Scheduling). We do believe that in the future many more Information Technology companies will make use of metaheuristic techniques to solve those very difficult problems, such as those relating to integrated logistics and production scheduling.

2.9 Information Systems and DSS

Computer and information technology has been utilized to support logistics for many years. Information technology is seen as the key factor that will affect the growth and development of logistics, Tilanus (1997). It is the most important factor in an integrated supply chain, also playing an important role in the executive decision-making process. More sophisticated applications of information technology such as decision support systems (DSS) based on expert systems, simulation and metaheuristics systems will be applied directly to support decision making within modern businesses and particularly in LM. A DSS incorporates information from the organization's database into an analytical framework with the objective of easing and improving the decision making. A critical element in a DSS for logistics

decisions is the quality of the data used as input for the system. Therefore, in any implementation, efforts should be made to ensure the data is accurate. Consequently, modeling and techniques can be applied to obtain scenarios and analysis of the logistics situations within the environment of the company and, can be used to support the managers and executives in their decision processes.

We believe that metaheuristics, when incorporated into a DSS for LM, can contribute significantly to the decision process, particularly when taking into consideration the increased complexity of the logistics problems previously presented. DSS based on metaheuristics are not currently widespread, but the technique appears to be growing as a potential method of solving difficult problems such as the one relating to LM.

2.10 E-commerce and E-logistics

In just a few short years, the Internet has transformed the way in which the world conducts business and business partners interact between themselves. E-business and electronic commerce are some of the hottest topics of our days, Deitel, Deitel and Steinbuhler (2001), Chaffey (2001). In e-commerce, business partners and customers connect together through Internet or other electronic communication systems to participate in commercial trading or interaction. We will not discuss e-commerce in detail at this stage, but it certainly makes new and high demands on the company's logistics systems, calling in, in some cases, completely new distribution concepts and a new supply chain design. Companies are looking for DSS, such as the one relating to e-commerce and e-business that help them to make the best decisions in an uncertain and rapidly changing world. Many of the problems can be seen as extensions of the ones described above, such as, for example, transportation management, while others are completely new with some added complexities such as the uncertainties associated with the evolution of commerce on the web. An example of new problems that can appear relate to home distribution, generated by business-to-consumer (B2C), during non-labor hours and the search for a solution which will allow an efficient distribution. An example of this is the inclusion of 24-hour dropping-points, where transportation companies can leave a package that will be collected later by the customer, thus avoiding the need for distribution during nighttime or on Saturdays and Sundays. Questions as to the location and size, for example, of these dropping-points, frequency of visits, partnership with stores, etc. are issues that have not yet been dealt with in metaheuristics and logistics literature.

2.11 Reverse and Green Logistics

Concern over the environment has never been as strong as today. Also, strict regulations regarding removal, recycling and reuse are on the increase, especially in Europe. This will bring Reverse Logistics and Green Logistics into the main focus in the near future, Rogers and Tibben-Lembke (1998). Reverse logistics are related to the process of recycling, reusing and reducing material, i.e. goods or materials that are sent "backwards" in the supply chain. The issues faced in reverse logistics are not just the "reverse" issues of a traditional supply chain, they can be more complex, such as, for example, aspects relating to the transportation and disposal of dangerous materials. Manufacturers in Europe will soon be held responsible for the total cost of recycling or disposal of all materials used in their product. This legislation will put an enormous emphasis on efficient reverse logistics decisions that will need to be optimized. Green logistics is generally understood as being activities relating to choosing the best possible means of transportation, load carriers and routes and reducing the environmental impact of the complete supply chain. Some of the areas clearly affected are product packaging, transportation means and product development, as well as many others. Logistics is also involved in the removal and disposal of waste material left over from the production, distribution or packaging process, as well as the recycling and reusable products.

All the above points make the relevance of the reverse and green logistics area clear, since many companies have to re-organize their supply chains and even extend them in order to be able to return, reuse or dispose of their product and materials. This poses many new and challenging questions to the area of LM.

2.12 Customer Service

Customers have never before been taken so seriously. The successful fulfillment of customer expectations is a task pertaining to LM, and deciding the level of customer service to offer customers is essential to meeting a firm's profit objective. Customer service is a broad term that may include many elements ranging from product availability to after-sales maintenance. In brief, customer service can be seen as the output of all logistics activities, that also interact with other functions in the firm, especially with marketing. Since all the elements in the supply chain interact and a decision on one element affects all the others, any logistic decision within the supply chain can affect the customer service. Therefore, systemwide DSS that help the decision maker at a strategic, tactical and operation level, to evaluate, simulate and analyze different options and scenarios, and the interaction

between the players in a supply chain are being requested more and more by many companies.

We have briefly reviewed some actual issues and aspects of the logistics management of an integrated supply chain. The problems, in general, are complex and the decision maker will benefit from having a DSS that can generate several scenarios and what-if analyses in a short time, allowing him to analyze the impact of one decision on the whole system. In the next chapter we will argue that metaheuristics can be an excellent tool to be included in such a DSS for LM.

3. Metaheuristics for LM

As we have seen in previous sections, the supply chain is a complex network of facilities and organizations with interconnected activities, but different and conflicting objectives. Many companies are interested in analyzing the logistics activities of their supply chain as an entire and unique system in order to be able to improve their business. However, in most cases, the task of designing, analyzing and managing the supply chain has been carried out based on experience and intuition; very few analytical models and design tools have been used in the process. This means that finding the best logistics strategies for a particular firm, group of firms or industry poses significant challenges to the industry and academia. We argue that metaheuristics can be an important aid to managers and consultants in the decision process.

Optimization literature focuses on algorithms for computing solutions to constrained optimization problems. An exact or optimal algorithm in the optimization context refers to a method that computes an optimal solution. A heuristic algorithm (often shortened to heuristic) is a solution method that does not guarantee an optimal solution, but, in general, has a good level of performance in terms of solution quality or convergence. Heuristics may be constructive (producing a single solution) or local search (starting from one or given random solutions and moving iteratively to other nearby solutions) or a combination of the two (constructing one or more solutions and using them to start a local search). A metaheuristic is a framework for producing heuristics, such as simulated annealing and tabu search. To develop an heuristic for a particular problem some problem-specific characteristics must be defined, but others can be general for all problems. The problem-specific may include the definition of a feasible solution, the neighborhood of a solution, rules for changing solutions, and rules for setting certain parameters during the course of execution. For a general discussion on heuristics see Corne, Dorigo and Glover (1999), Aarts and Lenstra (1997) and Glover and Kochenberger (2001).

Well-designed heuristics packages can maintain their advantage over optimization packages in terms of computer resources required, a consideration unlikely to diminish in importance so long as the size and complexity of the models arising in practice continue to increase. This is true for many areas in the firm, but especially for LM related problems.

Metaheuristics have many desirable features making them an excellent method for solving very complex LM problems: in general they are simple, easy to implement, robust and have been proven highly effective in solving difficult problems. Even in their simplest and most basic implementation, metaheuristics have been able to effectively solve very difficult and complex problems. Several other aspects are worth mentioning. The first one is the modular nature of metaheuristics giving rise to short development times and updates, a clear advantage over other techniques for industrial applications. This modular aspect is especially important given the current times required for implementing a DSS in a firm and the rapid changes that occur in the area of LM.

The next important aspect is the amount of data involved in any optimization model for an integrated logistic problem, which can be overwhelming. The complexity of the models for LM and the inability to solve the problems in real time using traditional techniques, necessitate the use of the obvious technique for reducing this complex issue: data aggregation, Simchi-Levi and Kaminsky (2000). However, this approach can hide important aspects that have an impact on decisions. For example, consider the aggregation of customers by distance, customers located near to another can be aggregated, but suppose they require a totally different level of service? Therefore, instead of aggregating data so as to be able to obtain a simple and solvable model, but one which is not a good reflection of the reality, maybe we should consider the complex model but using an approximation algorithm.

The last aspect that we would like to mention in favor of using metaheuristics is the estimation of costs, such as transportation and inventory costs. Why spend time on an optimal solution to a model when the data in hand consists solely of estimations? Maybe we should use the time to produce several scenarios for the same problem. For example, various possible scenarios representing a variety of possible future demand patterns or transportation costs can be generated. These scenarios can then be directly incorporated into the model to determine the best distribution strategy or the best network design. The scenario-based approaches can incorporate a metaheuristic to obtain the best possible decision within a scenario. The combination of the best characteristics of human decision-making and a computarized model and algorithmic based systems into interactive and graphical design frameworks have proven to be very effective in LM, since

many integrated logistic problems are new, subject to rapid changes and, moreover, there is no clear understanding of all of the issues involved.

Hax and Candea (1984) proposed a two-stage approach to solving LM problems and take advantage of the system dynamics:

1. Use an optimization model to generate a number of least-cost solutions at the macro level, taking into account the most important cost components.
 2. Use simulation models to evaluate the solutions generated in the first phase.

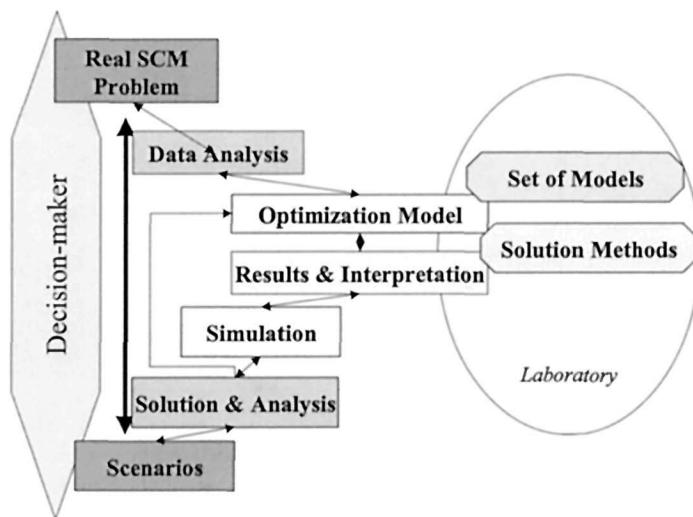


Figure 15.2. A DSS scheme for Logistics Management

In Figure 15.2, we present a scheme for a DSS combining simulation and optimization techniques. We argue that the user can analyze more and better scenarios within the same time framework if metaheuristics techniques are used as the solution method instead of the exact method or other heuristics techniques. We use the laboratory to study the application of metaheuristics to well-known models, with the objective of providing guidelines for real applications. Moreover, upgrades of the DSS can be easily developed and also implementing for a specific firm with specific logistic problems.

Metaheuristics will help users to take system decisions within certain parameters and environments and then simulation techniques can be applied to analyze the system behavior in the presence of uncertainties. Simulation-based tools take into account the dynamics of the system and are capable of characterizing system performance for a given design (or decisions). The limitations of the simulation models are that they only represent a pre-specified system, i.e. given a particular configuration, a simulation model can be used to help estimate the costs associated with operating the configuration (simulation is not an optimization tool). Therefore the combination metaheuristics-simulation can provide a very interesting approach to the solving of complex logistic problems. The use of simulation has produced widespread benefits in the decision process within firms, however, simulation-based tools have not been good in proposing the optimal or near optimal solution of several possible solutions. On the other hand, mathematical programming models and techniques are able to find the best solutions, but not able to simulate the behavior and effects of a particular decision in the presence of uncertainties. Recent developments are changing this, and the decision making process can benefit enormously by having a system that is able to identify and evaluate the optimal or near optimal solution in the presence of uncertainties. These advances have been made possible by the developments in heuristic research, particularly in metaheuristics. The OptQuest computer software, by Glover, Kelly and Laguna, of OptTek Systems, Inc. (<http://www.opttek.com/>) already offers this innovation, Laguna (1997, 1998), Glover, Kelly and Laguna (2000). OptQuest replaces the inaccuracy of trial-and-error usual in simulation systems by using a potent search engine that can find the best decisions that fall within a domain that the simulation or other evaluation model encompasses. Actually, the OptQuest has been integrated with several commercial simulation packages.

We believe that in the future more combinations of simulation and optimization techniques will be developed. Metaheuristics techniques play a very important role in this direction since they can obtain very good solutions within a small time framework, which can be easily adapted and developed to solve very complex logistic problems.

Next, we focus on three metaheuristics: iterated local search, tabu search and scatter search. Many others have similar features and are also potential methods that could be applied to LM problems. We discuss these ones because, in their simple form, they present quite good results and are somehow representative of the latest developments in modern heuristic research. At the end of the chapter we will comment on common aspects of these metaheuristics that can be relevant in solving LM problems, Lourenço, Martin and Stützle (2001).

Iterated Local Search (ILS) is a simple, yet powerful metaheuristic to improve the performance of local search algorithms. Its simplicity stems from the underlying principle and the fact that only a few lines of code have to be added to an already existing local search algorithm to implement an ILS algorithm. ILS is currently one of the best performing approximation methods for many combinatorial optimization problems.

To apply an ILS algorithm to a given problem, three “ingredients” have to be defined. One is a procedure called *Perturbation* that perturbs the current solution s (usually a local optimum) giving rise to an intermediate solution s' . Next, a *Local Search* is applied taking s' to a local minimum s'' . Finally, one has to decide which solution should be chosen for the next modification step. This decision is made according to an *Acceptance Criterion* that takes into account the previous solution s , the new candidate solution s'' and possibly the search history. For an extended reference on ILS see Lourenço, Martin and Stützle (2001). An algorithmic outline for the ILS is given in Figure 15.3.

Iterated Local Search Procedure:

```

 $s_0 = \text{GenerateInitialSolution};$ 
 $s^* = \text{LocalSearch}(s_0);$ 
Repeat
   $s' = \text{Perturbations}(s^*, \text{history});$ 
   $s'' = \text{LocalSearch}(s');$ 
   $s^* = \text{AcceptanceCriterion}(s^*, s'', \text{history});$ 
Until termination criterion met
end return  $s^*$ .

```

Figure 15.3. Iterated local search

Tabu Search is an adaptive procedure originally proposed by Glover (1986). Recently, this metaheuristic has been gaining ground as a very good search strategy method for solving combinatorial optimization methods. For a survey, see Glover and Laguna (1997). We sketch a simplified version of the approach, as follows.

The basic idea of tabu search is to escape from a local optimum by means of memory structures. Each neighbor solution is characterized by a move and in the illustrated simplified formulation short term memory is used to memorize the attributes of the most recently applied moves, incorporated via

one or more tabu list. Therefore, some moves are classified as tabu and, as a result, some neighbor solutions are not considered. To avoid not visiting a good solution, an aspiration criterion can be considered. At each iteration, we choose the best neighbor to the current solution that is neither tabu nor verifies an aspiration criterion. The aspiration criterion used here was the most common one; the tabu status is overruled if the neighbor solution has an objective function value smaller than the best value found up to that iteration. The algorithm stops when a certain stopping-criterion is verified. The best solution found during the search is then the output of the method. The main steps of the tabu search algorithm are given in Figure 15.4.

Tabu Search Procedure

$x = \text{GenerateInitialSolution};$

repeat

Let $x' = \text{neighbor}(x)$, not tabu or satisfying an aspiration criteria, with minimal value of the objective function;

Set $x = x'$ and update the tabu list and aspiration criteria;

until *termination criterion met*

end return the best solution found.

Figure 15.4. Short term tabu search procedure

Scatter search, from the standpoint of metaheuristic classification, may be viewed as an evolutionary (also called population-based) algorithm that constructs solutions by combining others. It derives its foundations from strategies originally proposed for combining decision rules and constraints (in the context of integer programming). The goal of this methodology is to enable the implementation of solution procedures that can derive new solutions from combined elements in order to yield better solutions than those procedures that base their combinations only on a set of original elements. As described in tutorial articles (Glover 1998 and Laguna 1999) and other implementations based on this framework (Campos, Laguna and Martí 1998), the methodology includes the following basic elements:

- Generate a population P .
- Extract a reference set R .
- Combine elements of R and maintain and update R .

Scatter search finds improved solutions by combining solutions in R . This set, known as the reference set, consists of high quality solutions that are also diverse. The overall proposed procedure, based on the scatter-search elements listed above, is as follows:

Generate a population P : Apply the diversification generator to generate diverse solutions.

Construct the reference set R : Add to R the best r_1 solutions in P . Add also r_2 diverse solutions from P to construct R with $|R| = r_1 + r_2$ solutions.

Maintain and update the reference set R : Apply the subset generation method (Glover 1998) to combine solutions from R . Update R , adding solutions that improve the quality of the worst in the set.

Since the main feature of the scatter-search is a population-based search, we believe it will be an adequate technique for solving difficult and large-scale multi-objective problems by finding an approximation of the set of Pareto-optimal solutions. Therefore, the inclusion of scatter search methods on scenario-based DSS software can have an important impact, since several good scenarios can be obtained in just one run. Several applications have already been completed, Martí, Lourenço and Assignment (2000) and Corberán et al.(2000), and we believe more will appear in literature in the near future.

All the above metaheuristics have this in common: even the simpler implementations are able to solve difficult problems in a short amount of running time. Moreover, the inclusion of new features, constraints, objective functions can be relatively simple, which is quite important in the rapidly changing world of LM.

Given a description of the problem and the necessary data, the above metaheuristics have the following modules in common:

- Generate an initial solution or a population of solutions;
- Obtain the objective value of a solution;
- Obtain the neighborhood of a solution;
- Perform a perturbation on a solution or obtain a subset of combined solutions;
- Perform an acceptance test;
- Stopping criteria.

All of the above modules can be developed for complex problems, regardless of the specific mathematical characteristics of the problem, as linear or non-linear functions, and for the specific characteristics of the logistic problem within a firm. The integration aspects of a logistic problem can be implemented by adapting the objective function, or the multi-objective functions, and by taking this into account in the neighborhood definition. For example, if a metaheuristic has been developed to solve the logistic problem of a firm, and the consulting company needs to solve a similar problem, but with different aspects, for another firm, the adaptation can be relatively

simple since many modules can be the same. This gives metaheuristics a great advantage. Moreover, since the methods share modules, it would be easy to implement several metaheuristics for the same problem in hand. As mentioned, the logistics decision-makers are experiencing enormous changes every day and challenges that need a quick response. The consideration of these changes can be quickly incorporated into a metaheuristics-based DSS without the need for modification of the complete software, which is an enormous advantage for software and consulting companies.

Problems in the LM area can also benefit from the vast amount of successful applications of metaheuristics in combinatorial optimizations and other well-known difficult problems. From this list, the DSS developer can learn what can be the best approach to a specific LM problem by studying the combinatorial optimization problem that has the most aspects in common with the specific real problem, such as, for example, the best metaheuristics, the best neighborhood, etc. Research on metaheuristics should give insights and guidelines for future implementations of metaheuristics into other similar problems, so commercial producers of software and consulting companies could develop the most adequate solution method to a specific problem. The excellent results, modularity and flexibility of metaheuristics are the major advantages of this technique for solving LM problems. As we have discussed, there is enormous potential in LM for applications of metaheuristics. Research academia and industry should both benefit from these applications.

Next, we will present some applications that exemplified the advantages of using a metaheuristic in an LM decision process and their role in the logistics management of an integrated supply chain.

4. Applications

We can already find several applications of metaheuristics in LM problems and incorporation of metaheuristics in DSS into LM, however, they are not yet as widespread as might be expected given the potential of the technique. We will now review some successful logistics applications, ranging from vehicle routing to container operation problems. The objective is not to make a survey on the applications of metaheuristics to LM, but to give a few examples of the possibilities of metaheuristics in LM.

Weigel and Cao (1999) presented a vehicle-routing DDS to help the decision process relating to the home-delivery and home-service business for Sears, Roebuck and Company (www.sears.com). The system was developed together with a software company ESRI (www.esri.com) and is based on a combination of geographical information systems (GIS) and operations research. More specifically, the main techniques used in the development of the algorithms behind the DSS are local search and tabu search methods. The

algorithms and their technical implementations have proven to be generic enough to be successfully applied to other types of business. This generic capability derives from using the OptQuest Engine (www.opttek.com) to adaptively tune the parameter settings for different regions. The system has improved the Sears technician-dispatching and home-delivery business resulting in an annual saving of over \$42 million. This is a clear example of how metaheuristics integrated in a DSS for SCM can make a strong impact on a company by helping them, within the decision process, to gain understanding of the problem, use their resources more efficiently, give a better customer service and finally, but of no less importance, to reduce costs.

Ribeiro and Lourenço (2001) presented a complex vehicle routing model for distribution in the food and beverages industries. The main idea is to design routes taking into consideration the varying responsibilities of different departments in a firm. This cross-function planning is the basis for obtaining integrated logistics. The authors propose a multi-objective multi-period vehicle routing model, where there are three objective functions that respond to three different areas; the usual cost function which is the responsibility of the distribution department; a human resources management objective which related to a fair work load, and, in the case of variable salary, also relates to a fair equilibrium of possible percentages of sales; and finally a marketing objective, which aims to always assign the same driver to the same customer in order to improve customer service. To be able to solve such a complex model in a short space of time, or integrate a solution method within a DSS to help distribution logistics, the solution method must give a solution in a very short time and allow simple updates and changes during the installation process and future use. This, of course, advocates metaheuristics techniques. In their report, Ribeiro and Lourenço (2001) proved the importance of taking several functions and the difficulty of solving the model even for very small instances of the problem. They propose an ILS method to solve the problem.

Other applications of logistics relating to vehicle routing can be found in literature, such as the inventory routing problem for vending machines, Kubo (2001).

Ichoua, Gendreau and Potvin (2000) present a new strategy for the dynamic assignment of new requests in dynamic vehicle routing problems which include diversion. These dynamic vehicle routing problems are common in organizations such as courier services, police services, dial-and-ride companies and many others. In the dynamic context, each new request is inserted in real time in the current set of planned routes, where a planned route is the sequence of requests that have been assigned to a vehicle but not yet served. A tabu search heuristic was used to make an empirical evaluation of the new strategy. The results demonstrate the potential savings in total

distance, total lateness and number of unserved customers when compared to a simple heuristic where the current destination of each vehicle is fixed. This application shows a potential use of metaheuristics, not only as a direct aid in operational decisions, but, more relevantly, as an aid in the identification of the best strategies for handling highly dynamic problems such as real-time vehicle dispatching.

Bosë et al. (2000) describe the main processes at a container terminal and the methods, based on evolutionary algorithms, currently used to optimize these processes. They focus on the process of container transport, by gantry cranes and straddle carriers, between the container vessel and the container yard. The reduction in the time spent by the vessel in port, the time required for loading and unloading the vessel and the increase in the productivity of the equipment are main objectives for the management of a container yard. The global increment in container transportation, the competition between ports and the increase in multi-modal parks give rise to the need for improved techniques to help the decision process of the senior management of a container terminal. Bosë et al. (2000) proved that with a simple genetic algorithm, combined with a reorganization of the process, the amount of time in port for container vessels can be reduced, leading to a competitive advantage for the container terminal. As future research, they expect to develop a hybrid system using simulation and evolutionary methods which will allow uncertainties to be taken into account. This report is a good example of the direction LM is following in order to be able to solve the complex problems in the area.

Fanni et al. (2000) describes the application of a tabu search to design, plan and maintain a water distribution system. Since water is a sparse resource, especially in some countries, and the design and maintenance of pipe networks for water supply distribution involve high costs, achieving the highest level of performance of existing networks at minimum cost is mandatory. The complexity of a real water distribution network grows with the necessity to consider non-smooth non-convex large-size problems and discrete variables. This is a clear application in the continuous flow industry that can be seen as an application in the area of green logistics.

In service industries, the logistics to produce a service are highly dependent on the human resources. Therefore, in this firm the most important problem can be the crew or personnel scheduling. Many authors have applied metaheuristics to crew scheduling in airline industries, Campbell, Durfee and Hines (1997), and bus and train companies, Cavique, Rego and Themido (1998), Kwan et al. (1997), to mention just a few.

Scheduling is another area where a vast amount of metaheuristics applications are to be found, see, for example, Voß et al. (1998) and Osman and Kelly (1996). However, most of the applications focus on a specific

scheduling problem and little attention has been given to the integration of logistics into a supply chain. The main applications are for job-shop scheduling problems or similar, however, these models pay little attention to the integration of production scheduling with the rest of the elements in the supply chain. However, efficient production scheduling is enormously relevant to logistics integration within a supply chain, as discussed in the previous chapter. So, aspects such as customer service and delivery times must be integrated into the scheduling decisions, turning, in many cases, into non-linear multi-objective problems.

For an extensive list of applications, many in the area of LM, we refer the author to Glover and Laguna (1997). We believe that we have missed many references on the applications of metaheuristics to supply chain problems. However, our intention in writing this report, was not to carry out a complete survey on the issue (something that we would like to do in the near future), but to bring the reader's attention to potential of metaheuristics in the field of LM, especially when logistics integration has to be taken in account.

5. Conclusions

Logistics management in a supply chain offers significant benefits to the elements across the chain, reducing waste, reducing cost and improving customer satisfaction. However, this strategy is a challenging and significant task for companies, decision-makers, consultants and academics. The process of implementing and managing integrated logistics has been shown to be very difficult. We have discussed several important logistics activities within a supply chain and their interrelationships.

Many other issues and questions surrounding LM are not treated in the paper, since this is a rapidly changing world with new challenges appearing every day. We strongly believe that the recent developments in the area of metaheuristics techniques will put them on the front page as regards solving existing LM problems and new complex ones that arise due to the lack of any integrated management. Their modularity, easy implementation, easy updating and adaptation to new situations combined with simulation systems and DSS can make a strong positive impact on the decision process in LM. We have focused on Iterated Local Search, Tabu Search and Scatter Search as being some metaheuristics that present characteristics for potential successful application to LM. Developers can learn from the extensive applications of these metaheuristics to well-known optimization problems, and consequently, these methods have short development and implementation times.

With this paper, we hope to contribute to a better understanding of the issues involved in integrated logistics and to encourage further research on the applications of metaheuristics for solving complex LM problems.

Metaheuristics can make an important contribution to coping with the challenges posed by LM, especially with the new economy and electronic business. Applications of metaheuristics-based DSS for LM are work-in-process. In many companies, ambitious projects to implement DSS to evaluate and help the decision process of the integrated logistics within a supply chain have yet to be completed, and many others have not yet begun serious initiatives in this direction. We believe that this work should be updated sometime in the near future, as a large amount of successful applications of metaheuristics-based DSS in LM problems will be developed.

Acknowledgments

The research was funded by Ministerio de Ciencia y Tecnología, Spain (BEC2000-1027). I would like to thank Fred Glover, Cesar Rego and Gary Kochenberger (Hearin Center for Enterprise Science, University of Mississippi) for inviting me to present this research at *The International Conference on Memory and Evolution: Scatter Search and Tabu Search*, Oxford, Mississippi, March 7-10, 2001.

References

- Ballou, R.H. (1999) *Business Logistics Management*, Prentice-Hall, New Jersey.
- Beamon, B.M. (1999) "Designing the Green Supply Chain," *Logistics Information Management* 12(4):332-342.
- Blanchard, B.S. (1992) *Logistics Engineering and Management*, Prentice-Hall.
- Bloemhof-Ruwaard, J.M., P. van Beek, L. Hordijk and L.N. van Wassenhove (1995) "Interactions between Operational Research and Environmental Management," *European Journal of Operations Research* 85:229-243.
- Böse, J., T. Reiners , D. Steenken and S. Voß (2000) "Vehicle Dispatching at Seaport Container Terminals using Evolutionary Algorithms," Proceeding of the 33rd Hawaii International Conference on System Sciences, 1-10.
- Bowersox, D. and D. Closs (1996) *Logistical Management: The Integrated Supply Chain Process*, McGraw-Hill.
- Bramel, J. and D. Simchi-Levi (1997) *The Logic of Logistics: Theory, Algorithms and Applications for Logistics Management*, Springer-Verlag, New York.
- Campbell, K.W., R.B. Durfee and G.S Hines (1997) "FedEX Generates Bid Lines using Simulated Annealing, " *Interfaces* 27(2):1-16.
- Campos, V., M. Laguna and R. Martí (1998) "Scatter Search for the Linear Ordering Problem," in *New Methods in Optimisation*, D. Corne, M. Dorigo and F. Glover (Eds.), McGraw-Hill.
- Cavique, L., C. Rego and I. Themido (1998) "New Heuristic Algorithms for the Crew Scheduling Problem," in S. Voß, S. Martello, I.H. Osman, C.

- Roucairol, "Metaheuristics: Advances and Trends in Local Search Paradigms for Optimization", Kluwer Academic Publishers, 37–47.
- Chaffey, D. (2001) *E-business and E-commerce Management: Strategy, Implementation and Practice*, Prentice Hall.
- Cooper, M.C., D.M. Lambert and J.D. Pagh (1997) "Supply Chain Management: More Than a New Name for Logistics," *The International Journal of Logistics Management*, 8(1): 1–14.
- Corberán, A., E. Fernández, M. Laguna and R. Martí (2000) "Heuristic Solutions to the Problem of Routing School Buses with Multiple Objectives," Working Paper, Universitat de Valencia, Spain.
- Corne, D., M. Dorigo and F. Glover (1999) *New Ideas in Optimisation*, McGraw-Hill.
- Coyle, J.J., E.J. Bardi and C.J. Langley, Jr. (1988) *The Management of Business Logistics*, West Publishing Company.
- Crainic, T.G. and G. Laporte (1998) *Fleet Management and Logistics*, Kluwer Academic Publisher, Massachusetts.
- Chase, R.B., F.R. Jacobs and N.J. Aquilano (2004) *Operations Management for Competitive Advantage*, 10th ed., Irwin, McGraw-Hill.
- Daganzo, C.F. (1997) *Fundamentals of Transportation and Traffic Operations*, Pergamon-Elsevier.
- Dale, S.R. and R.S. Tibben-Lembke (1998) Going Backwards: Reverse Logistics Trends and Practices, Reverse Logistics Executive Council.
- Daskin, M.S. (1995) *Network and discrete location: models, algorithms, and applications*, John Wiley, New York.
- Dornier, P.P., R. Ernst, M. Fender and P. Kouvelis (1998) *Global Operations and Logistics*, John Wiley & Sons.
- Deitel, H. M., P. J. Deitel and Steinbuhler, K. (2001) *E-business and e-commerce for managers*, Prentice Hall.
- Drezner, Z. (1995) *Facility Location. A Survey of Applications and Methods*, Springer Verlag, Berlin.
- Elliam, T. and G. Orange (2000) "Electronic Commerce to Support Construction Design and Supply-Chain Management: a Research Note," *International Journal of Physical Distribution and Logistics Management*, 30(3/4): 345–360.
- Fanni, A., S. Liberatore, G.M. Sechi, M. Soro and P. Zuddas (2000) "Optimization of Water Distribution Systems by a Tabu Search Method," in *Computing Tools for Modeling, Optimization and Simulation*, M. Laguna and J.L. González Velarde (eds.), Kluwer Academic Publishers, 279–298.
- Fine, C. (1998) *Clockspeed: Winning industry control in the age of temporary advantage*, Perseus Books.
- Fleischman, M., J.M. Bloemhof-Ruwaard, R. Dekker, E. van der Laan, J.A.E.E. van Nunen and L.N. van Wassenhove (1997) Quantitative Models for Reverse Logistics: a Review, *European Journal of Operations Research* 103:1–17.
- Fleischmann, B., J.A.E.E. van Nunen, M. Grazia Speranza and P. Stähly (Eds.) (1998) *Advances in Distribution Logistics*, Springer-Verlag, New York.

- Glover, F. and G. Kochenberger (2001) (eds.) *Handbook in Metaheuristics*, Kluwer Academic Publishers.
- Glover, F., J.P. Kelly and M. Laguna (2000) "The OptQuest Approach to Crystal Ball Simulation Optimization" Decisioneering White Paper. (http://www.decisioneering.com/articles/article_index.html).
- Glover, F. (1998) "A Template for Scatter Search and Path Relinking," in *Artificial Evolution, Lecture Notes in Computer Science* 1363, J.K. Hao, E. Lutton, E. Ronald, M. Schoenauer and D. Snyers (Eds.), Springer-Verlag, 3–54.
- Graham, G. and G. Hardaker (2000) "Supply-Chain Management across the Internet", *International Journal of Physical Distribution and Logistics Management* 30(3/4):286–295.
- Graves, S.C., A.H.G. Rinnoy Kan and P.H. Zipkin (1993) *Logistics of Production and Inventory*, North-Holland, Elsevier Science.
- Hax, A.C. and D. Candea (1984) *Production and inventory management*, Prentice Hall.
- Ichoua, S., M. Gendreau and J.Y. Potvin (2000) "Diversion Issues in Real-Time Vehicle Dispatching," *Transportation Science* 34(4):426–438.
- James, F.R. and W.C. Copacino, (1994) *The Logistic Handbook*, Andersen Consulting, Free Press.
- John, E.S. (1992) *Logistics Modeling*, Pitman Publishing.
- Johnson, J.C., D.F. Wood, D.L. Wardlow and P.R. Murphy (1999) *Contemporary Logistics*, Prentice Hall Business Publishing
- Kasilingam, R.G. (1999) *Logistics and Transportation Design and Planning*, Kluwer.
- Kubo, M. (2001) "On the Inventory Routing Problem," *International Conference on Adaptive memory and Evolution: Tabu search and Scatter Search*, Oxford, Mississippi, March 7–10.
- Kwan, A.S.K., R.S.K. Kwan, M.E. Parker and A. Wren (1997) "Producing Train Driver Schedules under Operating Strategies," in *Preprints of the 7th International Workshop on Computer-Aided Scheduling of Public Transportation*, Boston, USA.
- Laguna, M. (1997) Metaheuristic Optimization with Evolver, Genocop and OptQuest, *EURO/INFORMS Joint International Meeting, Plenaries and Tutorials*, J. Barceló (Ed.), 141–150.
- Laguna, M. (1998) Optimization of Complex Systems with OptQuest, OptQuest for Crystal Ball User Manual, Decisioneering.
- Laguna, M. (1999) "Scatter Search," to appear in *Handbook of Applied Optimization*, P. M. Pardalos and M. G. C. Resende (Eds.), Oxford Academic Press.
- Lambert, D., R. Stock and L.M. Ellram (1998) *Fundamentals of Logistics*, McGraw-Hill.
- Lapide, L. (1998) "Supply Chain Planning Optimization," *The Report on Supply Chain Management, Advanced Manufacturing Research*.
- Lee, H., P. Padmanabhan and S. Whang (1997) "The Paralyzing Curse of the Bullwhip Effect in a Supply Chain," *Sloan Management Review*, Spring 93–102.

- Lee, H.L., C. Billington and B. Carter (1993) "Hewlett-Packard Gains Control of Inventory and Service through Design for Localization," *Interfaces* 23(4):1–11.
- Lourenço, H.R., O. Martin and T. Stützle (2001) "Iterated Local Search," in *Handbook of Metaheuristics*, F.Glover and G. Kochenberger, (eds.), Kluwer Academic Publishers, International Series in Operations Research & Management Science, 321–353.
- Martí, R., H. Lourenço and M.L. Assigning (2000) "Proctors to Exams with Scatter Search," published in "*Computing Tools for Modeling, Optimization and Simulation*", M. Laguna and J.L. González Velarde (eds.), Kluwer Academic Publishers, 215–228.
- Miller, T.C., T.C. Friesz and R. Tobin (1996) *Equilibrium Facility Location on Networks*, Springer, Berlin.
- Osman, I.H. and J.P. Kelly (1996) *Meta-Heuristics: Theory and Applications*, Kluwer Academic Publishers.
- Pinedo, M. (1995) *Scheduling: Theory, Algorithms and Systems*, Prentice-Hall.
- Rayward-Smith, V.J., I.H. Osman, C.R. Reeves and G.D. Smith (1996) *Modern Heuristic Search Methods*, John Wiley.
- Ribeiro, R. and H.R. Lourenço (2001) "A Multi-Objective Model for a Multi-Period Distribution Management Problem," *Economic Working Papers Series*, Universitat Pompeu Fabra, Spain.
- Robert, B.H and E.L. Nichols (1999) *Introduction to Supply Chain Management*, 1/e, Prentice Hall.
- Simchi-Levi, D., P. Kaminsky and E. Simchi-Levi (2000) *Designing and Managing the Supply Chain*, McGraw-Hill.
- Skjoett-Larsen, T. (2000) "European Logistics beyond 2000," *International Journal of Physical Distribution and Logistics Management* 30(5):377–387.
- Slats, P.A., B. Bhola, J.J.M. Evers and G. Dijkhuizen (1995) Logistics Chain Modelling, *European Journal of Operational Research*, 87:1–20.
- Stevenson W.J. (1999) *Production/Operations Management*, 6th ed., Irwin, McGraw-Hill.
- Tayur, S., R. Ganeshan and M. Magazine (1998) Quantitative Models for Supply Chain Management. Kluwer Academic Publishers.
- Voß S., S. Martello, I.H. Osman and C. Roucairol (1998) *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*. Kluwer Academic Publishers.

Chapter 16

ON THE INTEGRATION OF METAHEURISTIC STRATEGIES IN CONSTRAINT PROGRAMMING

Mauro Dell'Amico¹ and Andrea Lodi²

¹*DISMI, University of Modena and Reggio Emilia Viale A. Allegri 13, 42100 Reggio Emilia, Italy. dellamico@unimore.it;*

²*DEIS, University of Bologna Viale Risorgimento 2, 40136 Bologna, Italy. alodi@deis.unibo.it*

Abstract In a recent paper, Focacci, Laburthe and Lodi (2002) surveyed the integration between Local Search and Constraint Programming which seems to be suitable to address real-world combinatorial optimization problems. In this paper, we focus on the integration of the machinery developed in the Tabu Search context into incomplete global search algorithms based on CP. The main issue is to re-interpret the techniques developed within Tabu Search for complete solutions so as to apply them to internal nodes of a tree search, i.e., to partial solutions.

Keywords: Local Search, Constraint Programming, Tabu Search, Global Optimization, Metaheuristics

1. Introduction

Local Search (LS) methods are based on the simple and general idea of: (i) starting from a feasible solution, say s ; (ii) defining a *neighborhood* of such a solution, say $\mathcal{N}(s)$, as the set of solutions which can be reached from s through a *move* (a simple manipulation of s whose effect is the transition to another solution $s' \in \mathcal{N}(s)$), and finally; (iii) exploring this neighborhood $\mathcal{N}(s)$ in order to find another solution, say s^* , which is better than s with respect to the objective function. If such a s^* exists, the process is iterated by using s^* as starting solution. Otherwise (no solution better than s exists in the neighborhood), s is a local optimum and several possible escape mechanisms of *metaheuristic* flavor can be applied (see, Aarts and Lenstra 1997, for a complete treatment of LS and metaheuristic algorithms).

Constraint Programming (CP) is a programming paradigm exploiting Constraint Satisfaction techniques (see, Mackworth, 1977), and in the following we

restrict our attention to CP on *Finite Domains* (CP(FD)) which is the case of all constraint tools for discrete optimization. In the next few paragraphs we briefly discuss the CP terminology we will use in the paper. The reader is referred to Marriott and Stuckey (1998) for a textbook on the subject or to Focacci et al. (2001) for a basic introduction.

CP(FD) models combinatorial optimization problems with a set of variables in the domain of integers, and a set of constraints on these variables. Constraints are both mathematical and *symbolic*, where symbolic (or global) means that they model well-defined subparts of the overall problem. The most classic example of symbolic constraints is the `all_different`(X_1, \dots, X_n) constraint which imposes that variables X_1, \dots, X_n must assume different values in a feasible solution.

To each constraint is associated a *propagation* algorithm aimed at deleting from variable domains the values that cannot lead to feasible solutions. Constraints interact through shared variables, i.e., as soon as a constraint has been propagated (no more values can be removed), and at least a value has been eliminated from the domain of a variable, say X , then the propagation algorithms of all the other constraints involving X are triggered (see again Mackworth, 1977).

Since in the general case a full propagation of the constraints is as difficult as the problem itself, propagation algorithms are usually incomplete in the sense that they are possibly stopped with some inconsistent values in the domains. Therefore, a propagation phase is generally followed by a *search* one in which the current problem is partitioned into smaller (easier) subproblems, e.g., by instantiating a variable to a feasible value in its domain. An improvement of the incumbent solution during the search resorts to the addition of a new constraint stating that further solutions should have a better value. Thus, CP optimize the objective function by solving a sequence of feasibility problems that improve the solution value.

In a recent paper, Focacci, Laburthe and Lodi (2002) surveyed the integration between LS and CP which seems to be suitable to address real-world combinatorial optimization problems. In real-world applications one can usually recognize a ‘core’ problem whose structure is most of the times well known in the literature, plus a set of complex constraints, often called ‘side’ constraints. These side constraints are in general very much oriented to the application at hand, and lead the overall problem to become much more difficult to solve. Indeed, the union between the core problem and the side constraints usually leads to (Mixed) Integer linear Programs (MIPs) whose size is huge and such that the corresponding models present relevant gaps, i.e., poor linear programming relaxations. In other words, the addition of heterogeneous side constraints typically ‘destroy’ the pure structure of the core problems.

Although the big improvements of the general-purpose MIP solvers in the last fifteen years (see Bixby, 2002) these problems often remain out of practice with respect to optimality, and heuristic and metaheuristic algorithms turn out to be a conceivable choice.

Moreover, the side constraints of real-world applications change frequently, thus it seems to be unlikely to spend too much time in studying the underline structure of the problems (e.g., through a deep polyhedral analysis) since either the addition of ‘new’ constraints or the update of ‘old’ ones can change this structure in the short term.

This is one of the main reasons for which methods based on the integration of LS and CP are interesting: CP tools provide *modeling* facilities and the *solving* technique is based on ‘local’ reasoning (the propagation algorithms encapsulated into the symbolic constraints), thus it is robust with respect to the addition/update of constraints. Specifically, propagation algorithms are usually designed to reduce the solution space through *logical* implications which are local to a specific part of the overall problem, and these logical reasoning typically remains valid when another part of the problem changes. The same does not hold when the solving technique is based on the polyhedral analysis since a polyhedron is defined by the overall set of constraints, thus the reasoning on it is *global*.

In Focacci, Laburthe and Lodi (2002), two main directions for LS and CP integration are investigated. From one side, the possibility of using constraint-reasoning and techniques based on CP within a classical LS algorithm so as to produce a *constrained local search* framework. From the other side, the transformation of the classical global search algorithm provided by CP tools into an *incomplete global search* one through the use of LS techniques. This second direction seems to be very powerful and promising, typically leading to flexible algorithms whose great advantage is moving through a decision tree. Indeed, the classical exploration methods of a decision tree must be rigid enough to guarantee optimality. However, in a heuristic context, relaxing this rigidity but still remaining inside the tree allows using all the CP machinery in terms of problem reductions and efficiency by obviously accelerating the process. Moreover, as shown in Focacci, Laburthe and Lodi (2002), both classical and recent LS techniques can be effectively incorporated in order to either limit the size of the portion of the tree *systematically* generated/explored or iteratively changing the exploration direction, thus leading to ‘jumps’ in the tree.

In this paper, we concentrate on the second direction discussed in Focacci, Laburthe and Lodi (2002), and in particular, we focus on the integration of the machinery developed in the *Tabu Search* (TS) context, and in particular by Glover and Laguna (1997) in the TS book, into incomplete global search algorithms based on CP. More precisely, in the TS framework one can logically recognize two ingredients: from one side, the basic idea of escaping from a

local optimum by allowing worsening moves and by using memory so as to avoid cycling, while on the other side, the wide set of ‘strategic’ techniques which are used to drive the search through the solution space or accelerate the neighborhood exploration.

The basic idea of TS has been already considered and extensively used in the CP literature (see, e.g., De Backer et al. 2000), while in this paper, we are interested in some of the strategic techniques developed by Glover and Laguna (1997) in the chapter 3 of the TS book, with the aim of integrating them into incomplete global search algorithms based on CP.

The main issue of such an integration is the following. The techniques originating in TS context usually have a *complete* solution at hand (most of the times feasible, sometimes infeasible) and iteratively improve it (where ‘improving’ includes the fact of moving an infeasible solution toward feasibility). Within a global search algorithm, instead, we want to apply LS techniques (and in particular TS techniques) to *partial* solutions, i.e., to internal nodes of the branching decision tree. The aim of the paper is to re-interpret some of the techniques developed in the former case to the latter one. The treatment is neither methodologically exhaustive nor provides a complete description of implementation issues, but it represents a kick-off for the research into this promising area.

It is worth mentioning that incomplete global search algorithms do not necessarily require an implementation by means of CP tools. There are several effective ways of devising such kind of algorithms in the Integer Programming context (e.g., GRASP by Feo and Resende 1995, recovering beam search by Della Croce and T'kindt, 2002).

The paper is organized as follows. In Section 2 we discuss several TS techniques that can be used for an ‘*a priori*’ limiting of the size of the search tree, while in Section 3 the limiting based on the knowledge of the best solution(s) found so far is considered. In Section 4 we discuss techniques that perform a logical restructuring of the tree, i.e., ‘jump’ from an internal node to another, possibly in a completely different part of the tree. Finally, in Section 5 some conclusions are drawn and interesting areas for future research are pointed out.

2. Limiting the size of the tree ‘*a priori*’

The aim of this section is to show how the size of the decision tree can be limited by using *candidate list strategies*, i.e., by an ‘*a priori*’ analysis of the potential effect of either one decision or a set of them. In the general TS context, this means that not all the possible ‘moves’ are taken into account where this filtering is based on several criteria (see, Glover and Laguna 1997, chapter 3). We re-interpret these criteria by keeping their flavor and applying them within the decisions of a branching tree.

We consider the completely general case such that the CP branching (or search) step consists of assigning a specific *value*, say v_j , to a *variable*, say X_i , such that v_j belongs to the domain of X_i : $v_j \in D(X_i)$. We denote this variable/value decision with the pair (X_i, v_j) . This means that at each node of the decision tree the set of decisions is composed by all the variables which have not yet been assigned and all the values in their domain. This is the most general and easiest-to-handle situation in CP context and it is often the case in practice.

The application of TS techniques to partial solutions instead of complete ones can be easily seen in terms of the relation between moves and decisions. Specifically, by considering only a subset of the possible moves in TS context we can limit the size of the neighborhood explored, while within a tree search this corresponds to taking into account only a subset of decisions, thus limiting the tree itself. Obviously, the selection of a subset of the decisions is a crucial task and we will often assume in the following that the selection method, heuristic in nature, can be rather time-consuming since the accuracy is important. We call `evaluate()` a general procedure that ranks with some criteria **all** the possible decisions.

Our re-interpretation of candidate list strategies in this context leads to the following four different techniques which turn out to be four different ‘implementations’ of procedure `evaluate()`.

Aspiration plus. This technique is based on the definition of a *threshold* on the improvement of a decision: an evaluation procedure computes some ‘expected improvement’ and the first decision for which this value is greater or equal to the threshold is considered for possible selection. In order to be more accurate the evaluation continues for the next *plus* decisions. All the ‘good’ decisions, i.e., the ones whose improvement is greater or equal to the threshold are stored in a buffer associated with the node, and the one with the highest improvement is taken. When a backtracking to the current node occurs the evaluation procedure starts from the first decision which has not yet been evaluated and iterate the above procedure by also re-evaluating the decision stored in the node-buffer, whose expected improvement can be changed¹. This technique does not require to evaluate all the possible variable/value assignments. Its effectiveness can vary with the ordering in which the decisions are considered. A problem-dependent ordering, if any, can be used to sort the decisions accordingly before calling our evaluating procedure.

Formally, a specific evaluation procedure, called `evaluate_plus()`, is executed at each point a new decision must be taken. A scheme of the procedure

¹Some of them cannot be feasible anymore after propagation.

is as follows:

```
procedure evaluate_plus()
  if ('node-buffer is not empty') then
    re-evaluate the decisions in the node-buffer
  endif
  find next decision with improvement  $\geq$  threshold and store it
  evaluate the next set P of 'plus' decisions
  store the decisions of P with improvement  $\geq$  threshold
  returnBest( $X_i, v_j$ )
```

A stopping criterion can be the complete evaluation of the overall set of pairs variable/value together with the emptying of the buffer. In this case, several possible recovering/restart criteria can be used in order to continue the search, e.g., the threshold can be changed (namely, reduced) in order to enlarge the set of decisions classified as 'good'.

Elite candidate list. As mentioned before, we assume that the evaluation of the decisions is time-consuming, and in this case we do not want to run `evaluate()` at each node of the tree. Thus, the evaluation procedure is run as soon as a *global-buffer* of likely decisions is empty. More specifically, the evaluation procedure ranks in this case all the variable/value assignments, then the k best decisions involving different variables are stored in the global-buffer. From this point, each time a new decision must be taken, the best *feasible* (to be discussed later) one in the global-buffer is selected and performed until the buffer is empty. A decision (X_i, v_j) is considered still feasible after that some others in the global-buffer have been taken if and only if $v_j \in D(X_i)$, i.e., value v_j still belongs to the domain of variable X_i after the previous decisions and the corresponding propagations.

Formally, the specific evaluation procedure, called `evaluate_elite()`, can be outlined as follows:

```
procedure evaluate_elite()
  if ('global-buffer is empty') then
    execute procedure evaluate()
    store the  $k$  best decisions involving different variables
  endif
  returnBest( $X_i, v_j$ )
```

Since in the case the global-buffer is empty a new buffer can be filled up, the above implementation is *complete*, i.e., every possible branch is considered. Thus, the interest of this technique would be to obtain a reduction in terms of computational time since procedure `evaluate()` is executed only a limited

number of times (depending on the value of k). However, it is easy to see that an incomplete version can be devised if the number of times each node is allowed to fill up the buffer is kept limited, say p times. In this way, together with the reduction in the computational effort of evaluating the decisions, some of them will never be considered according to the rank.

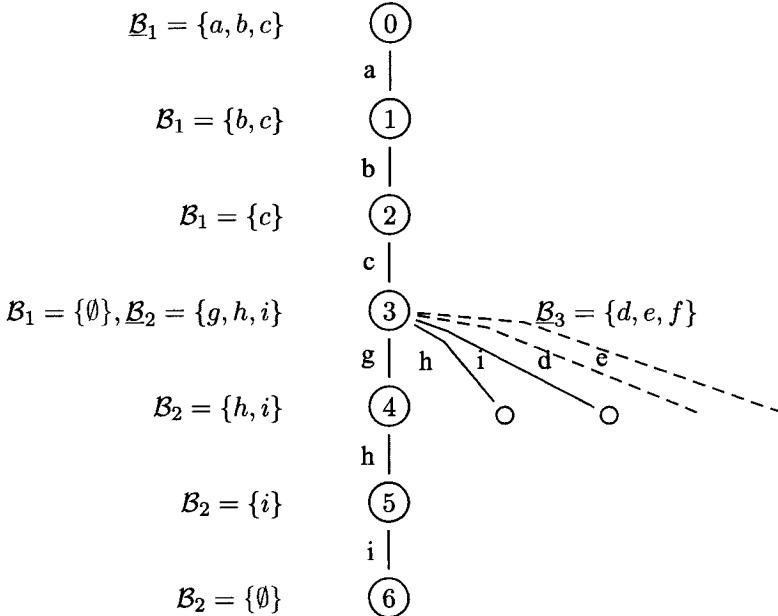


Figure 16.1. decision tree generated with `evaluate_elite()`: computed buffers are underlined

In Figure 16.1 a buffer B_1 is computed at node 0 and it is used to generate the descending nodes 1 and 2. When B_1 is emptied B_2 is computed and used to generate nodes 4, 5 and 6. When a backtracking to node 3 occurs the remaining decisions of B_2 are used to generate the next sons, then a new buffer B_3 is computed and used to generate new sons. One can check that from decision node 3 (so as from all other nodes) the descending sons correspond to all possible decisions and the method is indeed complete.

Successive filter strategy. In tree search algorithms sometimes not all the variables have the same importance. Often, in the design of ad-hoc branching schemes, one can prefer branching before on variables belonging to a specific subset since they are considered more crucial.

With a similar flavor, one can reduce the size of the search tree, thus transforming the overall algorithm into an incomplete one, by selecting at each node

a subset of the possible variables, and for each of them only a subset of the values in their domain. Without loss of generality we consider a partition of the set of variables in two subsets, say A and B . At each upper node of the tree a subset of the A variables is selected and a pruning on the possible values of each of them is performed with some criteria. This means that in the consequent subtree some of the decisions are missing, thus the tree is smaller and no guarantee of optimality is given. The method is iterated until all the variables in A has been assigned, and (possibly) repeated at the lower nodes of the tree for the variables in B . This a priori filtering does not only reduce the size of the tree but also allows to only evaluate a subset of the decisions.

Formally, the specific evaluation procedure, called `evaluate_filter()`, can be outlined as follows:

```
procedure evaluate_filter()
    select a subset of the variables  $I$ 
    for each variable  $X_i \in I$ 
        select a subset of the values  $J$ 
        execute evaluate() on the subset defined by the pair  $(I, J)$ 
    return Best( $X_i, v_j$ )
```

It is easy to see that this method is not really concerned with multiple levels of variables. However, the idea comes to mind mainly when different sets of variables assume quite a different meaning, thus it seems to be easier to perform an a priory (obviously, heuristic) pruning.

Sequential fan candidate list. Another classical way of transforming a complete tree search into an incomplete one is the so-called *truncated* branch-and-bound which is very well known in the Operations Research (OR) community. There are obviously many ways to perform such a truncation among which the easiest one is imposing a time limit, while more sophisticated ones impose for example a limit in backtracking steps².

A possible implementation of the idea requires to consider the tree by levels in order to keep the number of nodes active in each level limited to a given value k . This means that at the root node, level 0, all the decisions, i.e., pairs variable/value, are evaluated, and the best k decisions (according to the evaluation criterion) define k sons which are the only active at level 1. The process is then iterated by considering at the same time the entire set of decisions that can be taken in the k active nodes of level 1, selecting the best k ones, generating the k nodes of level 2, and so on.

²Actually, with an extension of the terminology almost all the methods we are concerned with in this section and in the next one fall down into the ‘truncated branch-and-bound’ category.

Formally, the specific evaluation procedure, called `evaluate_fan()`, is executed at each node of the decision tree and can be outlined as follows:

```

procedure evaluate_fan(node, level)
  if ('level-buffer(level + 1) was never created') then
    for each active node in level
      execute procedure evaluate()
      update the (level + 1)-buffer containing the k best decisions
    endfor
  endif
  returnFirst( ( $X_i, v_j$ ) descending from node, if any )

```

3. Limiting the size of the tree ‘based on history’

In the previous section we have considered methods for limiting the search tree ‘*a priori*’. In other words, one does **not** use explicitly any information on the paths explored during the search for driving the algorithm through the tree **but** only implicitly in the evaluation of the single decisions. This means that the reduction of the tree is imposed in a rigid way from the beginning. Conversely, it is conceivable that the reduction could be imposed dynamically by exploiting the ‘history’ of the search which in particular translates in a more flexible exploration of the tree.

In this section we consider a special way of exploiting the ‘history’. Specifically, the pieces of information we exploit are the *good* solutions encountered during the search and the idea is to use them within a *path relinking* framework.

The idea of path relinking is to search the solution space through paths connecting good solutions. These paths can be more or less seen as greedy explorations of the subspace induced by the set of variables assuming different values in the set of good solutions. In other words, the feeling is that good solutions generally share important parts, and the paths connecting them are relevant optimization directions. Following these directions corresponds to perform a local search in the subspace.

The same concept can be immediately applied within a tree search framework. During the search good solutions are stored and at a given level of the branching tree, instead of an exhaustive search, partial solutions are completed through the generation of paths which take into account the stored solutions. In Figure 16.2 the search is exhaustive from the root to level *L*, while partial solutions are completed according to different techniques in the remaining levels. Some ways of generating these paths are the following.

- 1 In order to keep the size of the tree small enough and the completion of the partial solutions fast, a possibility is to only generate for each partial solution the path to the best solution found so far, i.e., we instantiate

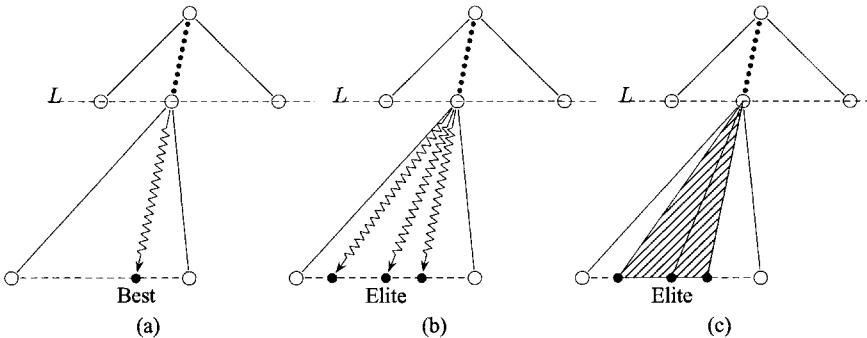


Figure 16.2. possible completions of a partial solution

the free variables with the values they have in the best solution (see Fig. 16.2(a)). This method has two strong drawbacks, indeed the resulting solution could be: (a) not feasible; (b) incomplete (due to the fact that the free variables were assigned no specific values in the best solution). Case (a) determines a failure, while we can solve case (b) by applying a heuristic designed to complete a partial solution, or better, continuing the search with CP from the current partial solution.

- 2 A more complete but still straightforward way is to generate a greedy path to each solution in the set of stored (elite) ones (see Fig. 16.2(b)). This technique works exactly as the previous one, but tests several solutions instead of a single one.
- 3 In a more sophisticated way, one can generate a set of paths obtained as combination of the elite solutions. This is a variant of case 2 in which we extend the set of the elite solutions using them to generate additional ones. The method used to combine the solutions and its effectiveness is very problem dependent.
- 4 In our view, the most interesting way of completing each partial solution is to explore a subtree induced by the set of elite solutions, namely the subtree defined on the not instantiated variables, with domains restricted to the only values appearing in the elite solutions (see Fig. 16.2(c)). The subtree exploration can be speed up through *discrepancy-based search* (xDS) (see, e.g., Harvey 1995, Harvey and Ginsberg 1995, Walsh 1997). According to the xDS framework, a discrepancy in the search is introduced each time the algorithm takes a decision which is different from the best one suggested by an evaluation heuristic. The tree search can be explored so as to keep the total number of discrepancies limited, say h .

For example, if the heuristic always suggests the decisions in the greedy path to the best solution found so far and h is set to 0, then we get again case 1 above.

4. Restructuring the tree

The methods introduced in the previous sections use reductions of the number of nodes visited on the search tree to speed up the search, thus transforming a global search algorithm into an incomplete one. Roughly speaking, these methods operate by smart cutting some branches of the tree. The main drawback is that a bad decision taken at a certain node j is fixed for all the descending nodes, i.e., a cut of a branch containing good solutions cannot be recovered.

In this section, instead, we propose the use of techniques from TS, that hybridize the CP method by means of *soft decisions* that can be reverted or removed in a descendent node. This gives an extreme flexibility to the search that is allowed to move from a father to a son node either by adding a decision, or removing some decision previously taken if it results to be an obstacle for reaching good solutions. The net effect of such a deletion can be seen as a "jump" from a node j to a new node not descendent from j . In other words, the scheme we are proposing is an enhancement of an incomplete search algorithm obtained by applying jumps to recover from bad branch cutting. A jump allows to enter into a subtree that was previously eliminated by some cut, but that now is recognized to contain promising solutions. This is a very appealing feature that may lead to a significant improvement of the incomplete search methods based only on tree reduction techniques. However, this powerful tool needs to be managed carefully, since it could result in dangerous effects. Indeed, using arbitrarily the jumps we could lose completely the control on the search and we could experience the opposite of our goal: visiting a very restricted part of the tree, in the limit, the same solutions in a cyclic manner.

The key ideas from TS that we use are the *principle of congenial structures* and the method of *logical restructuring* (see Glover and Laguna 1997, sections 5.6 and 3.4 for a complete description).

The principle of congenial structures is based on the fact that in many combinatorial problems there are particular types of solution structures, or partial solutions, that provide a great accessibility to solutions of highest quality. As possible examples consider: a Bin Packing Problem and a subset of items that well fit into some bins; a scheduling problem with sequence dependent setup times and a subset of jobs organized in a subsequence with null or very small setup time; a Vehicle Routing Problem and a feasible route that almost fills the vehicle capacity and serves customers located in a restricted area.

The logical restructuring is a way to meet the combined concerns of quality and influence of a given solution. The goal is to find how influence (structural,

local or global) can discover improved paths to high quality solutions. Logical restructuring is an implicit mechanism in other general techniques as strategic oscillation, path relinking and ejection chains.

The proposed method consists of a search on a CP decision tree divided into two regions: the strong-decision's area and the soft-decision's area. The first one corresponds to the usual CP decision tree, whereas the latter is the region in which we allow to jump from a part of the tree to another.

The simplest way to define the two regions is to fix a *threshold* level separating them, but more sophisticated methods are possible, as *multi-thresholds* (different thresholds for different branches) and *dynamic-thresholds* (thresholds updated according to the search evolution). Here, we only consider a simple fixed-threshold method operating as an usual CP method from the root to the threshold level, and allowing soft decisions in the remaining levels.

In the soft-decision's region the system must be free to look for good solutions, using the two guiding principles above, without being restricted by the problem constraints. To do this we propose to consider a relaxation of some constraints, thus allowing the construction of infeasible solutions that are later driven to feasible ones by changing or reverting decisions previously taken. To see how this method induces a jump in the original tree let us suppose that we have an infeasible solution by taking two sets of decisions, namely S_1 and S_2 . Further suppose that $S_1 \cap S_2 \neq \emptyset$, each set alone determines a (partial) feasible solution, and exists a set $S \subset (S_1 \cup S_2)$ such that the decisions in $(S_1 \cup S_2) \setminus S$ define an optimal solution. When we build an infeasible solution by taking the decisions in $S_1 \cup S_2$ we can re-obtain a feasible solution by removing the decisions in S . In a standard CP tree we have a decision-path R_1 that includes all the decisions in S_1 , a second path P_2 including the decisions in S_2 and the optimal path P_3 including $(S_1 \cup S_2) \setminus S$. Our method gives the algorithm more alternative ways to reach the optimal solution: (i) a descent along R_1 followed by a jump to path P_3 obtained removing the decisions in $S_1 \cap S$; (ii) a descent along P_2 followed by a jump to path P_3 ; (iii) a descent along a new path made by the concatenation of paths R_1 and P_2 followed by the removal of the decisions in S ; etc.

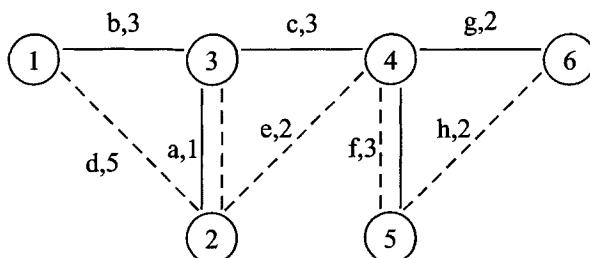


Figure 16.3. a spanning tree

In Figure 16.3 we give an example where a min-cost spanning tree is required. We have $S_1 = \{a, b, c, f, g\}$ (continuous lines, cost = 12), $S_2 = \{a, d, e, f, h\}$ (dashed lines, cost = 13) and $S = \{c, d, f\}$. Set $S_1 \cup S_2 \setminus S = \{a, b, e, g, h\}$ with cost 10 is the optimum tree.

Soft decisions can be seen as constraint relaxation and can be implemented in different ways. Here we propose two possible implementations: 1) apply the constraints as usual and relax them when necessary to perform the jump; 2) add the constraint immediately to the model, but not propagate it. A ‘constraint check’ is then performed at a specific level, thus a jump to repair a possible failure is adopted.

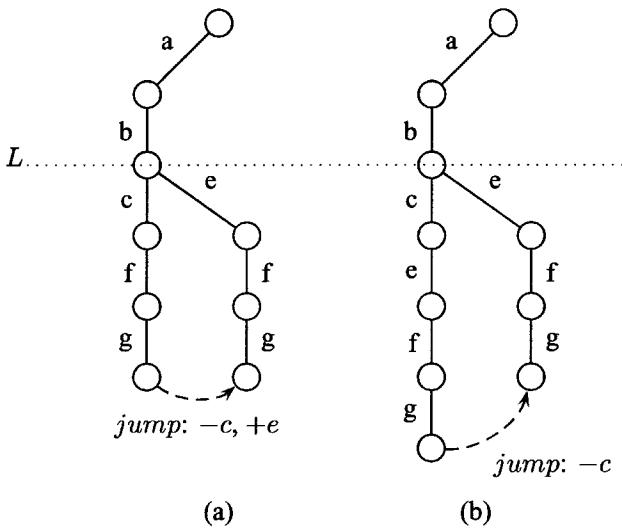


Figure 16.4. managing the soft decisions

Figure 16.4 reports two possible partial decision trees for the min-cost tree problem associated with the graph of Figure 16.3. At each node we may select any variable (edge). We use a lexicographical ordering (alphabetical, in fact) for the choice of the next decision to be applied.

Figure 16.4(a) shows a possible tree obtained by applying technique (1) above. When we reach a leaf we consider soft all the decisions under level L . Hence, we could remove the decision of selecting edge c , thus jumping to the partial solution a, b, f, g , or, better, we could examine this solution and modify it with a local reasoning, so jumping to the complete solution a, b, e, f, g , as shown in the figure.

An example of application of technique (2) is given in figure 16.4(b). In this case when we are below level L we do not perform the propagation, so we construct the redundant solution a, b, c, e, f, g . Note that edge d is not included

in this set since it creates a cycle with the two edges a, b selected in levels higher than L . Performing the constraint check at this level we identify cycle a, c, e , i.e., a violated constraint, and we can recover it by removing the heaviest edge instantiated in the soft decisions region: edge c .

The second technique seems to be the simplest one for an immediate use in CP, although it partially runs contrary to the CP paradigm by temporarily avoiding propagation.

5. Conclusions

Despite the integration of LS ideas in CP has been explored in recent years, no work has been devoted on integrating CP with techniques originating in TS context. CP methodology is based on solution construction techniques, while TS mainly operates by iteratively modifying a complete solution (possibly infeasible). The main issue of the paper was to re-interpret some of the strategic TS techniques and to adapt them to be applied to partial solutions within a CP tree search. The treatment is neither methodologically exhaustive nor provides a complete description of implementation issues, but it represents a kick-off for the research into this promising area.

Concerning further research directions, a lot must be done on the methodological side, and, in particular, hybrid methods allowing jumps on the tree, such as *recovering beam search* (Della Croce and T'kindt 2002), *incremental local optimization* (Caseau and Laburthe 1999), and *incomplete dynamic backtracking* (Prestwich 2002), seem to indicate a very promising area. However, in order to obtain effective algorithms for practical problems the implementation issues play a crucial role, and we believe that they require a very deep investigation.

Acknowledgments

The authors greatly thank Filippo Focacci, César Rego and Andrea Roli for a lot of interesting discussions on the topic. Thanks are also due to an anonymous referee for helpful comments. This work has been partially supported by MIUR and CNR, Italy.

References

- Aarts, E. and J.K. Lenstra (1997) *Local Search in Combinatorial Optimization*, John Wiley & Sons, Chichester.
- Bixby, R.E. (2002) "A new Generation of MIP Codes", Lecture presented at *Integer Programming Conference in Honor of Egon Balas*, Pittsburgh, June 3-5.
- Caseau, Y. and F. Laburthe (1999) "Heuristics for Large Constrained Routing Problems", *Journal of Heuristics*, 5: 281–303.

- De Backer, B., V. Furnon, P. Shaw, P. Kilby and P. Prosser (2000) "Solving Vehicle Routing Problems using Constraint Programming and Meta-Heuristics", *Journal of Heuristics*, 6: 481–500.
- Della Croce, F. and V. T'kindt (2000) "A Recovering Beam Search Algorithm for the One-Machine Dynamic Total Completion Time Scheduling Problem", *Journal of the Operational Research Society*, 53:1275–1280.
- Feo, T. and M. Resende (1995) "Greedy Randomized Adaptive Search Procedures", *Journal of Global Optimization*, 6:109–133.
- Focacci, F., F. Laburthe and A. Lodi (2002) "Local Search and Constraint Programming", in F. Glover, G. Kochenberger, Eds., *Handbook of Meta-Heuristics*, Kluwer Academic Publishers.
- Focacci, F., A. Lodi, M. Milano and D. Vigo (2001) "An Introduction to Constraint Programming", *Ricerca Operativa*, 91:5–20.
- Glover, F. and M. Laguna (1997) *Tabu Search*, Kluwer Academic Publishers, Boston.
- Harvey, W. (1995) *Nonsystematic Backtracking Search*, PhD thesis, Stanford University.
- Harvey, W. and M. Ginsberg (1995) "Limited Discrepancy Search", in *Proceedings of the 14th IJCAI*, Morgan Kaufmann, 607–615.
- Mackworth, A. (1997) "Consistency in Networks of Relations", *Artificial Intelligence*, 8:99–118.
- Marriott, K. and P. Stuckey (1998) *Programming with Constraints*, The MIT Press.
- Prestwich, S. (2002) "Combining the Scalability of Local Search with the Pruning Techniques of Systematic Search", *Annals of Operations Research*, 115:51–72.
- Walsh, T. (1997) "Depth-Bounded Discrepancy Search", in *Proceedings of the 15th IJCAI*, Morgan Kaufmann.

Chapter 17

GENERAL PURPOSE METRICS FOR SOLUTION VARIETY

David L. Woodruff

Graduate School of Management, University of California at Davis. DLWoodruff@UCDavis.edu

Abstract When using optimization in the context of a decision support system, a decision maker would often be more interested in seeing a diverse collection of good solutions rather than the one solution that is in some mathematical sense the optimal solution to an approximate problem with imprecise data. A recent paper (Glover, Løkketangen and Woodruff, 2000) describes a method for generating diverse solutions using scatter search. However, very little research has been done on general purpose methods to characterize solution variety. This paper describes and explains desirable characteristics for general purpose methods for characterizing diversity. Field research serves to demonstrate and compare various methods. We provide some fundamental research concerning methods that are fully general and do not rely on any knowledge of the problem.

Keywords: Decision Support, Optimization, Variety Metrics, Chunking

1. Introduction

The word “optimization” is cropping up in a wide variety of settings, especially advertisements for decision support and enterprise resources planning software. The problems being addressed are extremely complex and consequently the models being optimized can be at best an approximation of the goals of the organization and its stakeholders. In such an environment, a decision maker would often be more interested in seeing a diverse collection of good solutions rather than the one solution that is in some mathematical sense the optimal solution to an approximate problem with imprecise data.

The need for variety metrics also arises in the process of designing population based algorithms such as scatter search (Glover 1995, Glover 1997, Glover and Laguna, 1997). Solutions that are varied are required so various methods have been proposed that can generate a diverse set of solutions. A metric in the space

of solutions would be useful operationally and methods for putting an order on the diversity of sets of solutions would be useful in research to compare population generators.

However, very little research has been done on methods to characterize solution diversity. For example, if the decision maker has been shown a few solutions and there are a few more good solutions that could be shown, the next solution to be displayed must be selected based on some criteria. The solution with the best objective function may well be extremely similar to one that has already been displayed and hence would waste the time of the decisions maker. It would be better to find a solution that would diversify the set that has been displayed. An extensive study of issues related to diversification and pool maintenance is provided by another chapter in this volume (Greistorfer and Voß, 2004).

A recent paper (Glover, Løkketangen and Woodruff, 2000) describes a method for generating diverse solutions using scatter search and uses a dimension reduction technique known as *chunking* Woodruff (1998) to create a metric to compare their method with branch and bound. Our interest here is in creation of techniques that can be widely used by developers of decision support systems (DSS) (see, e.g., Hoch and Schkade, 1996, Sharda, Barr and McDonnell, 1988) and in population based metaheuristics such as scatter search. Toward that end, we list characteristics that such an addition to the DSS implementer's toolkit should have. Further, we propose methods that have various of these properties. We then describe samples collected via field research for comparison between the Euclidean metric and a more sophisticated general-purpose metric that relies on principal components as a generalization of chunking application.

If one has a particular problem to solve with a particular data instance, then it would often be possible for a domain expert to compare a few solutions and rank their pairwise distances. It might even be possible to construct an expert system to do this. This could be worthwhile in some settings, but our interest here is developing general methods that can be applied across a broad range of problems and problem instances. Our work is based on the notion that there is structure in good solutions that can be discovered at some level of abstraction and used to construct a metric.

Our research is quite different from sensitivity analysis. For optimization problems there is a rich literature concerning the sensitivity of the optimal solution to changes in the problem data (Greenberg 1998, Wagner 1995). Our work augments this by providing means to characterize the variety provided by multiple solutions, but not necessarily the optimal solution.

There is somewhat stronger connection between this work and previous work in the area of multiple criteria optimization (MCO); among the many works, see, e.g., (Steuer 1986, Zeleny 1982). Readers familiar with the multi-criteria optimization literature will notice this paper presents ideas that are potentially

competitive with the basic idea of MCO, but potentially complimentary with filtering, which is often discussed in the context of MCO. The potential competition is strictly theoretical. If one could somehow capture all of the various objectives of all decision makers and stakeholders for a problem and quantify them all, then one could use the technology of MCO to map out all of the undominated solutions on the so-called efficient frontier. In practice, decision makers might be interested in solutions close to the efficient frontier if they are different enough from those on the frontier. The problem of identifying solutions that “different enough” crops up. In this paper we propose metrics that take covariances into account, which goes beyond the proposals of an L_k metric with univariate scaling that have been proposed.

The next section outlines some of the characteristics that would be desirable for a metric on the space of solutions. In §3 we describe and justify a method of obtaining a metric. The method is further motivated in §4, where we give two examples that are the result of field research. These examples serve both to motivate our metric and to provide a tractable testbed for other general purpose metrics. The paper closes with some concluding remarks and directions for further research.

2. Characterizing Solution Diversity

Although the objective function used for optimization might not exactly match the objectives of the organization, it is designed expressly for the purpose of giving a rough assessment the relative quality of solutions. Hence, if the user of a DSS asked to see another solution that was good and different from the solutions seen so far, it would be reasonable to use the objective function to quantify the notion of “good.” We now turn to the question of characterizing what is meant by “different.”

To discuss general methods, we assume that we have available a set of solution vectors that we will refer to as \mathcal{A} . The composition of the set would depend on the particular application. It might be a set of good solutions, the set of solutions shown to the user so far, or some other population of solutions for which diversity is an issue.

To be part of a general DSS toolkit, the following would be useful for diversity measurement:

- 1 The methods should not be based on the presence of *training data*, which is representative set of solution vectors for which the data for which the user has indicated the correct metric. In many applications, there is no way the user would be willing to spend enough time with the instance to generate the training data, and if even if they did, the decision would have to be made before they finished, rendering the training set worthless. This is not to say that there are not situations where training data can be used,

but a fully general-purpose method that does not assume the existence of training data is also needed.

- 2 The methods should be based on a true metric, which means that for metric d and all solution vectors x and y ,

$$\begin{aligned} d(x, x) &= 0, \\ d(x, y) &> 0 \text{ if } x \neq y, \\ d(x, y) &= d(y, x), \text{ and} \\ d(x, y) &\leq d(x, z) + d(y, z). \end{aligned}$$

Confusing inconsistencies can crop up in a decision support system that relies on a method without these properties. Although there has been some psychometric research (Love 2000) questioning the importance of symmetry requirement and the final requirement, which is often called the *triangle inequality*.

- 3 It should be possible to characterize both tendency to diversify a set of solutions and the distance between two points. This feature comes automatically when a true metric is used because a metric induces a topology, which enables computation of volumes. A volume corresponds naturally to notions of size such as diversity. Hence, we can say that a solution that would increase the volume spanned by a set of solutions would increase its diversity. The amount of volume increase can even be used to quantify the effect. If a true metric is not used, this feature can be difficult to obtain. For example, an expert system can be created that compares two solutions but this will not induce a topology unless it results in a true metric.
- 4 Evidence concerning the structure of the solutions should affect the metric. For example, correlation effects should be considered. Examine Figure 17.1. Suppose all of the points shown have the equal objective function values. The point labeled 'o' would reasonably be considered to be highly unusual even though the value in each of the two dimensions considered in isolation is not at all unusual. The presence of a good solution with unusual correlations is typically very interesting to decision makers.
- 5 The methods should be insensitive to changes in scale and it would be better if they are affine equivariant, which means that stretching or rotating measurement scales will either not alter the outcome. Column scaling to improve the numerical stability of an optimization problem is an affine transformation as is the standardization of a set of solution vectors to have mean zero and unit variance. As a practical matter it can be very

useful to have methods whose results are not qualitatively altered by such transformations.

We will refer to this as the *criteria list*. Before proceeding, we pause to provide the definition of affine equivariance for two types of functions: vector valued and matrix valued. In both cases the functions take as arguments n column vectors of length p in a matrix \mathcal{A} .

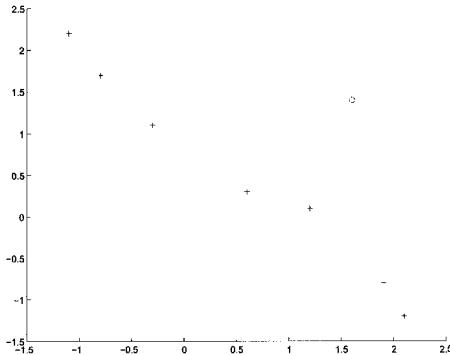


Figure 17.1. A Small Set of Points in Two Dimensions

A vector valued function (e.g., location estimator) $t \in \mathcal{R}^p$ is affine equivariant if and only if for any vector $b \in \mathcal{R}^p$ and any non-singular $p \times p$ matrix B

$$t(B\mathcal{A} + b) = Bt(\mathcal{A}) + b$$

and a matrix function (e.g., covariance estimator) $C \in \text{PDS}(p)$ is affine equivariant if and only if for any vector $b \in \mathcal{R}^p$ and any non-singular $p \times p$ matrix B

$$C(B\mathcal{A} + b) = BC(\mathcal{A})B^T,$$

where $\text{PDS}(p)$ means positive definite and symmetric with dimension p .

We will not always be able to get all of these desirable features. For example, two popular methods of measuring distance between solution vectors are the Euclidean distance (or inner product of the differences) and the Hamming distances, which is a count of the number of vector elements that differ. The Euclidean metric generalizes to the family of L_k metrics, where it is referred to as the L_2 metric. These methods do not meet the desirable criteria in general as noted in the following remark, which is easily verified.

REMARK 1 *Euclidean distances fail on the last two items on the criteria list and in general Hamming distances fail on all but the first.*

It is possible to mitigate scale dependencies by dividing each dimension by its a measure of its univariate dispersion, such as the standard deviation or range. This concept is generalized by the *Mahalanobis* distance.

3. Mahalanobis Distances

A p -vector x can be said to have a squared *Mahalanobis* distance from a p -vector μ under a $p \times p$ positive, definite symmetric matrix S that is given by

$$m^2(\mu, x; S) \equiv (x - \mu)^T S^{-1} (x - \mu)$$

We use the notation $m(\cdot)$ to signify $\sqrt{m^2(\cdot)}$. Note that the Euclidean metric is a special case that uses the identity matrix as S . Mahalanobis distances are scale invariant and take correlations into account if the matrix S is a covariance matrix. An estimate of the covariance matrix for the population from which a sample, \mathcal{A} with a vectors is drawn is

$$\frac{1}{a-1} \sum_{j=1}^h (\mathcal{A}^{(j)} - \bar{\mathcal{A}})(\mathcal{A}^{(j)} - \bar{\mathcal{A}})^T,$$

where $\bar{\mathcal{A}}$ is the usual mean of the set of vectors. The set is indexed here by j .

The motivation for the Mahalanobis distance comes from multivariate probability theory. If a multivariate normal distribution is defined by mean μ and covariance S , then for points v governed by the distribution, the Mahalanobis distances, $m^2(\mu, v; S)$, have a χ^2 distribution (see e.g., Anderson 1984). The use of the covariance estimate to parameterize the metric connects naturally with a scalar measure of the diversity of a set of vectors, which is the determinant of the covariance matrix of the set. The covariance matrix characterizes the dispersion of the set and its determinant measures its volume. The assumption of multivariate normality is not needed to use the covariance determinant to measure the diversity of sets of vectors, although it does provide asymptotic justification.

Figure 17.2 shows the influence of the shape (covariance matrix) of the cluster on the distance calculation. Data points that are located on the same ellipsoid have the same Mahalanobis distance to the mean of the cluster. The Mahalanobis distances are parameterized by the covariance matrix of the data.

Adding points with large Mahalanobis distances will increase the covariance determinant as noted by (distribution free) Remark 2 that adds some rigor to the notion of “increasing the diversity.” The following remark is well known in the statistics literature, so the proof is omitted.

REMARK 2 *When there are multiple candidate points that can be added to a set of points, the one with the greatest Mahalanobis distance under the covariance estimate of the set from the mean of the set maximizes the resulting covariance determinant.*

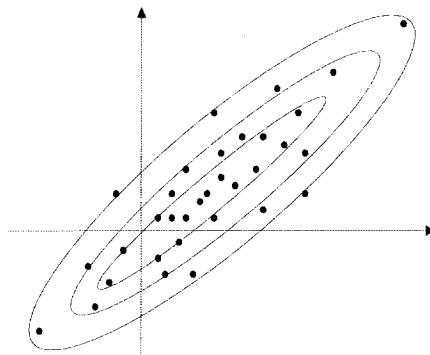


Figure 17.2. Data points located on the same ellipsoid have the same Mahalanobis distance to the center of the data set.

Vector	L.A.	S.F.	Hamburg	AJ8172
$\mathcal{A}^{(1)}$	1	0	0	0.94
$\mathcal{A}^{(2)}$	0	1	0	0.32
$\mathcal{A}^{(3)}$	1	0	1	0.04
$\mathcal{A}^{(4)}$	0	0	1	0.03
$\mathcal{A}^{(5)}$	1	0	1	0.13
Mean,	0.6,	0.2,	0.6,	0.29
Std. Dev.	0.5	0.4	0.5	0.38

Figure 17.3. A Small Example of a Set of Solution Vectors

Note that Euclidean distances do not have this property in general.

We can use the covariance determinant to put an ordering on the diversity of sets of solution vectors and the Mahalanobis distance to quantify the diversifying effect of vectors on a set. The Mahalanobis distance based on the covariance estimated from the data satisfies all four criteria on the list.

To illustrate the computations, we begin with an example shown in Figure 17.3, which is similar to the one given by (Glover, Løkketangen and Woodruff, 2000). In this contrived example, we label the fours columns as indicators for the use of sub-contractors in L.A., S.F., and Hamburg and the in-house production quantity of SKU AJ8172 (in thousands)

The addition of a vector $B=(0,0,1,0.31)$ would seem to be much more diversifying than $C=(0,0,1,0.04)$. The only variable that differs between B and C is the production of AJ8172. Both values are within a standard deviation of the mean for set \mathcal{A} so neither would be considered an outlier using only univariate analysis. However, if one conditions the statistical analysis on the value of the Hamburg column, a reasonable person would say that vector B does not seem

to come from the same population as \mathcal{A} . I.e., its addition would be diversifying. This is because for the other vectors, the production of AJ8172 is negatively correlated with the Hamburg facility.

The Hamming distance does not distinguish. The Euclidean distance from the mean of \mathcal{A} to B is less than the distance to C . In other words, Euclidean distances give the wrong answer. Scaling the elementwise contribution to the distance by the range or standard deviation does not help (as an aside, note that the range of binary elements that vary is always one.)

The Mahalanobis distances from the mean of \mathcal{A} under its covariance matrix correctly identifies B as being more distant. Unfortunately, the calculation of a full-rank covariance matrix for a set of vectors of length $p = n$ requires a set of vectors that does not lie entirely in a subspace. This means that at a minimum the set must contain $n + 1$ vectors and for most optimization problems, many more vectors will typically be required to span the full n dimensions. For even modest sized optimization problems this is not reasonable. In order to have a working definition of diversity, thousands of solution vectors are needed. Although it was not on our list of criteria, one would implicitly assume that a general purpose method for measuring solution variety in a DSS would function properly after a reasonable number of solutions having been found, with the meaning of “reasonable” parameterizable. A remedy for this difficulty that also increases the plausibility of normality is the use of *principal components* (PC).

PC is a standard statistical technique for dimension reduction. Given q , which is the reduced dimension, we project the data vectors onto vectors in dimension q selected so that the fraction of the variance represented by the lower dimensional vectors is as high as possible. The projection vectors used are the first q eigenvectors of the sample covariance matrix. Clearly, q must be less than the sample size used to estimate the covariance matrix, since this bounds its rank. See, e.g., (Morrison 1990) for a detailed description. A geometric interpretation of interest here, is that PC results in a projection on to the principal axes of the ellipsoids of equal Mahalanobis distance. A drawback is that the process of computing the projection vectors is not affine equivariant.

4. Fieldwork

Ultimately the long run success of topological manipulations in a DSS environment is a matter of having metrics that do a reasonable job of matching human perceptions. This creates a need for field research.

We collected data from two different examples, each of which is small enough to be reproduced in this paper but real examples that are large enough to be of interest to the planners. In both cases we asked for a set of good solutions from one or more domain experts and then asked other domain experts to provide topological statements about the solutions.

Pair $\mathcal{A}^{(1)}, \mathcal{A}^{(2)}$	Expert	Euclidean	$m(\cdot, \mathcal{A})$	
			$q = 1$	$q = 2$
$\mathcal{A}^{(1)}, \mathcal{A}^{(3)}$	close	217	0.84	
$\mathcal{A}^{(2)}, \mathcal{A}^{(3)}$	far	437	1.99	
$\mathcal{Z}^{(1)}, \bar{\mathcal{A}}$		2488	0.32	1.1
$\mathcal{Z}^{(1)}, \bar{\mathcal{A}}$	far	2692	4.84	22.3
$\mathcal{Z}^{(2)}, \bar{\mathcal{A}}$	close	1105	2.12	5.1

Figure 17.4. Cisco planning example

4.1 Production Planning

Production planners at Cisco Systems, Inc. were asked to generate a varied set of good production plans for a particular product. This planning problem has 10 decision variables. They produced the following three plans, given in vector form:

$$\mathcal{A}^{(1)} = (1060, 1040, 1200, 950, 950, 1100, 950, 950, 1100, 950);$$

$$\mathcal{A}^{(2)} = (1200, 1050, 1050, 1000, 1000, 1000, 1000, 1000, 1000, 1000);$$

$$\mathcal{A}^{(3)} = (920, 920, 1160, 920, 920, 1160, 920, 920, 1160, 920).$$

Using the CISCO spreadsheet to produce good plans, we produced the following three plans

$$\mathcal{Z}^{(1)} = (1580, 0, 1720, 0, 1525, 1675, 0, 1600, 0, 1650);$$

$$\mathcal{Z}^{(2)} = (1900, 0, 0, 1720, 1525, 1675, 0, 1600, 0, 1650);$$

$$\mathcal{Z}^{(3)} = (1240, 550, 1160, 630, 1210, 590, 1280, 920, 620, 1450).$$

When considering the \mathcal{A} set alone, Cisco personnel considered $\mathcal{A}^{(1)}$ and $\mathcal{A}^{(3)}$ to be the most similar pair; $\mathcal{A}^{(2)}$ and $\mathcal{A}^{(3)}$ were deemed the least similar. From among the solutions in set \mathcal{Z} , $\mathcal{Z}^{(3)}$ was thought by Cisco planners to be the most similar to the solutions in set \mathcal{A} and $\mathcal{Z}^{(2)}$ was said to be the least similar.

Table 17.4 summarizes the application of various metrics to the data. Within each group, we see that both the Euclidean metric and PC coupled with Mahalanobis distances qualitatively matched the Cisco personnel with respect to the most distant solutions, which in many DSS and scatter search applications is more important than finding similar solutions. As an aside we note that the $q = 2$ results are not presented for the within- \mathcal{A} because the Mahalanobis distances are all the same between a group of points with only one more member than the dimension (in this case three points in dimension 2) based on a covariance estimated from the same set. An important thing to note is that the most distant point has a distance that is beyond the 95% point of a χ^2 distribution in both the $q = 1$ and $q = 2$ cases.

4.2 Course Scheduling

Personnel responsible for scheduling courses for an MBA program were asked to generate five good schedules and the scheduling decision maker was asked to assess their relative distances. Schedule pairs (2,3) and (3,4) were thought to be far apart and schedule pair (2,4) was considered to be the closest pair.

Course schedules are not generated by people or displayed to people as numerical vectors, but rather as lists of courses with corresponding meeting times and days of the week. In order to manipulate schedules as part of a DSS or a metaheuristic such as scatter search, a vector representation is needed. Depending on the purpose (e.g., optimized scheduling), a variety of representations are possible (see, e.g., Burke and Ross, 1990 or Vigla and Karaboyas, 1998).

We tried two sensible vector representations for the solutions that we were given. For all representations, we numbered the courses. The simplest representation is to represent the schedules as a permutation vector giving the sequence of courses, but this is too simple since it discards information about the course meeting times. Two representations that are more sensible for distance comparisons are referred to as the *block representation* and the *center representation*. For the block representation we numbered the time blocks used by courses and created vectors indexed by the course numbers with the block number for the course as the data for the vector element (i.e., if the course numbered three occupies the block numbered two, then $x_3 = 2$). Here are the five solutions in block representation:

$$\begin{aligned}\mathcal{A}^{(1)} &= (4, 15, 23, 14, 21, 19, 18, 1, 17, 2, 21, 20) \\ \mathcal{A}^{(2)} &= (6, 12, 25, 19, 11, 10, 18, 24, 13, 20, 1, 5) \\ \mathcal{A}^{(3)} &= (4, 9, 23, 16, 17, 11, 10, 5, 8, 7, 26, 24) \\ \mathcal{A}^{(4)} &= (20, 22, 1, 17, 14, 16, 10, 26, 15, 22, 3, 7) \\ \mathcal{A}^{(5)} &= (1, 19, 20, 10, 15, 13, 16, 22, 14, 26, 7, 3)\end{aligned}$$

The second representation is similar. For the center representation the average meeting time during the week is computed for each course and these are sorted and given a number based on their position in the sorted list. The vectors are indexed by the course numbers with the data for the element given by the center number. Here are the five solutions in center representation:

$$\begin{aligned}\mathcal{A}^{(1)} &= (4, 16, 21, 14, 17, 23, 22, 1, 20, 2, 17, 15) \\ \mathcal{A}^{(2)} &= (8, 12, 25, 23, 10, 9, 22, 24, 13, 15, 1, 6) \\ \mathcal{A}^{(3)} &= (4, 7, 21, 18, 20, 10, 9, 6, 5, 11, 26, 24) \\ \mathcal{A}^{(4)} &= (15, 19, 1, 20, 14, 18, 9, 26, 16, 19, 3, 11) \\ \mathcal{A}^{(5)} &= (1, 23, 15, 9, 16, 13, 18, 19, 14, 26, 11, 3)\end{aligned}$$

Table 17.5 outlines the results and Table 17.6 summarizes them. The (3,4) pair is fairly well identified as being a well separated pair and the (2,3) pair seems to be identified by the center representation based metrics. While $q = 2$ seemed to work a little better in the Cisco example, it didn't seem to work as

Pair	Expert	Block Representation $m(\cdot, \mathcal{A})$			Center Representation $m(\cdot, \mathcal{A})$		
		Euclidean	$q = 1$	$q = 2$	Euclidean	$q = 1$	$q = 2$
$\mathcal{A}^{(1)}, \mathcal{A}^{(2)}$	far	41.4	1.64	2.08	38.2	1.53	1.54
$\mathcal{A}^{(1)}, \mathcal{A}^{(3)}$		18.6	0.01	0.08	30.6	0.23	1.83
$\mathcal{A}^{(1)}, \mathcal{A}^{(4)}$		49.4	2.08	2.42	43.7	1.97	2.77
$\mathcal{A}^{(1)}, \mathcal{A}^{(5)}$		40.4	1.65	1.95	36.8	1.51	1.58
$\mathcal{A}^{(2)}, \mathcal{A}^{(3)}$		40.8	1.65	2.04	41.3	1.77	2.63
$\mathcal{A}^{(2)}, \mathcal{A}^{(4)}$		31.7	0.44	2.56	31.7	0.44	2.13
$\mathcal{A}^{(2)}, \mathcal{A}^{(5)}$		17.0	0.01	0.23	28.1	0.03	0.60
$\mathcal{A}^{(3)}, \mathcal{A}^{(4)}$		49.8	2.09	2.47	45.3	2.20	2.21
$\mathcal{A}^{(3)}, \mathcal{A}^{(5)}$		41.0	1.66	1.92	40.3	1.74	2.21
$\mathcal{A}^{(4)}, \mathcal{A}^{(5)}$		29.8	0.43	2.32	29.5	0.46	1.56

Figure 17.5. Course scheduling example

Pair	Expert	Block Representation $m(\cdot, \mathcal{A})$			Center Representation $m(\cdot, \mathcal{A})$		
		Euclidean	$q = 1$	$q = 2$	Euclidean	$q = 1$	$q = 2$
$\mathcal{A}^{(2)}, \mathcal{A}^{(3)}$	far	4	3	6	3	2	2
$\mathcal{A}^{(2)}, \mathcal{A}^{(4)}$	close	4	4	10	4	3	6
$\mathcal{A}^{(3)}, \mathcal{A}^{(4)}$	far	1	1	2	1	1	3

Figure 17.6. Course scheduling example summary giving the rank sorting by increasing or decreasing distance, as appropriate

well as $q = 1$ here. The pairs that were thought to be similar were not well identified by the metrics tested.

5. Conclusions and Directions for Further Research

We have described methods for inducing a metric for the space of solutions. The use of principal components for dimension reduction comes at the cost of some information loss but facilitates use of covariance information. Furthermore, we provide some fundamental research concerning methods that are fully general and do not rely on any knowledge of the problem.

There are two potential uses for metrics in the space of solutions: 1) to identify distant vectors to add to a population and 2) to induce a topology for support of DSS development. The second is outside the normal paradigm of metaheuristic research, but is potentially very important. This paper has employed field research for basic research to shed some light on this topic.

It is encouraging that the Euclidean metric worked pretty well given appropriate solution representations. There is some evidence that Mahalanobis distances might provide better results in some settings with the appropriate dimension reduction using PC. Unfortunately, none of the metrics did a good job of identifying solutions that were considered by domain experts to be close to one another or to a population. Happily, for many of the applications in DSS's or in algorithm design, identifying distant solutions is the more important.

Of course, the research presented here is not conclusive and since it is behavioral it can never be, but the results are encouraging. In addition to the need for additional field work, additional computational and theoretical work would be helpful. For example, it would be useful to have some indication of the properties of solution representations that do and do not lead to effective discrimination of distant solutions using the Euclidean metric. The importance of affine equivariance is also worthy of further study.

As optimization methodologies are embedded in an increasing number of application from Enterprise Resources Planning through intelligent agents for web search, metrics in the space of solution vectors will become an major issue. In many of these applications one would like to have a population based method such as scatter search so that the user can be shown a variety of good solutions. The objective function provides a reasonable way to quantify the meaning "good." It is hoped that this paper provides a first step in the direction of computational viable means of quantifying the meaning of "variety."

Acknowledgments

This research was supported in part by grants from the National Science Foundation and the National Partnership for Academic Computing Infrastructure.

References

- Anderson, T.W. (1984) *An Introduction to Multivariate Statistical Analysis*, 72–75, John Wiley and Sons, New York.
- Burke, E. and P. Ross, eds. (1990) *Proceedings of the 1st International Conference on the Practice and Theory of Automated Timetabling*, Springer-Verlag 235–239.
- Glover, F. (1995) "Scatter Search and Star Paths: Beyond the Genetic Metaphor," *OR Spektrum*, 17:125–137.
- Glover, F. (1997) "A Template for Scatter Search and Path Relinking," In Hao et al. (eds) *Artificial Evolution, Lecture Notes in Computer Science*, Springer 13–54.
- Glover, F. and M. Laguna (1997) *Tabu Search*, Kluwer, Boston.
- Glover, F., A. Løkketangen and D.L. Woodruff (2000) "Scatter Search to Generate Diverse MIP Solutions," in *Computing Tools for Modeling Optimization and Simulation*, Laguna and Velarde eds., Kluwer, 299–320.
- Greenberg, H.J. (1998) "An Annotated Bibliography for Post-solution Analysis in Mixed Integer Programming and Combinatorial Optimization," In D.L. Woodruff (ed), *Advances in Computational and Stochastic Optimization, Logic Programming, and Heuristic Search*, 97–148, Kluwer, Boston, MA.
- Greistorfer, P. and S. Voß (2004) Controlled Pool Maintenance in Combinatorial Optimization", In C. Rego and B. Alidaee, *Adaptive Memory and Evolution: Tabu Search and Scatter Search*, Kluwer.
- Hoch, S.J. and D.A. Schkade (1996) "A Psychological Approach to Decision Support Systems," *Management Science*, 42: 51–64.
- Love, B.C. (2000) "A Computational Level Theory of Similarity," *Proceeding of the Cognitive Science Society*, 316–321.
- Morrison, D.F. (1990) *Multivariate Statistical Methods*, McGraw-Hill.
- Sharda, R., S.H. Barr and J.C. McDonnell (1988) "Decision Support System Effectiveness: A Review and an Empirical Test," *Management Science*, 34:139–159.
- Stamatopoulos, P., E. Viglas and S. Karaboyas (1998) "Nearly Optimum Timetable Construction Through CLP and Intelligent Search," *International Journal on Artificial Intelligence Tools*, 7:415–442.
- Steuer (1986) *Multiple Criteria Optimization: Theory, Computation and Application*, Wiley, New York.
- Wagner, H.M. (1995) "Global Sensitivity Analysis," *Operations Research*, 43:948–969.
- Woodruff, D.L. (1998) "Proposals for Chunking and Tabu Search," *European J. Oper. Res.*, 106:585–598.
- Zeleny, M. (1982) *Multiple Criteria Decision Making*, McGraw-Hill, New York.

Chapter 18

CONTROLLED POOL MAINTENANCE FOR METAHEURISTICS

Peter Greistorfer¹ and Stefan Voß²

¹*Institut für Industrie und Fertigungswirtschaft, Karl-Franzens-University Graz, Universitätstraße 15/G2, A-8010 Graz, Austria, peter.greistorfer@kfunigraz.ac.at;*

²*University of Hamburg, Institute of Information Systems (Wirtschaftsinformatik), Von-Melle-Park 5, D-20146 Hamburg, Germany, stefan.voss@uni-hamburg.de*

Abstract Recent metaheuristic developments have proved to be successful especially in cases where their fundamental concepts are complemented with pool-oriented approaches such as scatter search. These algorithms maintain a reference set of high quality solutions which are repeatedly used during the search in order to guarantee a fruitful balance between diversification and intensification. However, maintaining such a pool is not a trivial undertaking. The main problem is to find a balance between the attempt to collect a number of high quality solutions, which often results in similar solution properties, and the need to guarantee a certain degree of diversity in the pool. In the present study we highlight the advantage of pool-oriented design while focusing on controlled input and output operations. We analyze the main pool components for metaheuristics and present a review of the literature on relevant strategies in tabu search, scatter search, and path relinking. Additionally, we provide a synthesis of the literature that highlights the critical aspects of effective pool maintenance.

Keywords: Pool, Population, Elite Solutions, Diversification, Intensification, Similarity, Distance, Tabu Search, Scatter Search, Path Relinking

1. Introduction

Recently, metaheuristics have been investigated to explain the behavior, the success, and the relationships between various generic methods. One of the key aspects regarding metaheuristics in general is the interplay between *intensification* (concentrating the search into a specific region of the search space) and *diversification* (elaborating various diverse regions within the solution space). That is, it has very often been appropriate, on one hand, to explore promising

regions of the search space in a detailed manner (intensification) and, on the other hand, to lead the search into new and yet unexplored regions of the search space (diversification). Within intelligent search including the relationship between these two significant mechanisms the exploration of memory plays a most important role in ongoing research.

In this context, developments, e.g., in scatter search (SCS) and path relinking (PR), Campos et al. (1999), Glover (1998), have proved to be successful where fundamental metaheuristic concepts are complemented with pool-oriented approaches. In those a reference set of solutions (e.g., high quality or elite solutions) is built, maintained, and used either repeatedly or at least once during the search in order to guarantee a fruitful balance between diversification and intensification. Here, fruitful indicates an intelligent interplay where intensification does not outweigh diversification and vice versa but to use them together. However, maintaining such a pool is not a trivial undertaking.

In this paper we primarily focus on approaches that make use of a pool where all elements, commonly designated as elite solutions or reference set members, are directly encoded as *phenotypical* solution instances. We refer to a *pool* as a data structure used to store a number of solutions which have turned out to be potentially useful throughout the search. Members of a pool are called *elite solutions*. Its well-known encoding counterpart is the *genotypical* encoding based *population* of genetic algorithms (GA). In GA terminology (see, e.g., Reeves 1993) a genotype means the (encoded) chromosome and the phenotype its physical (decoded) expression.¹

The meaning and consequences of these different representations are not obvious. All the more, the more effective computational schemes or data structures work with vectors of binary variables or with permutation vectors. The main difference is not the encoding in a computer program, but the way solutions are treated in the algorithm. Phenotypical algorithms directly work on solution representations which need not be de-/encoded, whereas genotypical algorithms work on encoded solution representations. To our belief, we assume that phenotypical algorithms favor a direct information utilization, whereas the genotypical algorithms put a risk of loss of information on the fly from iterative genotypical processing to the final result which is – as a rule – always expressed as a phenotypical solution.

As another distinction we point out that the number of elements in a pool, following the designs of SCS and PR, is typically smaller than commonly used GA population sizes. Further, it is noted that the creation of a pool may only be performed step-by-step during the search procedure and that its use need not be a permanent one like it is done in GAs. Therefore, while discussing the role of pool architectures, it is an important distinction in which way a pool maintenance corresponds to the time line suggested for a specific method.

The aim of our paper is to provide a conceptual, yet unifying framework for pool architectures. After providing a pool template in Section 2, Section 3 gives a survey of several options and ideas for elaborating such a framework. While the metaheuristics community currently favors the idea of including elite solutions into such a pool based on objective function values, we incorporate different ideas that might have been overlooked such as pool solutions based on structural properties (e.g., measured by means of diversity instead of objective values). In Section 4 we put several concepts found in the literature into the light of our framework. While the discussion includes famous exponents such as SCS and PR we emphasize other ideas, too. The discussion is moderately interleaved with some computational results as well as hints to related studies. Finally, some conclusions and ideas for future research are provided.

2. A Pool Template

Various approaches in the literature have shown the advantage of applying pool based methods as they use a pool to direct or redirect the search to explore the search space. To date, however, a unifying framework has not yet been provided and discussed throughout the literature. While SCS may be referred to as a forerunner to such a framework there may be additional approaches that are similar to SCS but do not fit exactly into the SCS framework. One example in this respect with recent interest is PR. Instead of building and maintaining a pool, PR often starts from a given pool.

The aim of this section is to propose new ideas of how a pool method may be composed, which tasks are combined in main functions, and which evaluation approaches have been neglected in the literature. In doing so, first, the main focus lies on transfer functions and their detailed design with respect to the relations that are contained in the aggregated information of solution subsets.

Based on the idea of a pool template, we have various optional processes to be performed if a heuristic algorithm works in conjunction with a pool component. While such a component may serve several aims, the most important aspect is to collect high-quality information.² These processes are directly reflected in some components of the source code which describes a heuristic procedure. For the sake of an easy explanation we call them *input function* and *output function* (*IF* and *OF*). Generally, an *IF* decides upon the inclusion of a new potential elite solution into the pool, whereas an *OF* refers to the generic output questions. As pointed out above, an *elite solution* is any solution that has been accepted for inclusion in the pool. Both, *IF* as well as *OF*, are equipped with facilities to incorporate intensification and diversification issues. The main algorithmic settings to be clarified and, therefore, essential questions are the following.

The basis of our further investigations is a *pool template* (PT), where the following notations are used (cf. Figure 18.1). A pool of $p \geq 1$ solutions is

denoted by P . Its input and output transfer is managed by two functions which are called IF and OF , respectively. S is a set of solutions with cardinality $s \geq 1$. A solution combination method (procedure SCM) constructs a solution from a given set S , and IM is an improvement method.

```

1. Initialize  $P$  by an external procedure
WHILE termination=FALSE DO BEGIN
    2.  $S := OF(P)$ 
    3. IF  $s > 1$  THEN  $S' := SCM(S)$  ELSE  $S' := S$ 
    4.  $S'' := IM(S')$ 
    5.  $P := IF(S'')$ 
END
6. Apply a post-optimizing procedure to  $P$ 
```

Figure 18.1. Pool Template

Depending on the method used, in Step 1 either a pool is completely (or partially) built by a (randomized) diversification generator or filled with a single solution which has been provided, e.g., by a simple greedy mechanism. Note that a crucial parameter that deserves careful elaboration is the cardinality p of the pool. The main loop, executed until a termination criterion becomes true, consists of Steps 2 to 5. Step 2 is the call of the output function which selects a set of solutions, S , from the pool. Depending on the kind of method represented in the PT, these solutions may be assembled (Step 3) to a working solution S' which is the starting point for the improvement phase of Step 4. The outcome of the improvement phase, S'' , is then evaluated by means of the input function which possibly feeds the new solution into the pool. Note that a post-optimizing procedure in Step 6 is for facultative use. Although it is usually not contained in many heuristics, it is strongly recommended to do so (see, e.g., Moscato 1993). It may be a straightforward greedy improvement procedure if used for single-solution heuristics or a pool method on its own. As an example we quote a *sequential* pool method, the tabu search with path relinking in Bastos and Ribeiro (2000). Here a PR phase is added *after* a tabu search (TS) has built the necessary pool. A *parallel* pool method on the other hand uses a pool of solutions *while* it is constructed by the main or guiding procedure (e.g., a GA or SCS).

Several heuristic and metaheuristic paradigms, which are more or less pool-oriented or are not pool-oriented at all, can be summarized under the common PT frame:³

- a) Local Search: PT with $p = s = 1$.

- b) Simulated Annealing: $p = 2, s = 1$ incorporating its probabilistic acceptance criterion in IM .⁴
- c) Standard Tabu Search: $p = 2, s = 1$ incorporating adaptive memory in IM .
- d) Genetic Algorithms: $p > 1$ and $s > 1$ with population mechanism (crossover, reproduction and mutation) in SCM of Step 3 and without the use of Step 4.
- e) Scatter Search: $p > 1$ and $s > 1$ with subset generation in OF of Step 2, linear combination of elite solutions by means of SCM in Step 3, e.g., a tabu search for procedure IM and a reference set update method in IF of Step 5.
- f) Path Relinking (as a parallel pool method): $p > 1$ and $s = 2$ with a path relinking neighborhood in SCM . Facultative use of Step 4.

For excellent framework discussions in the spirit of our work the reader may refer, e.g., to Vaessens, Aarts and Lenstra (1998), Hertz and Kobler (2000), Taillard et al. (1998).⁵ It should be noted that the primary intention of this paper is not to provide the PT as a new framework within the field of metaheuristics (even if PT may be classified in this way) but to provide a discussion of options for combining search intensification and search diversification by means of pool oriented approaches. Furthermore, these important issues within metaheuristics are not mutually exclusive but simultaneous features.

An early local search framework or template is given in Vaessens, Aarts and Lenstra (1998). Some additional insights on a framework regarding evolutionary algorithms are provided in Hertz and Kobler (2000). According to them, on a simplified scale many algorithms may be coined evolutionary if they can be reduced to the following frame:

- 1 Generate an initial population of individuals
- 2 While no stopping condition is met do
 - (a) cooperation
 - (b) self-adaptation

Cooperation refers to an information exchange among individuals while self-adaptation refers to the fact that individuals (solutions) evolve independently.

The phrase *adaptive memory programming* (AMP) emerged in connection with TS to refer to a collection of general and flexible memory-based strategies to enhance the power of heuristic search (Glover 1997). Considerations relevant to incorporating adaptive memory within other metaheuristics are also described

in Taillard et al. (1998). That is, iteratively constructing (new) solutions based on the exploitation of some sort of memory (as we intend to do based on our pool approach) may be viewed as AMP process, especially when guided by learning mechanisms, that help to adapt the collection and use of the memory. Based on the simple idea of initializing the memory and then iteratively generating new solutions (utilizing the given memory) while updating the memory based on the search, most metaheuristics can be subsumed by the AMP approach. This also includes the idea to exploit provisional solutions in population based methods that are improved by a local search approach. For an interesting empirical study which may fit into the AMP frame as well as into our pool template, see Fleurent and Glover (1999).

3. Options for Elaborating a Pool Template

In this section we discuss ideas for elaborating a framework as developed above by means of the PT. Based on some key components for the maintenance of a pool we show possible uses in the interplay of intensification and diversification. Most importantly, we investigate quality measures to define, incorporate, and use elite solutions.

3.1 Key Components for Pool Maintenance

For both, *IF* and *OF*, a basic design decision relates to the timing of the pool changes, i.e., to the point in time when it is indicated to change the content, and, in doing so, eventually the structure of the pool. Regarding an *IF* the following questions need to be addressed.⁶

- At which *time* should an *IF* be called and applied (IF_{time})?
- *How many* solutions should be added into the pool (IF_{number})?⁷
- How is the *quality* of a solution and its potential input status determined ($IF_{quality}$)?
- Which element(s) should be *removed* (IF_{remove}) from the pool to make room for new arrivals?

Regarding an *OF* the following questions may be considered:

- *When* should a solution be *taken* from the pool (OF_{time})?
- *How many* solutions should be taken from the pool (OF_{number})?
- How can these solutions be identified? This is again a question of quality ($OF_{quality}$), even if it is seen from the other end of the quality spectrum.

- In which way should these solutions be *used* to improve the search (OF_{use})?

The above description structures the main problematic nature of pool management but it can be easily seen that several questions directly refer to more than one functional aspect such as $IF_{quality}$ and $OF_{quality}$ or OF_{number} and OF_{use} . Table 18.1 summarizes the main pool functions and the next paragraphs specify some more or less well-known approaches from the literature, which can give an answer to the questions raised above.

Table 18.1. Summary about pool functions

	<i>IF</i>	<i>OF</i>
time	IF_{time}	OF_{time}
quality	$IF_{quality}$	$OF_{quality}$
removal	IF_{remove}	—
how many	1	OF_{number}
use	—	OF_{use}

IF_{time} and OF_{time} : Answers to these are both, simple and complex. The moment of a pool input (IF_{time}) can be easily recognized. Generally, this is the case when it is advisable, due to an $IF_{quality}$ evaluation, to store the current solution. Output times (OF_{time}) also may be uniquely given where the heuristic method itself iteratively needs a set of solutions as it is the case for all SCMs. By far more difficult is the decision when to make use of the pool if diversification or intensification tasks should be performed. This may be triggered by adaptive memory or running time and/or storage restrictions.

$IF_{quality}$ and $OF_{quality}$: The main aspect in evaluating a given solution certainly is its associated objective value. In the literature the better-than-worst ($IF_{quality}$) and the best-of-the-pool approach ($OF_{quality}$) are the most frequently used techniques. Here a current solution is fed into the pool if it improves on the current worst solution of the pool, whereas on the output side always the best element of the pool, which is usually the current best solution, is chosen to be reconsidered. Note that counterarguments against this proceeding may easily be developed as the current best solution might not only diversify the search but also lead it into a local optimum or a basin of attraction with only local optima which need not be globally optimal.

Other settings for the input as well as the output side come from evolutionary and/or genetic optimization. Well-known choice criteria in this context, all of them based on randomization, are the use of uniformly distributed selections, the roulette wheel criterion, linear ranking, and pure evolutionary settings as they are proposed, e.g., by Rechenberg (1972), Schwefel (1977).

These basic and straightforward considerations, however, completely neglect the information provided in structural aspects. Such an information is always manifested and embedded in the specific domain of a certain optimization problem. For example, in a vehicle routing algorithm it may be a good idea to utilize the information how often a certain customer is linked with a specified successor (and/or customer), or in a facility location problem the subset of open facilities in elite solutions is usually seen as a good indicator for quality. The importance of structural knowledge has also a distinct justification in situations where more than two solutions (most frequently the current one and a best-known solution) have to be related to each other. Usually pool based solution procedures decide whether a solution is included into the pool due to its objective value (as can be seen for various GA approaches). However, regarding the interdependencies and the simultaneous use of intensification and diversification, we argue that the structure of a solution or the structural difference between two or more solutions should play a much more important role.

In cost and structural quality again the time component is significant. The data can be evaluated in a *static* or in a *dynamic* way. Usually, a static as well as a dynamic determination of an objective value is not a problem. Moreover, as demonstrated in many papers, this determination can be performed extremely fast if effective update mechanisms accompany the evaluation process while changing solutions within neighborhood search iterations. This ease of computations has to be denied if it is a question how structural elite attributes should be treated. Firstly, the evaluation of a given set of good solutions is the easiest task and can be done by simple comparison (static, *a posteriori* view). Secondly, also the detection of quality attributes during the search (*dynamic* view) can be easily managed, e.g., by means of frequency counts. However, it may be difficult to imply useful decisions based on quality attributes. One of the reasons can be a limited knowledge of search trajectories and respective search directions (which are difficult to anticipate). Finally, the most demanding case is the one in which it has to be decided whether a given solution should be added to the pool because of its promising structure or should be rejected because of its poor structure (static *a priori* view). In this case a solution has to be judged based on its decision variable values (which naturally does not exclude the possibility of gaining still more information by adding cost considerations). The basis of such a structural judgement is a measure of *distance* between solutions and will be outlined below.

IF_{remove}: The most common setting for eliminating a solution from the pool is the removal of the worst solution contained in it. However, it is the message of this paper to highlight that structural differences as they were explained in the context with *IF_{quality}* and *OF_{quality}* may also be considered to decide which solution should be cancelled.

OF_{number} and OF_{use} : The number of solutions from the pool depends on the task which is provided in intensification, diversification, as well as in algorithmic foundations.

Commonly, an algorithm falls back on a pool solution if the current search trajectories appear to be of lower interest. One of these settings, where a degenerated 1-element-pool is maintained, is the restart with the best-known solution that is always stored regardless which heuristic method is used. Usually this is done, e.g., after a number of non-improving iterations. The aim of more diversification may be realized by taking the worst solution from a pool. This kind of strategy may be referred to as some sort of min-max quality strategy. In doing so, it is guaranteed – though the worst element is chosen – that quality aspects also are sufficiently treated since, otherwise, this worst element would not have been feeded into the pool. Another option is to choose an element that is most different from those already explored (see Section 3.2).

A further aspect on a pool output side finds its relevance in algorithmic needs. This is especially true for pure SCMs like GA or SCS procedures which build upon the combination of a number of solutions to simply keep the algorithm running. Regarding the numerical value, in GAs usually two solutions can be combined whereas an SCM in an SCS provides mechanisms to combine more than two elements.

Following this overview of influencing maintenance functions within an effective pool management, we next concentrate on some special topics which will be the guidance of the search by means of intensification and diversification (OF_{use}) and the measurement of distance (an aspect of $IF_{quality}$ and $OF_{quality}$).

3.2 Diversification Versus Solution Quality

An open research area is devoted to measures for evaluating diversity. In this section we address some of the not yet answered questions using simple examples taken from the literature. We first provide some arguments regarding the coordination of diversification and intensification with a focus on diversification issues. Note that most of the strategies sketched below may also be adapted to pursue intensification issues. While this list of arguments and strategies is by no means complete, it helps to understand the elaboration in subsequent sections. The main question seems to be finding the right degree of diversification. Some aspects from the following list will be described more carefully in the next section where we consider checking for redundancy.

- Whenever a memory-guided search seems to be trapped in a local optimum region, a simple strategy would be to perform a random move, reset the memory, and restart the search.

- A more sophisticated approach which is founded on the TS short-term memory is realized in the framework of the *Reactive Tabu Search* (RTS). The central idea of RTS incorporates an adaptive *self-adjustment* mechanism for the length of the prohibition period. This is done whenever there is evidence that solutions are repeatedly generated in the search directory. Repetitions are recognized by comparing a trial solution with those ones of the past. Such a comparison is enabled by keeping all solutions or their hashed values in memory. (See Battiti and Tecchiolli (1994) for detailed adjustment functions.)
- To support the self-adjustment approach in RTS, a more radical *escape mechanism* is released when too many solutions are repeated too often. It simply consists of a number of random moves. Here the RTS diversifies on a pool of sequentially stored solutions, which currently suffer from being trapped in a local basin of attraction. A somewhat different concept in the same spirit concerns the idea of so-called kick moves (see, e.g., Martin, Otto and Felten, 1991). More advanced escape mechanisms may incorporate, e.g., a simulated annealing run instead of pure random moves. One approach which considers this idea combines the RTS escape mechanism with a more structured one. For example, a simulated annealing run is applied instead of the usual escape mechanism (see, e.g., Fink and Voß, 2001 for an indepth numerical study).
- Frequency approaches known from advanced TS incorporate additional memory, e.g., based on frequency counts maintained throughout the search Glover and Laguna (1997). A simultaneous use of intensification and diversification may be obtained by means of a coordinated utilization of both recency as well as frequency based memory. An interesting example of a multi-trajectory oriented approach in this respect extends the reverse elimination method (see, e.g., Sondergeld and Voß, 1996).

Information utilization based on a pool template has primarily been described for elite solutions entering and leaving a pool (IF and OF). In a more general sense we need not only those solutions but also additional information besides them. To make this clear, we illustrate it by a simple example considering simulated annealing (SA). While SA usually allows for immense diversification in the beginning of the search it tends to intensify in just one region throughout later stages of the search. By using additional information (similar but much simpler to that used in frequency based memory indicated above) this may be overcome by so-called reheating based on some information provided by an OF. This even allows a better tuning between diversification and intensification within the SA (first empirical results on this idea are reported in Osman and Christofides, 1994).

As we have pointed out earlier, commonly, an algorithm falls back on a pool solution if the current search trajectories appear to be of lower interest. Another reason may be a search trajectory that requires some sort of indication in which direction it needs to go (e.g., when it is unable to leave a so-called basin of attraction without additional indications). While this is usually performed on the basis of an objective function evaluation, here also memory components may come into play (as, e.g., outlined in AMP). Based on our pool approach we need not only incorporate a solution from a pool but can also reassemble information based on the solutions from the pool as may be done in a specific implementation of Steps 2 and 3 of our pool template. Here again the RTS may serve as an example. Another simple example is the reheating strategy within SA.

3.3 Checking for Redundancy

Based on the observations in the previous section an important issue deserving discussion refers to the checking for redundancy. We say that *redundancy* occurs in an algorithm run when, according to the history of past iterations, parts of search trajectories, or more generally, solution generations and corresponding calculations are performed repeatedly. Concerning (dis)similarity or distance considerations, as discussed in Section 3.4, this naturally involves redundancy aspects. So every identity may be interpreted as (total) similarity. But there is also another topic covered by redundancy, which can be directly ascribed to neighborhood properties itself. It is designated by algorithmic redundancy and, so to speak, located on an offshore solution properties level.

Two solutions are identical if their structures or physical expressions coincide. On closer inspection, however, this involves certain pitfalls. Departing from a real-world solution, in modelling one has to determine a so-called data solution, which is the algorithmical correspondence of the former one. That is, one obviously deals with a possibly twofold error. On the one hand, the checking of a pair of data solutions leads to the wrong acceptance of the hypothesis that the corresponding real-world solutions are different (*identity error*, IE). And, on the other hand, the counter-hypothesis may have been wrongly accepted although the real-world solutions actually are different (*discrimination error*, DE).

A straightforward example shows that IEs and DEs often can be traced back to insufficient data structures and incomplete identity checking, respectively. Assume three data solutions of a symmetric vehicle routing problem of dimension 5 to be given as a sequence of records with the two attributes (*customer number*, *route number*) like

$$\begin{aligned}D_1 &= ((1, 1), (3, 1), (4, 1), (5, 2), (2, 2)), \\D_2 &= ((5, 1), (2, 1), (4, 2), (3, 2), (1, 2)), \\D_3 &= ((1, 1), (3, 1), (4, 2), (5, 2), (2, 2)).\end{aligned}$$

Identity is checked by means of a function I_d which scans all positions of a given attribute from left to right while comparing the entries in some D_i and D_j . Then, in testing D_1 and D_2 , an IE occurs since $I_d(D_1, D_2) = \text{false}$ leads to the false assumption that there exist two different real-world solutions R_1 and R_2 . In reality, however, D_1 and D_2 correspond to the single solution $R = \{(1, 3, 4), (2, 5)\}$ which is a set of two sequences (routes) that contain the customer numbers. The false result, i.e. the occurrence of an IE, does not depend on the degree in which I_d scans the positions. I_d would be *false*, regardless of the facts how *many* of the positions $(1, \dots, 5)$ are compared and *which* set of attributes (customer, route, or both) is taken for comparison. Even if in this example incomplete and complete checking result in the same (wrong) decision, there are also situations where an error can occur due to incomplete checking. For example, based on the comparison of the first two positions, $I_d(D_1, D_3) = \text{true}$ has the consequence of accepting identity, which is a DE. Unlike that, in a complete checking, I_d clearly would reject identity, because already the next position, 3, correctly results in assuming dissimilarities.

To summarize our example, one can detect that the success of identity checking basically depends on the data structure and on the checking dimension. The IE demonstrated, of course, is due to the insufficient representation of the real-world situation in the data structure. Firstly, an adequate dealing with the symmetry condition is missing and, secondly, the sequence representation of routes does not correspond to the real-world set situation. Sets are poorly represented in data structures and can only be treated at the expense of higher computational effort. But this has to be accepted, since an asymmetric representation reduces the real degree of freedom, which wrongly lets the problem look easier than it is.

Another direct aspect of identity is to avoid algorithmic redundancies that may emerge within compound neighborhoods. An exchange neighborhood for permutations (as a full method), e.g., can be easily computed and no duplicate solutions will be present. Neither two solutions of the computed set will be identical, nor will any neighborhood member be equal to the solution from which the neighborhood was built. The same is valid for insertion moves. This situation, however, changes completely if both types of moves are combined. Figure 18.2 gives an extract of the neighborhood built from the permutation $(1, 2, 3, 4, 5)$. Each of the 57 neighbors is constructed according to the lexicographical increase of the *4-loops-template* which is given in brackets as (s, sl, t, tl) at the beginning of every line. Indices s, t denote the starting positions of sequences to be moved and sl, tl their respective lengths. If a zero-

length sequence participates then an insertion move is addressed, otherwise an exchange move.

1=(1021)	ins 2 before 1	2-1-3-4-5	
2=(1022)	ins 23 before 1	2-3-1-4-5	
3=(1031)	ins 3 before 1	3-1-2-4-5	
4=(1032)	ins 34 before 1	3-4-1-2-5	
5=(1041)	ins 4 before 1	4-1-2-3-5	
6=(1042)	ins 45 before 1	4-5-1-2-3	
7=(1051)	ins 5 before 1	5-1-2-3-4	
8=(1120)	ins 1 before 2	1-2-3-4-5 D1	
9=(1121)	exc 1 with 2	2-1-3-4-5	built last at k=1
10=(1122)	exc 1 with 23	2-3-1-4-5	built last at k=2
11=(1130)	ins 1 before 3	2-1-3-4-5	built last at k=9
12=(1131)	exc 1 with 3	3-2-1-4-5	
13=(1132)	exc 1 with 34	3-4-2-1-5	
14=(1140)	ins 1 before 4	2-3-1-4-5	built last at k=10
15=(1141)	exc 1 with 4	4-2-3-1-5	
16=(1142)	exc 1 with 45	4-5-2-3-1	
17=(1150)	ins 1 before 5	2-3-4-1-5	
18=(1151)	exc 1 with 5	5-2-3-4-1	
19=(1230)	ins 12 before 3	1-2-3-4-5 D2	built last at k=8
20=(1231)	exc 12 with 3	3-1-2-4-5	built last at k=3
:			
50=(3142)	exc 3 with 45	1-2-4-5-3	built last at k=46
51=(3150)	ins 3 before 5	1-2-4-3-5	built last at k=49
52=(3151)	exc 3 with 5	1-2-5-4-3	
53=(3250)	ins 34 before 5	1-2-3-4-5 D3	built last at k=48
54=(3251)	exc 34 with 5	1-2-5-3-4	built last at k=47
55=(4051)	ins 5 before 4	1-2-3-5-4	
56=(4150)	ins 4 before 5	1-2-3-4-5 D4	built last at k=53
57=(4151)	exc 4 with 5	1-2-3-5-4	built last at k=55

Figure 18.2. Neighborhood redundancies

As can be seen from the last column in Figure 18.2, which gives a link to the original appearance of a permutation, the number of duplications is considerably high. Overall, there are 26 duplicate solutions which are 46% of the whole neighborhood. Although made for the sake of demonstration, this example shows that significant improvement can be achieved if the neighborhood mechanism is carefully observed within the algorithmic circumstances. Naturally, all identity aspects have their true justification in the context of regulating any pool transfer. Checking for duplicate solutions regarding the pool transfer is the key element according to Glover (1998). He offers advanced subroutines dealing with two scopes: No solution which has already been checked should be added to the pool and the (partial) guarantee that no subset of the pool is selected twice in order to generate new trial solutions. Additional hints are given in Section 4.

Another feature of redundancy comes up with the inspection of several TS strategies which all deal with the problem of cycling. Cycling is a process where solutions are re-visited during the search and, hence, it is one of the basic tasks to lower the number of these duplicate inspections. To trigger such

a behavior several concepts have been proposed. We distinguish between two tabu list concepts Voß(1996), the static concept, and the dynamic concept.

The historically first memory structure is the so-called *static* concept, sometimes also referred to as *tabu navigation*, *simple TS*, or *fixed TS*. It uses a short-term recency memory, which stores recent moves and/or a subset of attributes characterizing a solution, respectively. The central function of this memory is the management of the so-called tenure (period), which is the number of iterations for which a certain move is declared to be forbidden.⁸ Following Battiti (1996), however, one may prefer to speak about a tenure as a *prohibition period* since TS short-term memory structures, even in their advanced forms, can be designed without the use of a list environment. The tenure is deterministically fixed or varied due to some probabilistic scheme. The main disadvantages of the static concept are that tenure values cannot be given easily – often they have its origin in simple rules of thumb – and that static considerations tend to be too restrictive with respect to the release of a tabu status.

The static concept is complemented by a *dynamic* concept, in which the tabu status of a trial move is explicitly checked by the use of special routines which enable an adaptive inclusion of the search history. To point out the most prominent ones, we cite the *reverse elimination method* (REM) and the *cancellation sequence method* (CSM). Both work with an adaptive tenure rule which depends on sequences of moves. These sequences are maintained in memory and used to determine the tabu status of a potential move. The REM can be designed to serve as explicit memory, where all traversed solutions are stored via their attribute relations. At least theoretically (depending on the depth of the backtracing), REM is able to exactly exclude those moves which lead to a solution that had been generated.⁹ This does not hold for the CSM which may suffer from the prohibition of more moves than actually needed to prevent returning to a known solution. For a deeper insight into tabu list management see Dammeyer,Forst and Voß(1991); Dammeyer and Voß(1993); Glover (1990), Taillard (1991).

A kind of dynamic concept is also embedded in the RTS (see, e.g., Battiti 1996). As outlined in Section 3.2, the main ideas in RTS are the adaptive prohibition period, the escape mechanism and the use of advanced data structures to store solutions. In terms of RTS terminology, the prohibition period is controlled by a reactive feedback mechanism. Further, to avoid *limit cycles* (endless cycling over a set of solutions) and *chaotic trapping* (being trapped in a limited search space), respectively, an *escape mechanism* is launched if too many solutions are repeated too often. The data structures mentioned above are used to map the search history into the memory and are organized within hash tables and linked lists whose locations, *bucket arrays*, are directly calculated from the solution treated. Such a memory can be accessed within constant time (linear with the number of iterations). With respect to some alternative RTS

design, a fruitful research area is to investigate different and higher-level escape mechanisms such as, e.g., the application of some randomized metaheuristic as it is the SA algorithm.

3.4 Identity and Beyond

A major concern in pool oriented search is the measuring of how solutions *relate* to each other in a structural context ($IF_{quality}$ and $OF_{quality}$). In a very simplified way, solutions are either identical or not. While identity is a binary attribute, non-identity always has to deal with a gradual measure which may represent an approximation. Often it is sufficient to evaluate a *distance* between two objects. Measuring the internal degree of similarity within a population can be based on such isolated pairwise calculations which then are aggregated to reflect a collective measure, e.g., by calculating a mean value. Generally, we designate a subset of solutions to be more similar than another subset if they have a smaller distance between each other than the solutions in the other subset. However, it is not clear what makes a good distance measure. Below we attempt to provide some insights regarding possible answers. Our main focus is on scales, distances and metrics.

With increasing information level an object attribute is either of nominal, ordinal (rank), interval, or ratio type. In the same sequence as just enumerated, attributes of these types give evidence about *equality*, *smaller/greater-relations*, *smaller/greater-differences*, and the *equality of ratios*. Interval and ratio type scales are often treated together in social sciences within the common group of cardinal scales. We start our overview with considerations for the cardinal scales.

An important aspect in measuring distances is the nature of the problem which influences the way how it can be transcribed into a mathematical model. The main representation schemes are vectors (arrays), matrices, or list structures. For our analysis we focus on vector models. Let X and Y be two vectors of dimension n (which represent two solutions of a problem to be solved), then the so-called *Minkowsky- r -distance* defines the distance between X and Y as

$$d_r := d(X, Y) = \left(\sum_{i=1}^n |x_i - y_i|^r \right)^{\frac{1}{r}}. \quad (18.1)$$

Well-known special cases are the *Manhattan* or *rectangular* distance d_1 and the *Euclidean* distance d_2 . Including these special cases, the interesting range for r is $1 \leq r \leq \infty$. d_r is a monotonously decreasing function of r . For $r \rightarrow \infty$, it leads to a pure dominance criterion, where the largest variable value difference is weighted with 1 (and all other differences with 0), known as the

Chebychev distance:

$$d_\infty := d_c(X, Y) = \max_{i=1}^n |x_i - y_i| . \quad (18.2)$$

Distance measure (18.1) satisfies the conditions of identity, $d(X, X) = 0$, symmetry, $d(X, Y) = d(Y, X)$, and transitivity, $d(X, Y) + d(Y, Z) \geq d(X, Z)$, and hence is a *metric*. Interval and ratio scales are also referred to as metric scales and nominal and ordinal scales as non-metric scales.

The Euclidean metric as an example suffers from several disadvantages. Firstly, it causes distorted distances if the variables are not equally scaled, i.e., the d_2 -norm is scale-variant. To overcome this problem, all vectors should be normalized by means of a simple transformation. A variable value, e.g. x_i , is set into relation to its average

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x_i \quad (18.3)$$

and to the standard deviation

$$s_x = \sqrt{\sum_{i=1}^n \frac{(x_i - \bar{x})^2}{n}} \quad (18.4)$$

to obtain its standardized value

$$x'_i = \frac{x_i - \bar{x}}{s_x} . \quad (18.5)$$

To outline the second disadvantage of the d_2 -norm it is helpful to understand the notion of statistical *correlation* which on its own constitutes a distance measure as will be discussed below. The classical correlation coefficient is known since 1864 as the *Bravais-Pearson* or *product-moment* correlation. Assuming a 2-dimensional random variable, whose realizations follow a normal distribution, allows an estimate for the population correlation with

$$r(X, Y) = \frac{\text{cov}(X, Y)}{s_x s_y} , \quad (18.6)$$

where $\text{cov}(X, Y)$, the *covariance* between the vectors X and Y , is

$$\text{cov}(X, Y) = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y}) . \quad (18.7)$$

The covariance is a frequency measure explaining whether above (below) average X -values often correspond to above (below) average Y -values, which is

known as *positive covariance* (negative and no-covariance are interpreted analogously). Already the covariance can be understood as a measure of similarity since it informs about the difference of two vectors. It is only adequately used if the scales over all variables in all vectors are the same, i.e., $\text{cov}(X, Y)$ is scale-variant, which, e.g., can be expressed by $k\text{cov}(X, Y) = \text{cov}(kX, lY)$. The correlation coefficient can be understood as a standardized covariance measure. It is (scale-)invariant against linear transformations and describes the narrowness of a coherence by a value between -1 and 1 (perfect positive and negative coherence) where 0 stands for no (linear) coherence. The correlation may also be represented as the covariance of two standardized variables, which can be empirically expressed by

$$r(X, Y) = \frac{1}{n} \sum_{i=1}^n x'_i y'_i . \quad (18.8)$$

The correlation coefficient is very sensitive with respect to the sample which has been drawn from the population if either not the whole range of characteristics is expressed in the sample or if it is composed of marginal groups. Testing the significance of an empirical correlation can be performed with t -distributed test quantities.

Now we can state the second disadvantage of the d_2 -norm, which is the disadvantage of not taking population correlations into account. That is, if certain correlation patterns are present in the population, then one would expect them to be also present in a sample of vectors drawn from that population. Vectors that differ in ways that do not match the correlation patterns of the population (or distribution) should be considered to differ more than solutions whose differences are consistent with the variance (and covariance, see below) patterns in the population. Consider the following example: Let there be three 3-dimensional vectors $X = (1, 5, 6)$, $Y = (4, 5, 6)$, and $Z = (1, 2, 6)$ and assume that in the reference population the first vector elements are highly variant and uncorrelated, whereas the second and third elements have low variance, but are highly correlated. Having these population patterns in mind, one would assume that $d_2(X, Y) < d_2(X, Z)$ because z_2 does not match the correlation and variance pattern of the population. This assumption is additionally motivated by $r(X, Y) = 0, 94$ and $r(X, Z) = 0, 79$. However, in fact both distances are identical ($d_2(X, Y) = d_2(X, Z) = 3$). Such a misinterpretation can be overcome by more sophisticated distance concepts like the so-called *Mahalanobis* distance.

The Mahalanobis distance measure, shortly d_m , is a Euclidean distance measure which is adjusted with respect to correlation influence. If X and Y are realizations of an n -dimensional random variable and $S^{-1} := (s_{ij}^{-1})$ describes the inverse of a symmetric positive definite variance-covariance matrix, then function

$$d_m := d_m(X, Y) = \sqrt{\sum_{i=1}^n \sum_{j=1}^n (x_i - y_i) s_{ij}^{-1} (x_j - y_j)} \quad (18.9)$$

is scale-invariant and can take correlations into account (see Glover et al. 2000). To quote the authors, one "can use the covariance determinant to put an ordering on the diversity of sets of solution vectors and d_m to put an order on the diversifying effect of vectors in a set". It can be argued that d_m is a "perfect" measure for evaluating distances which have to be adjusted. However, one has to pay the price of a higher computational effort, because the determination of an empirical S , more precisely, the determination of a valid *estimate* of S , as well as the determination of its inverse are no trivial tasks (theoretically, it needs a set of $n + 1$ vectors). In the context of mixed integer programming it is reported in Glover, Løkketangen and Woodruff (2000) that there is a practical need for thousands of solution vectors, although the theoretical lower limit is $n + 1$, which can only be managed by concepts such as the *chunking* of solution vectors.

One of the main application areas of the Mahalanobis distance is cluster analysis. If variables of different solution vectors correlate more or less with each other, then the value of d_2 is biased towards those characteristics which highly correlate. This leads to an emphasis of certain aspects which may distort the results. That is, if actually a balanced weighting scheme over all variables is assumed in a practical application *and* if the data is correlated, then the Mahalanobis distance function is a solution.¹⁰ To end the discussion on the Mahalanobis distance note that this measure is identical with the Euclidean distance if no correlations are present, i.e., in this case we can set $d_m(X, Y, S) = d_m(X, Y, I) = d_2$, where I is the identity matrix.

All the distance measures treated so far are applicable to cardinal scales. The next type of scale, containing less information than the interval scale, is the ordinal scale, where information is stored within smaller/greater-relations and no normal distribution assumption is given.

Here so-called *rank correlations* are used. *Spearman's rank correlation* assigns to each value in $X(Y)$ a rank which reflects its quality with respect to all other $X(Y)$ values. These ranks then are used in the correlation formula (18.8). Likewise in a similar working approach, which has become known as *Kendall's τ* , the hypothesis can be tested, whether two series of ranked observations are independent or not. Another approach to handle ordinal attributes is to transfer the ranks into binary code. For example, the ranks 1, 2, and 3 can be transformed to 2-dimensional realizations (0,0), (0,1), and (1,0). To these vectors one of the nominal scale approaches can be applied.

The nominal scale is the one with the lowest scale of information and provides yes/no-information on the basis of binary vectors. For our explanations we use

the following notation. Assume two n -dimensional vectors X and Y and let a, b, c , and d be the number of positions where an (x_i, y_i) entry matches the corresponding patterns (1,1), (0,1), (1,0), and (0,0). This in mind, several *similarity* coefficients can be defined (see also Bortz 1999). The *S-coefficient* is

$$s(X, Y) = \frac{a}{a + b + c} \quad (18.10)$$

with the corresponding distance $d_s = 1 - s(X, Y)$. It puts into proportion the amount of existing common attributes with respect to the number of characteristics present in at least one vector. If an overall similarity measure is required, i.e., one which also includes the evaluation of non existing characteristics then the numerator of (18.10) is increased by the value d and related to the dimension $n = a + b + c + d$. This results in the so-called *Simple Matching Coefficient*

$$SMC(X, Y) = \frac{a + d}{a + b + c + d}. \quad (18.11)$$

The distance measure according to (18.11) reads as $d_{SMC}(X, Y) = 1 - SMC(X, Y)$ and is, like d_s , a value between 0 and 1, giving an appropriate percentage number when multiplied by 100. Note that in specific research domains, e.g., pattern recognition or genetic algorithms, d_{SMC} has become well-known as *Hamming distance* which usually is not normalized by the dimension n since for a given encoding and/or instance the latter is a constant.

As a last remark to nominal scale measures, note that the product-moment correlation as given in (18.8) is the general case of its dichotomic counterpart, the so-called ϕ -coefficient, a correlation coefficient that is calculated as

$$\phi(X, Y) = \frac{ad - bc}{\sqrt{(a + c)(b + d)(a + b)(c + d)}} \quad (18.12)$$

describing the relations in terms of similarity between two binary (nominal) vectors X and Y .

Extending the scenario of various scales, we may further distinguish between two types of *heterogeneity*, i.e. overall and partial heterogeneity, which lead to special correlation coefficients and distance functions, respectively. By *overall* heterogeneity we mean the fact that two vectors are attributed with different scales. For example, the correlation between a binary attributed vector X and an interval scaled vector Y can be determined with the so-called *point-bi-serial correlation*, whereas a binary vector and a rank-attributed vector can be evaluated with respect to similarity by means of a *bi-serial rank correlation*. However, since these cases are of minor importance for heuristic problem representations, they are not discussed here. The interested reader can find related information in Bortz (1999). The influence of *partial* heterogeneous information plays a more important role in heuristic pool maintenance. A partial

heterogeneity is given if not all characteristics (variables) of a given object (vector) are of identical, i.e. *homogeneous*, character while all instances considered have the same type of heterogeneity. For example, it may be interesting to evaluate the relations of two solutions not only by their objective values, but also by a certain characteristic which does exist or not. In this case we are confronted with heterogeneous information that is represented on different scale levels. That is, one needs a distance function that is able to cope with a dichotomic (binary) value that describes the existence of that certain characteristic, e.g., a pre-defined high-quality sequence of consecutive elements, like customers in a route, and which is able to simultaneously judge an interval scaled objective value. These heterogeneous distance functions are not very common in the area of combinatorial optimization and heuristics, but could find an application in many ways. For a comprehensive overview over this topic see Wilson and Martinez (1997).

As an outlook to distance measures we briefly sketch the so-called *Levenshtein* or *edit* distance (Levenshtein 1966). It is well-known from pattern matching, DNA analysis or speech recognition and determines the distance between a pair of strings. In an unweighted version this distance can be defined as the minimum number of elementary operations which are commonly insert, delete and substitute (a character) to align a source string to a target string. Obviously, an adaptation of this technique could significantly contribute for measuring general diversity in metaheuristic pool methods, since it may provide an algorithm with specific means which is based on the neighborhood operators actually used. For the calculation of the Levenshtein distance various methods have been proposed, e.g., a dynamic programming formulation can be found in Myers (1999).

We should also note the NP-hard *maximum diversity problem* (see, e.g., Glover, Kuo and Dhir, 1998). Consider a set of elements (e.g., a set of solutions of a pool) and some of their common attributes (e.g., variables included, objective value), for every element in the set, each of these attributes can be in one of several possible states. The problem is to select a predetermined number of elements from the set that encompass a greatest variety of attribute states (which leaves again open the appropriate distance measure to be used for evaluation purpose). To summarize, important measures that have been overlooked in the design of a pool may incorporate a combination of structural information and objective values.

Finally, we list some examples of how distance measures have been adapted for specific problem formulations. Some of them will be exemplified in a later section.

- Hamming distances for the quadratic assignment problem (QAP) are introduced by Voß (1995) and later also used by Merz and Freisleben (2000).

- Clustering distances for the generalized assignment problem (GAP) can be found in Semet and Taillard (1993).
- Euclidean distances are utilized in the SCS approach of Cung et al. (1997).
- An adaption of the SCS of Cung et al. (1997) for a routing problem can be found in Greistorfer (2003b).

4. A Review of Pool Architectures

In this section we survey some papers and methods which make use of the elite solutions idea. Of course this selection is due to our subjective view and by far not complete. The intention is to provide some insights into possibilities for using the concepts described above.

4.1 Hybridizing Genetic Algorithms

Genetic algorithms as early exponents using pool based architectures are not comprehensively treated in our paper partly due to reasons mentioned above. Nevertheless, it is often discussed whether GAs can perform better than, e.g., random search as outlined by Wolpert and Macready (1997), among others. It is generally agreed upon that hybridizing a GA with a local search component may be advantageous in this respect (i.e., using an *IM* according to Step 4 of the PT). Corresponding approaches are widespread within the literature. Here we only provide a short remark regarding hybrid algorithms; some more detailed elaborations follow in the subsequent section.

Two major approaches exist for such hybrid algorithms: Lamarckian evolution and the use of the Baldwin effect (see Whitley, Gordon and Mathias, 1994). The Lamarckian evolution attempts to improve a chromosome directly by the local search procedure, i.e., the genetic code and the fitness value are altered. Exploiting the Baldwin effect impacts only the fitness values without changing the genetic code. Thus the Baldwin effect allows for some intensification while the diversity of the pool itself is not effected.

4.2 Memetic Algorithms

An introduction to population approaches for optimization is given by Moscato (1993). It is one of the founding papers dealing with multi-solution design based on the memetic algorithm's (MAs) metaphor. MAs basically "combine local search heuristics with crossover operators" and, therefore, "some researchers have viewed them as hybrid genetic algorithms" Moscato (2001). The applications studied are the traveling salesman problem (TSP) and the binary perceptron (BP), which are well-known problems that have been tested in a variety of solution frameworks and are suitable for their convenient vector representations.

The TSP approach is based on a pool of 16 solutions, understood as *agents*, which are arranged in a ring network topology. This topology determines the direction of interactions triggered by competition and cooperation mechanisms. In this context it can be helpful to specify the difference between the active behavior of an agent and the passive nature which is commonly ascribed to a pool member, either used in a GA or for an explicit controlling pool approach like PR or SCS. The difference is visible in the manner how these individual elements interact. Both types may be forced to act by means of an external function, e.g., an SCM, but only agents have the equipment to initiate an interaction on their own. All 16 TSP agents are involved in an iterative three-phase approach, each of which underlies a specific optimization philosophy. In Phase 1 each agent undergoes an individual search process where a 2- and 3-opt Lin-Kernighan type neighborhood is used (as an *IM*) that is embedded in a SA acceptance criterion. Due to the probabilistic nature of the latter, this phase is responsible for a diversity increase within the population of agents. In the competition Phase 2 all individuals both challenge and are challenged in a way that the challenged tour of one of two agents is again probabilistically deleted and substituted by the challenging tour. This behavior incorporates unifying tendencies in the population and, therefore, is responsible for intensification. The final Phase 3 is a period of cooperative interaction, again determined by the given ring topology. Cooperative interaction is based on a probabilistic proposition, like the competition criterion in Phase 2, of an individual but the cooperation takes place on another sub-level of the ring network. If a proposition is accepted between two agents then they are mated by means of the *order crossover* as suggested for permutation problems by Goldberg (1989). In order not to subsequently override the specific effects of competition and cooperation, i.e. Phases 2 and 3, there is always an individual search Phase 1 inserted after a cycle of operations in Phases 2 and 3.

The MA proposed for the BP is founded on a combination of a straightforward descent algorithm, the *discrete gradient descent* (DGD), a TS procedure, and a genetic one-point crossover generation mechanism. At the beginning each agent, representing a solution by an estimated weight vector, builds a set of neighbor solutions by flipping the value (+1 or -1) in each position of the weight vector and choosing the best improving neighbor for the next iteration. This DGD process stops if no further improving neighbors of a given solution can be found. Then a local optimal vector serves as the basis of a TS which attempts to decrease the objective function of the BP. The main loop of the procedure consists of the genetic generation mechanism which is followed by a DGD for each agent and the testing for the *loss of diversity* in the current population. If this happens then the population suffers from a diversity crisis and a diversifying TS phase is launched for each agent. Then the generation

mechanism starts again. Note that the diversifying TS operates on a different neighborhood than the "normal" TS in the initialization phase.

The results of the study, both for the TSP and the BP learning problem, are quite interesting. Agents which interact "via a competitive and cooperative set of rules, develop global optimization properties" while the blend of a pool method with an individually used TS is able to provide high-quality results. The main message of this approach is the success of adding phenotypical improvement algorithms to a standard GA approach, either a metaheuristic like TS or a simple greedy-like local optimizer.

4.3 Probabilistic Diversification and Intensification

Rochat and Taillard (1995) introduce the so-called *probabilistic diversification and intensification in local search* for the vehicle routing problem (VRP) with time windows. It lays the foundation for a basic optimization idea called POPMUSIC that was suggested some years later as a common metaheuristic framework by Taillard and Voß (2002). The approach in Rochat and Taillard (1995) again relates to a pool method but with a somewhat different character. Instead of storing whole solutions in the pool, there are only solution sub-parts collected. A natural sub-part decomposition for any VRP of course is a specific *tour* which covers a subset of customers to be serviced (commonly designated as *route*). In the initialization phase a number of (complete) solutions is generated by a non-deterministic local search from which in the sequel tours are separated and collected in the initial pool. This is done by labelling the tours in an ordered, i.e. increasing, sequence of their *fitness* value which is the overall cost of the schedule from which a tour had been derived. Such a collection of partial solutions and their subsequent combination contains an implicit "combination effect" (as it is naturally observed in GAs) which is able to intensify the search based on the individual sub-parts generated at the beginning by a diversifying local search. Then the main process probabilistically - based on the current evaluation - takes tours from a working pool, a copy of the initial pool, and extends an overall solution to be constructed. If the working pool is empty and the solution constructed is not complete it is made feasible by including the remaining uncovered customers. After a new local search run the resulting tours are inserted in the initial pool and the process re-starts with a new working pool, iterating until a stopping criterion is met. A single iteration of this algorithm combines diversification and intensification in different phases. If the size of the initial pool is large enough the search regions are diversified in a sufficient way, which is additionally supported by the creation of partial solutions. The more the search continues – with increasing completeness of solutions – the aspect of combining partial solutions and, hence, an intrinsic intensification takes place.

The authors report on the robustness of their method and mention the advantages of its parallelization potential as well as the ease of its adaptation for various types of VRPs. The most interesting aspect in this successful pool approach is the direct transfer of GA basics to the area of phenotypical encoding, though only partial solutions are stored. Finally, it should be noted that this approach is closely related to *chunking* and *vocabulary building*. See, e.g., Woodruff (1998), Kelly and Xu (1995) for some references exploring these concepts in more detail.

4.4 A Star-Shaped Diversification Approach in Tabu Search

A star-shaped diversification approach in TS is developed in Sondergeld and Voß(1996) and discussed for the turbine runner balancing problem (TRBP). The TRBP is to locate blades in a turbine to minimize the difference between the center of mass and the geometric center.¹¹

The first star-shaped diversification approach uses a REM with parallel running lists (and tabu lists) instead of single lists in conventional TS concepts. These lists are used to control a set of solutions which simultaneously evolve. By means of distance measures (e.g., based on the Hamming metric in certain settings where binary variables are used for problem representation) between these lists it is guaranteed that at each iteration a current solution referring to a specific running list maintains a given minimum distance to the current solutions of all other running lists. With a unique starting solution for each running list and a sufficiently large lower bound for this distance measure, a star-shaped spread of the search trajectories into different regions of the solution space can be realized.

In a further and more complex multiple list approach the disjointness is assured as follows. The first trajectory is built by the conventional REM performed on a given starting solution. Starting from the same solution, for each of the remaining trajectories the running lists of previous trajectories are successively connected to the current running list in reverse order. That is, pseudo-trajectories are constructed representing fictitious search paths from the current solutions of all other running lists or trajectories over the starting solution up to the current solution of the actual running list. These connected running lists together with the current running list determine which moves have to be set tabu. By this proceeding it is guaranteed that the current trajectory is not entering other trajectories. That is, duplication of a solution of any of the previous trajectories is avoided.

Note that this so-called exact star-shaped approach only works with TS methods operating on dynamically controlled running lists, whilst diversification using distance measures even works with conventional static methods. While

each trajectory intensifies its search it is simultaneously guided in a diversifying manner.

4.5 Reactive Tabu Search with Path Relinking

The *Reactive Tabu Search with Path Relinking applied to the Steiner problem in graphs* by Bastos and Ribeiro (2000) incorporates the RTS-typical adaptive tenure rule. This rule defines a tabu period as a function of the number of repetitions for a given solution. If this number reaches a specified upper limit, a diversifying random move is performed. After this reactive mechanism is passed, a local search neighborhood call either returns a Steiner node to be inserted (or eliminated) or forces the tenure to be reduced if all moves are forbidden. This decrease is typically greater than the standard decrease (within RTS) at the end of an iteration. If the local search call indicates that all moves are forbidden, the procedure restarts at the main loop. Otherwise, if an admissible move is found, the newly built solution is evaluated for a possible pool input (*IF*) based on the number of Steiner nodes (distance as a diversification criterion) and its "quality" (cost). Additionally to the random move, triggered by the repetition counts, a diversifying random move is performed if a number of non-improving iterations has passed. The main loop ends with a periodically performed decrease of the tenure.

After the RTS, the overall approach continues with a post-optimizing PR procedure which is regarded as an intensification phase. In the PR every pair of elite solutions is linked by greedily making the best current move (add or drop a Steiner node). If a new best solution is found, then it replaces the worst in the pool and the PR process starts again until no new improvement is found. This pool approach can be interpreted as a sequential one where the construction and the use of the pool relate to different algorithmic periods, whereas the SCS paradigm plans these periods to be run simultaneously. Similar to Bastos and Ribeiro (2000), the methods of Greistorfer (1998a), Greistorfer (1998b) extend a TS to a pool component, offering a sequential and a parallel design, as outlined in Section 4.7.

4.6 Scatter Search and Path Relinking

Glover (1998) introduces a detailed template for SCS and PR including complete descriptions for pool input and output routines. SCS is a heuristic process working with a pool of phenotypically encoded elements designated as *reference solutions*. Subsets of elite solutions, derived from the pool, are used to generate new derivatives, generally based on an SCM. The SCM produces weighted centers based on linear combinations of solution vectors. It is important to allow non-convex projections in order to explore regions which are external to the convex hull of the reference solutions. Therefore, new solu-

tions, which may often be infeasible, are transformed by generalized rounding procedures or by (linear) programming techniques which ensure an acceptable vicinity (based on a chosen distance metric), of the newly combined solution to its reference points from which they were derived.

PR works in a similar way. Here two elite solutions, an *initiating* and a *guiding* solution, are selected from the pool (*OF*) and a path is constructed between these two points with the aim to hopefully find an improving point. The rationale behind this "combination" strategy is the same as in the SCS paradigm. High-quality solutions are likely to share common attributes which are gradually introduced by the combining PR process. That is, PR provides a useful means for synchronized intensification and diversification. In contrast to SCS, new solutions are generated by exploring search trajectories that connect pairs of elite solutions.

The key-ideas in Glover (1998) are represented by sub-routines which are able to provide a high diversity while effectively avoiding *duplications*. This relates to both *IF* and (at least to a specific amount) *OF*. The sub-routines suggested are: A diversification generator to initiate the pool, a subset generation method (SGM) as *OF*, an SCM to transform the reference solutions of the selected subset into one or more trial point(s), an *IM* which operates on newly constructed trial solutions and a so-called *reference set update method* as *IF*. The SGM, responsible for the pool output control has the important role to select subsets of different sizes from the pool (RefSet) and may be designed in a static or dynamic way. Solely in the dynamic version the algorithm ensures that no subset will be generated twice. This is the function of a SGM subprocedure, *SubsetControl*, which has to partition a current RefSet into old and new instances. This information is passed to the actual subset building sub-routines, which accordingly select members of the RefSet. These are combined by an SCM and possibly improved, while providing a permanent input for the reference set update method.

In *Scatter Search to generate diverse MIP solutions* Glover et al. (2000) demonstrate the important role of the distance metric used for the evaluation of a number of reference points. Commonly used metrics, as it is often the Euclidean distance, fail to be reliable instruments. As outlined in Section 3.4, this is mainly a question of an adequate metric that takes interactions and correlations into account. There are cases, like in mixed integer programming, where traditional metrics give a wrong evidence of actual coherence. To overcome these difficulties, the Mahalanobis metric and a *chunking* approach are used. The difficulties which arise if distance measures are not adequately applied are highlighted together with the strong contribution of the chunking concept in cooperation with the high-level metric used.

Meanwhile, SCS and PR have emerged as powerful optimization techniques in a variety of applications (see, e.g., Aiex et al. 2001; Bastos and Ribeiro 2000;

Cung et al. 1997; Greistorfer 2003b; Rego 1998). Both, SCS and PR, are excellent examples to study the basic concepts of the PT. On the other hand, the main advantage of the template is its flexibility to be adapted for several problems and the possibility to support established improvement heuristics which additionally may be designed to overcome a temporary infeasibility.

4.7 Hybrid Pool Designs for Tabu Search

Greistorfer (1998a) investigates several pool methods for the so-called cyclic regular max-min scheduling problem (CRSP), the problem of locating all vertices of a number of regular polygons on a circumference so that the minimum distance between any two vertices of different polygons is maximized. This problem has applications, e.g., in public transit time tabling, where the minimal time interval, i.e. a certain security span, between two different vehicles should be maximized.

The exploration of the CRSP is based on a greedy construction algorithm (GCA) and a TS procedure. The GCA starts with a 2-polygon arrangement and completes this (partial) solution by adding the remaining polygons, while trying to keep the "loss of optimality" as small as possible. The TS then attempts to improve this solution by rotating polygons in clockwise or counter-clockwise direction. The adaptive memory incorporates a short-term memory, attributed to the recency of polygon rotations, and a long-term memory which works with frequency counts of used rotation types. This basic approach, GCA+TS, is compared with two pool designs, each of which starting with the GCA.

The first method is a classical GA, where every polygon is represented by a single (binary) chromosome in which a solution is completely encoded by the *initial* coordinates of all polygons. A classical generation mechanism with reproduction, crossover, and mutation is applied. As a crossover the so-called *path relinking crossover* is used, which forms a binary path starting from an initiating chromosome by step-wise building new solutions with increasing degree of similarity with respect to the guiding solution (compare Glover 1989). The second pool design sequentially combines the TS with the GA and is called *genetic TS* (GTS). Here a TS run is split into two phases with an intermediate GA phase. In particular, the first TS run builds a pool of elite solutions which are the basis for the population mechanism of the GA phase. After the GA again the TS is applied, trying to improve the two best solutions of the final GA pool.

It is important to note that all three strategies, GCA+TS, GCA+GA and GCA+GTS, are run over the same total number of iterations. The results from this study demonstrate the superiority of using a pool approach which does not only rely on genotypical information. That is, GCA+CTS leads to better results than GCA+TS which itself outperforms GCA+GA.

As an extension the *irregular* version of the max-min scheduling problem (CISP) is studied in Greistorfer (1998b). Here the concept of the GTS is changed into the direction of a parallel pool neighborhood, where *parallel* has an ambiguous interpretation. Firstly, elite solutions stored in the population of the GA are already accessed during the course of the TS. Secondly, in doing so, at each iteration of the algorithm, a *hybrid neighborhood* set is generated from the complete neighborhood set of the current TS solution and from decoded elite solutions of the current population, which are chosen by a roulette wheel criterion after having applied a generation step.

Guided from the experience with the regular problem, some additional changes are made in this GTS. One is the short-term memory which turned out not to be as effective as expected, since it did not directly work on solution attributes (initial polygon coordinates). Improving this for the CISP, the short-term memory now records complete solutions, i.e., an explicit memory instead of the former polygon rotation memory is used (which may be called an *indirect* attribute memory). What concerns the helpful effects of a long-term memory implementation, the maintained population itself, as it was, contributes its cumulative knowledge and, therefore, a long-term memory is not independently incorporated in the algorithm.

If the latter approach serves as hybrid, i.e. geno-/phenotypical, pool design, another parallel pool method in Greistorfer (2003b) is purely phenotypical and hence can be understood as an SCS method. The algorithm was developed for solving the capacitated Chinese postman problem (CCPP). The goal of the (undirected) CCPP is to determine a least-cost set of routes in an undirected network subject to vehicle capacity and the assumption of an unlimited fleet.

The algorithmic backbone again is a TS which works with a set of neighborhood *operators* (exchange and insert types) on a long-term diversification strategy guided by frequency counts. The short-term tabu memory is defined on *edges* and simply prohibits reversal moves within a dynamically varied tenure period. The SCM component combines pool elements which have been collected by the TS. It is due to a transportation problem formulation, which is the generalization of the assignment operator of Cung et al. (1997), and finds a new solution by minimizing the total Euclidean distance within a subset of elite solutions.

The overall *Tabu Scatter Search* (TSCS) starts from a random pool which is maintained during the TS run. The SCM is occasionally called and forms a combined solution, which is post-optimized by a greedy-improvement procedure and then returned to the TS. This alternating process between TS and the SCM terminates after a pre-defined number of iterations. Note that this TSCS architecture differs from the template ideas of Glover (1998).

The TSCS is tested on several classes of problem instances. There are a number of constructed Euclidean grid-graph instances, pure Euclidean random

instances and a set of well-known instances from the literature. In a direct comparison with an old TS method (see Greistorfer 1995) the TSCS significantly improves the results for the own Euclidean classes (in 54% of all cases) and is clearly able to keep up with the well-known CARPET arc routing heuristic of Hertz, Laporte and Mittaz (2000) with respect to the instances from literature, justifying the merits of adding a pool component to a TS.

In Greistorfer (2003a) the focus is on the identification of promising strategies around SCS as well as general pool design. Addressed are problems which may be represented by permutations, where a second attribute, e.g. a substring membership of an element due to a capacity restriction, must be allowed for. Examined are again the CCPP and the TSCS. The three main components to be analyzed are $IF_{quality}$, $OF_{quality}$ and OF_{use} . Altogether, there are four $IF_{quality}$ strategies, four $OF_{quality}$ strategies and three OF_{use} strategies, yielding a total of 48 runs over a representative sample of CCPP test instances. Each of these runs is evaluated on the average of all best cost values which have been generated during the search. For the evaluation of a single strategy, this one was completely checked crosswise against all other possible pairs of remaining strategies.

In Section 3.3 we have outlined the clear comparability of solutions for (symmetric) problems as a founding premise in every pool method. To achieve this task, a sorting criterion is introduced that rearranges a given solution structure according to the following two sorting keys: The primary sorting key is the increasing number of all substrings' (routes') first element (customer) and the secondary key asks for increasing element numbers in each substring. For our first two examples in Section 3.3 this would result in $D_1 = D_2 = ((1, 1), (3, 1), (4, 1), (2, 2), (5, 2))$, logically obtaining $I_d(D_1, D_2) = true$, which avoids the identity error. Besides, for the $IF_{quality}$ given by Glover (1998) as well as its variations, a two-attribute based hash function is proposed that allows the mapping of *capacitated* permutations as they are, e.g., present in every capacitated routing problem.

The statistical test results of this study directly confirm the state-of-the-art assumptions as they can be found in phenotypical pool programming literature, which is mostly devoted to the SCS paradigm. For $IF_{quality}$ the strict diversification based on cost and hash identities works extremely well. However, in contrast to Glover (1998) the additional full duplication check does not appear to be necessary. Regarding $OF_{quality}$, the best setting was found using a distance function that favors outputs from the pool at which the distance of all subset members to the current best solution (which is always a pool member) is maximized. The *distance* between two solutions is equally determined like the hashing scheme as the sum of the (absolute) differences of customer positions plus the sum of the absolute (differences) of corresponding route numbers. Moreover, the LP-based solution method of Greistorfer (2003b) could be

justified in a direct comparison with two other settings. The first one ("a position votes for its element"), is based on *average elements* which are derived from all positions in the subset of elements which are to be combined. The second OF_{use} produces the opposite effect, analogously to the former method, a combined solution that reflects *average positions* ("an element votes for its position").

Summarizing the experience obtained, one can say that a sort of structured duplication check always raises the quality of a pool method and that sophisticated combination operators should be preferred against straightforward mechanisms. Regarding the input/output operations, it appears that the output side is more sensitive relating to the overall performance of the pool method discussed. Finally, it is noted that simple random based decisions in an $OF_{quality}$ choice criterion work quite well, however, better performing settings can be easily found.

4.8 Testing Various Scatter Search Designs

One of the recent investigation areas in the context of pool design has been the *Linear Ordering Problem* (LOP) Campos et al. (1999), Campos, Laguna and Martí (1999), Laguna, Martí and Campos (1999) which in economics has its application for the triangulation problem of input-output tables. The LOP seeks a permutation of row/column indices (sectors) of a square weight matrix in order to maximize the sum of the weights in the upper triangle.

Laguna, Martí and Campos (1999) propose an elite-TS approach that alternates an intensification and diversification phase. During the intensification phase a sector is probabilistically chosen according to its weight and re-positioned due to a best insertion neighborhood. After a given number of intensification iterations and the application of a local greedy optimizer (which has the same task as the DGD in Moscato (1993) explained above) the method switches to a diversification phase which builds upon frequency counts of sector moves recorded in the previous intensification phase. In fact, a sector is randomly chosen using a distribution which reflects the relative inverse frequency counts over all sectors. Furthermore, there is an additional intensification phase added where the best solution of the diversification phase is subjected to be the initiating solution from which a path is generated to a number of best solutions found so far.

Campos, Laguna and Martí (1999) introduce the application of an SCS for solving the LOP. The main algorithmic component in their pool management is a *measure of dissimilarity* (MOD). The individual MOD, iMOD, is calculated as the sum of the (absolute) position differences between a given sector permutation to be evaluated and a median sector permutation derived from all members of the pool. The population MOD, pMOD, is the sum of all iMODs

calculated for each member. In an analogous way a measure pCost is derived to evaluate the cost situation of a pool. After having standardized both values to percentage measures, their sum is finally used as an overall relative measure. This approach is able to judge a set of solutions by its structural diversity as well as its quality represented by the objective values.

The SCS starts with the construction of an initial pool by using a problem specific diversification generator based on the MOD ideas. Afterwards all elements of the pool serve as starting solutions for an individual local search improvement attempt. The main SCS loop consists of the following components. A RefSet is built from the pool by iteratively using a min-max-MOD criterion. Then for all subsets (of RefSet) to be considered the following is performed: Apply the SCM to the elements in the subset and use a local search to improve the outcome. Input a new better-than-worst and not contained solution to RefSet and go back to the same subsets type creations (involving the just fed new RefSet member), select and run a new SCM. Otherwise, in the non-input case, continue with another subset type generation and the SCM application. If all subsets are considered in this way a new pool is built using the diversification generator. One half of the next RefSet consists of the best solutions of the current RefSet, the other half is built by using the new pool which completes RefSet via the min-max-MOD approach. The results for LOLIB instances show that this LOP-SCS provides excellent solution quality. However, the running time behavior of SCS is inferior to that of a TS or a greedy algorithm. It is interesting to note that this SCS variant explicitly uses two collections of elite solutions at the same time, namely a pool and an SCS typical RefSet.

The main difference of Campos et al. (1999) in comparison to Campos, Laguna and Martí (1999) is that there is no rebuilding of the pool after an SCS iteration. That is, the method terminates after all RefSet elements have been designated as old elements which means that there was no improvement found during the last Subset generation run and improvement attempts, respectively. Additionally, we want to highlight an evaluation approach referred to as the *tracking of best solutions*. To manage this, a matrix records the ranks of the input solutions (column) which have been chosen to combine an SCM output of a certain rank (row). This is done in a cumulative way. So at the end, for a given output rank one can deduce the total number of SCM inputs as a function of their respective rank. This experiment clearly reveals that "high quality solutions tend to generate new solutions that are admitted to the reference set". The computational results, which have been extended by literature and random instances, again confirm the leading performance of the SCS approach.

4.9 Clustered Pools for Tabu Search

In Voß (1995) a REM-TS (diversification, Phase 1) maintains a clustered pool with different sub-classes by means of Hamming distances. Restarts are performed from the two best class representatives with reset memory (intensification, Phase 2), and Phases 1 and 2 are alternated iteratively. The overall approach is a hybrid of *logical interdependencies* (REM) and heuristical *clustering*.

Whenever search intensification is performed for the QAP, based on a promising solution, the idea is to perform those moves that will keep the permutations within the near vicinity of the respective solution. If search intensification is performed several times throughout the search it may be advantageous for the starting solutions of the intensification phase to be different from each other. In order to *diversify the different phases of search intensification*, some permutations that have been visited within the neighborhood search may be stored as possible candidates for a restart in future iterations. To obtain reasonable differences in new starting solutions, we propose that these solutions are stored by a clustering approach.

For any two solutions π and π' we may define some measure of similarity $\Delta(\pi, \pi')$. Whenever $\Delta(\pi, \pi')$ is less than or equal to a given barrier value, these solutions will be considered to belong to the same class of solutions, and otherwise will be considered to belong to different classes. Any time the search is restarted, a starting solution is selected from a class, that has not previously been used for intensification purposes, and eliminated from the respective class.

Consider the following measure of similarity between two permutations π and π' of length n , based on the Hamming metric, where $d_i := 1$ if $\pi(i) \neq \pi'(i)$ and $d_i := 0$, otherwise:¹²

$$\Delta(\pi, \pi') := \sum_{i=1}^n d_i \quad (18.13)$$

To initialize these classes we apply the following approach. Each class c consists of at most c_{max} elements and the number of classes is bounded by a number tc . Each nonempty class is represented by a permutation π_c with the best objective value among all elements of that class. Any solution π of the search which lies within a certain percentage of the overall best solution found so far is a candidate for being included into one of the solution classes.

Two cases may occur. First, given a parameter $\Delta_{diff} \in (0, 1)$, if $\Delta(\pi, \pi_c) \leq \Delta_{diff} \cdot n$ for class representative π_c then π is included into class c . If π becomes representative of c then we successively check whether any two classes have to be merged as their representatives have a similarity not greater than $\Delta_{diff} \cdot n$. Second, if π is not included within any of the tc classes then it constitutes a new class, eventually replacing that class whose representative has the worst

objective value among all classes. At the time of developing the approach numerical results in Voß (1995) presented new best known solutions to several benchmark instances of the QAP.

5. Conclusions and Future Research

In this paper we have discussed a pool template and the controlled maintenance of information within the pool. Using information from the pool allows an intelligent guidance of search processes of various metaheuristics with a special focus on the coordination of intensification and diversification. One of the major concerns in this respect is the use of information obtained throughout the search. We have argued that elite solutions should not only be elaborated based on objective function values but also on structural properties, e.g., incorporating a measure of distance between solutions. Many of the ideas in this respect need a very careful experimental investigation as part of future research. Nevertheless, our main conclusions are that pool maintenance as well as the use of appropriate distance measures will be a major issue of future success on the interplay of intensification and diversification in metaheuristics.

Furthermore, utilization of additional information collected throughout the search will enhance pool-oriented output and, even more important, input functions. Here we may have to gain much more knowledge regarding distance measures as well as statistical techniques. Also some careful investigation of landscapes, e.g., using statistical methods Reeves (2001), is worth being explored.

Notes

1. To give a trivial example we refer to a small knapsack problem instance of dimension 5. Suppose we have a feasible solution which intends the objects O_2 , O_4 , and O_5 to be packed into the knapsack. Then, a genotypical solution representation would be the binary vector $S = (0, 1, 0, 1, 1)$ and a phenotypical representation given by the set $\{O_2, O_4, O_5\}$. For a detailed discussion on the distinction between genotype and phenotype the interested reader is referred to, e.g., Schaffer and Eshelman (1996).
2. It should be noted that high quality does not necessarily mean or imply a good objective value.
3. While we restrict ourselves to most prominent examples in this list, other methods may be included as well, such as the pilot method Duin and Voß(1999) or POPMUSIC Taillard and Voß (2002). For recent surveys on metaheuristics see, e.g., Voß (2001); Ribeiro and Hansen (2002).
4. It should be noted that $p = 2$ and $s = 1$ seems to be unusual at first glance. For simulated annealing we always have a current solution in the pool for which one or more neighbors are evaluated and eventually a neighbor is found which replaces the current solution. Furthermore, at all iterations throughout the search the so far best solution is stored, too (even if no real interaction between those two stored solutions takes place). The same is also valid for a simple tabu search. As for local search the current solution corresponds to the best solution of the specific search, we have $p = 1$.
5. For a survey on metaheuristics' frameworks in an object-oriented software development context the reader is referred to the papers in Voß and Woodruff (2002).
6. Indices of IF and OF may be understood as respective design options.
7. We restrict ourselves to non-parallel computation single-input pools, i.e., we assume that we evaluate one solution at a time for possible inclusion into the pool.

8. Sometimes the tenure is loosely associated with the size of a tabu list, which can give a wrong idea of the number of attributes that deem a move to be tabu.

9. Note that the REM with no restriction on the backtracing is equivalent to the so-called *strict* TS, where a permanent tabu status is imposed on all solutions which were ever accepted. A modified version of the REM is discussed in more detail in Section 4.4.

10. Other solutions can be found within the *principal component (factor) analysis* or by building *residual variables* and exclusion of variables with high correlations, respectively.

11. For more recent ideas on the TRBP based on the POPMUSIC approach see Taillard and Voß (2002).

12. Note that $\Delta(\pi, \pi')$ refers to the sum of coefficients b and c in the simple matching coefficient SMC of (18.11).

References

- Aiex, R.M., M.G.C. Resende, P.M. Pardalos and G. Toraldo (2001) GRASP with Path Relinking for the Three-Index Assignment Problem. Technical report, AT&T Bell Labs.
- Bastos, M.B. and C.C. Ribeiro (2000) Reactive Tabu Search with Path Relinking for the Steiner Problem in Graphs. Technical report, Department of Computer Science, Catholic University of Rio de Janeiro.
- Battiti, R. (1996) "Reactive Search: Toward Self-Tuning Heuristics," In V.J. Rayward-Smith, I.H. Osman, C.R. Reeves, and G.D. Smith, Editors, *Modern Heuristic Search Methods*, 61–83. Wiley, Chichester.
- Battiti, R. and G. Tecchiolli (1994) "The Reactive Tabu Search," *ORSA Journal on Computing*, 6:126–140.
- Bortz, J. (1999) *Statistik für Sozialwissenschaftler*. Springer, Berlin, 5th edition.
- Campos, V., F. Glover, M. Laguna and R. Martí (1999) An Experimental Evaluation of a Scatter Search for the Linear Ordering Problem. Technical report, Dpto. de Estadística e Investigación Operativa, Universitat de Valencia.
- Campos, V., M. Laguna and R. Martí (1999) "Scatter Search for the Linear Ordering Problem," In D. Corne, M. Dorigo, and F. Glover, Editors, *New Ideas in Optimization*, 331–339, McGraw-Hill, London.
- Cung, V.D., T. Mautor, P. Michelon and A. Tavares (1997) "A Scatter Search based Approach for the Quadratic Assignment Problem," In T. Bäck, Z. Michalewicz, and X. Yao, Editors, *Proceedings of IEEE-ICEC-EPS'97, IEEE International Conference on Evolutionary Computation and Evolutionary Programming Conference*, 165–170, Indianapolis, USA.
- Dammeyer, F., P. Forst, and S. Voß(1991) "On the Cancellation Sequence Method of Tabu Search," *ORSA Journal on Computing*, 3:262–265.
- Dammeyer, F. and S. Voß (1993) "Dynamic Tabu List Management using the Reverse Elimination Method," *Annals of Operations Research*, 41:31–46.
- Duin, C. and S. Voß (1999) "The Pilot Method: A Strategy for Heuristic Repetition with Application to the Steiner Problem in Graphs," *Networks*, 34:181–191.
- Fink, F. and S. Voß (2001) "Efficient Meta-Heuristics Approaches for Ring Load Balancing," In *Proceedings of the 9th International Conference on*

- Telecommunication Systems*, 243–250. Southern Methodist University, Dallas, USA.
- Fleurent, C. and F. Glover (1999) "Improved Constructive Multistart Strategies for the Quadratic Assignment Problem Using Adaptive Memory," *INFORMS Journal on Computing*, 11:198–204.
- Glover, F. (1989) "Tabu Search – Part I," *ORSA Journal on Computing*, 1:190–206.
- Glover, F. (1990) "Tabu Search – Part II," *ORSA Journal on Computing*, 2:4–32.
- Glover, F. (1997) "Tabu Search and Adaptive Memory Programming – Advances, Applications and Challenges," In R.S. Barr, R.V. Helgason, and J.L. Kennington, Editors, *Interfaces in Computer Science and Operations Research: Advances in Metaheuristics, Optimization, and Stochastic Modeling Technologies*, 1–75. Kluwer, Boston.
- Glover, F. (1998) "A Template for Scatter Search and Path Relinking," In J.K. Hao, E. Lutton, E. Ronald, M. Schoenauer, and D. Snyers, Editors, *Artificial Evolution. Lecture Notes in Computer Science*, 1363:3–51. Springer, Heidelberg.
- Glover, F., C.C. Kuo and K.S. Dhir (1998) "Heuristic Algorithms for the Maximum Diversity Problem," *Journal of Information & Optimization Sciences*, 19:109–132.
- Glover, F. and M. Laguna (1997) *Tabu Search*. Kluwer, Boston.
- Glover, F., A. Løkketangen, and D.L. Woodruff (2000) "Scatter Search to Generate Diverse MIP Solutions," In M. Laguna and J.L.G. Velarde, Editors, *Computing Tools for Modeling Optimization and Simulation*, 299–320. Kluwer, Boston.
- Goldberg, D.E. (1989) *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA.
- Greistorfer, P. (1995) "Computational Experiments with Heuristics for a Capacitated Arc Routing Problem," In U. Derigs, A. Bachem, and A. Drexl, Editors, *Operations Research Proceedings 1994*, 185–190. Springer, Berlin.
- Greistorfer, P. (1998a) "Genetic Tabu Search Extensions for the Solving of the Cyclic Regular Max-Min Scheduling Problem," International Conference on Operations Research (OR98), Zürich, Switzerland.
- Greistorfer, P. (1998b) "Hybrid Genetic Tabu Search for a Cyclic Scheduling Problem," In S. Voß, S. Martello, I.H. Osman, and C. Roucairol, Editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, 213–229. Kluwer, Boston.
- Greistorfer, P. (2003a) "Experimental Pool Design - Input, Output and Combination Strategies for Scatter Search," In M.G.C. Resende and J.P. de Sousa, Editors, *Metaheuristics: Computer Decision-Making, Applied Optimization*, 86:279–300. Kluwer, Boston.

- Greistorfer, P. (2003b) "A Tabu Scatter Search Metaheuristic for the Arc Routing Problem," *Computers & Industrial Engineering*, 44(2):249–266.
- Hertz, A. and D. Kobler (2000) "A Framework for the Description of Evolutionary Algorithms," *European Journal of Operational Research*, 126:1–12.
- Hertz, A., G. Laporte, and M. Mittaz (2000) "A Tabu Search Heuristic for the Capacitated Arc Routing Problem," *Operations Research*, 48:129–135.
- Kelly, J.P. and J. Xu (1995) Tabu search and vocabulary building for routing problems. Technical report, Graduate School of Business Administration, University of Colorado at Boulder.
- Laguna, M., R. Martí and V. Campos (1999) "Intensification and Diversification with Elite Tabu Search Solutions for the Linear Ordering Problem," *Computers & Operations Research*, 26:1217–1230.
- Levenshtein, V.I. (1966) "Binary Codes Capable of Correcting Deletions, Insertions, and Reversals," *Soviet Physics – Doklady*, 10(8):707–710.
- Martin, O., S.W. Otto and E.W. Felten (1991) "Large-step Markov Chains for the Traveling Salesman Problem," *Complex Systems*, 5:299–326.
- Merz, P. and B. Freisleben (2000) "Fitness Landscape Analysis and Memetic Algorithms for the Quadratic Assignment Problem," *IEEE Transactions on Evolutionary Computation*, 4(4):337–352.
- Moscato, P. (1993) "An Introduction to Population Approaches for Optimization and Hierarchical Objective Functions: A Discussion on the Role of Tabu Search," *Annals of Operations Research*, 41:85–121.
- Moscato, P. (2001) Memetic Algorithms' Home Page. http://www.densis.fee.unicamp.br/~moscato/memetic_home.html.
- Myers, G. (1999) "A Fast Bit-Vector Algorithm for Approximate String Matching based on Dynamic Programming," *Journal of the Association of Computing Machinery*, 46(3):395–415.
- Osman, I.H. and N. Christofides (1994) "Capacitated Clustering Problems by Hybrid Simulated Annealing and Tabu Search," *International Transactions in Operational Research*, 1:317–336.
- Rechenberg, I. (1972) *Evolutionsstrategie*. Friedrich Frommann, Stuttgart.
- Reeves, C.R. (1993) Genetic Algorithms. In C.R. Reeves Editor, *Modern Heuristic Techniques for Combinatorial Problems*, 151–196. Blackwell, Halsted.
- Reeves, C.R. (2001) Statistical Properties of Combinatorial Landscapes: An Application to Scheduling Problems. Technical report, School of Mathematical and Information Sciences, Coventry University.
- Rego, C. (1998) "A Subpath Ejection Method for the Vehicle Routing Problem," *Management Science*, 44:1447–1459.
- Ribeiro, C.C. and P. Hansen (2002) *Essays and Surveys in Metaheuristics*. Kluwer, Boston.

- Rochat, Y. and É.D. Taillard (1995) "Probabilistic Diversification and Intensification in Local Search for Vehicle Routing," *Journal of Heuristics*, 1:147–167.
- Schaffer, J.D. and L.J. Eshelman (1996) Combinatorial Optimization by Genetic Algorithms: The value of the genotype/phenotype distinction. In V.J. Rayward-Smith, I.H. Osman, C.R. Reeves, and G.D. Smith, Editors, *Modern Heuristic Search Methods*, 85–97. Wiley, Chichester.
- Schwefel, H.P. (1977) *Numerische Optimierung von Computer-Modellen*. Birkhäuser, Basel.
- Semet, F. and É.D. Taillard (1993) "Solving Real-life Vehicle Routing Problems Efficiently Using Tabu Search," *Annals of Operations Research*, 41:469–488.
- Sondergeld, L. and S. Voß (1996) A Star-Shaped Diversification Approach in Tabu Search. In I.H. Osman and J.P. Kelly, Editors, *Meta-Heuristics: Theory & Applications*, 489–502. Kluwer.
- Taillard, É.D. (1991) "Robust Taboo Search for the Quadratic Assignment Problem," *Parallel Computing*, 17:443–455.
- Taillard, É.D., L.M. Gambardella, M. Gendreau and J.Y. Potvin (2001) "Adaptive Memory Programming: A unified View of Meta-heuristics," *European Journal of Operational Research*, 135:1–16.
- Taillard, É.D. and S. Voß (2002) POPMUSIC - Partial Optimization Metaheuristic under special Intensification Conditions. In C.C. Ribeiro and P. Hansen, Editors, *Essays and Surveys in Metaheuristics*, 613–629. Kluwer, Boston.
- Vaessens, R.J.M., E.H.L. Aarts and J.K. Lenstra (1998) "A Local Search Template," *Computers & Operations Research*, 25:969–979.
- Voß, S. (1995) Solving Quadratic Assignment Problems using the Reverse Elimination Method. In S.G. Nash and A. Sofer, Editors, *The Impact of Emerging Technologies on Computer Science and Operations Research*, 281–296. Kluwer, Boston.
- Voß, S. (1996) Observing Logical Interdependencies in Tabu Search — Methods and results. In V.J. Rayward-Smith, I.H. Osman, C.R. Reeves, and G.D. Smith, Editors, *Modern Heuristic Search Methods*, 41–59. Wiley, Chichester.
- Voß, S. (2001) Meta-Heuristics: The State of the Art. In A. Nareyek, Editor, *Local Search for Planning and Scheduling, Lecture Notes in Artificial Intelligence*, 2148:1–23, Springer, Berlin.
- Voß, S. and D.L. Woodruff (2002) *Optimization Software Class Libraries*. OR/CS Interfaces Series. Kluwer, Boston.
- Whitley, D., S. Gordon and K. Mathias (1994) Lamarckian Evolution, the Baldwin Effect and Function Optimization. In Y. Davidor, H.P. Schwefel, and R. Männer, Editors, *Parallel Problem Solving from Nature – PPSN III*, number 866 in Lecture Notes in Computer Science, 6–15. Springer, Berlin.
- Wilson, D.R. and T.R. Martinez (1997) "Improved Heterogeneous Distance Functions," *Journal of Artificial Intelligence Research*, 6:1–34.

- Wolpert, D.H. and W.G. Macready (1997) "No Free Lunch Theorems for Optimization". *IEEE Transactions on Evolutionary Computation*, 1:67–82.
- Woodruff, D.L. (1998) "Proposals for Chunking and Tabu Search," *European Journal of Operational Research*, 106:585–598.

Chapter 19

ADAPTIVE MEMORY PROJECTION METHODS FOR INTEGER PROGRAMMING

Fred Glover

*Leeds School of Business, University of Colorado, Boulder, CO 80309-0419, USA,
fred.glover@colorado.edu*

Abstract: Projection methods, which hold selected variables fixed while manipulating others, have a particularly useful role in metaheuristic procedures, especially in connection with large scale optimization and parallelization approaches. This role is enriched by adaptive memory processes of tabu search, which provide a collection of easily stated strategies to uncover improved solutions during the course of the search. Within the context of pure and mixed integer programming, we show that intensification and diversification processes for adaptive memory projection can be supported in several ways, including the introduction of pseudo-cut inequalities that additionally focus the search. We describe how the resulting procedures can be embedded in constructive multi-start methods as well as in progressive improvement methods, and how they can benefit by the application of target analysis.

Keywords: Adaptive Memory, Restricted Search Space, Cutting Planes, Tabu Search

1. Introduction

An old and recurring idea in optimization is to generate solutions by iteratively holding selected subsets of variables fixed at particular values while varying the values of other variables. The simplex method of linear programming is a familiar special case of this idea, where only a single independent variable, designated nonbasic, is allowed to vary at a time, to identify moves between adjacent vertices of the feasible solution polyhedron. The idea also surfaces in the area of cutting plane methods for integer programming by the strategy called lifting, which assigns the role of variables to cutting plane coefficients. A common form of lifting approach, for

example, successively identifies values to assign to coefficients that are free, given the values presently assigned to fixed coefficients, until a complete cutting plane is generated. More generally, the underlying concept resides at the heart of projection mappings, which are pervasively used in nonlinear and mixed integer programming.

The realm of metaheuristics affords an opportunity to apply this idea in new ways, invoking an enriched set of strategic considerations. Confronted with complex optimization problems where exact methods often fail, the issues of iteratively choosing fixed and free variables, and of controlling the processes for assigning values to variables that are free, become challenging on levels not encountered in more classical settings.

This paper adopts the theme of basing such a solution approach on the principles of tabu search, to create an *adaptive memory projection (AMP) method* for pure and mixed integer programming.

Three main concepts of tabu search provide the starting point for adaptive memory projection methods.

- (1) strongly determined and consistent variables
- (2) intensification/diversification tradeoffs
- (3) persistent attractiveness

While the following discussion will be framed in terms of assigning values to variables, the concepts also apply directly to choosing moves in neighborhood spaces. In this case, “values assigned” may be considered to be the same as attributes imparted to solutions as a consequence of selecting a move. The basic approach can be viewed from the perspective of decomposition, since each projection “breaks apart” the problem by partitioning the variables into the fixed and free classes.

1.1 Strongly Determined and Consistent Variables

The notion of strongly determined and consistent variables (Glover, 1977) is to keep track of variables that receive particular value assignments (or ranges of assignments) within some interval of frequency over high quality solutions. The concept also applies to keeping track of such assignments in solutions where changing these assignments would significantly modify the structure or quality of the solutions.¹ To identify and take advantage of such variables involves:

¹ We emphasize that in the metaheuristic context, a solution is defined relative to the neighborhoods employed. Infeasible solutions produced from neighborhoods that admit such alternatives are as relevant as any others. This is notably true in applications where solutions are produced by strategic oscillation (which may purposely construct infeasibilities) and by solving surrogate or Lagrangean or LP relaxations.

- (a) Segregating sets of good solutions over which the variables and their assignments are identified (including the use of clustering to achieve this segregation);
- (b) Applying measures of frequency to determine which variables qualify as *consistent* (Such measures underlie the type of frequency memory used in intensification strategies in tabu search, noting that the meaning of “consistency” in the present case is determined by reference to a subset of good solutions.)
- (c) Applying criteria of change to determine which variables qualify as *strongly determined*. (This relates to the notion of influence in tabu search, which has an important role in diversification strategies.)

The variables identified by the foregoing mechanisms are exploited by constraining them to receive their preferred values (or lie in their preferred ranges), and then searching more intensively over the remaining variables. As expressed in Glover (1977), the key premises underlying the approach are:

- such variables are likely to receive their preferred values in other high quality solutions, including optimal and near-optimal solutions;
- constraining the variables to their preferred values creates a “combinatorial implosion” effect that accelerates the solution of the remaining problem;
- once selected variables are constrained in this manner, then other variables may also become strongly determined or consistent, allowing them to be exploited similarly.

1.2 Intensification/Diversification Tradeoffs

Although strongly determined and consistent variables were initially proposed for application in an intensification context, the tabu search emphasis on balancing intensification with diversification suggests the possibility of augmenting the use of these variables by periodically imposing restrictions or incentives to drive the solution into new regions. A coordinated treatment of such considerations is crucial for an adaptive memory projection method, and we subsequently discuss key elements for doing this in detail. It is to be noted that the idea of generating solutions by iteratively holding subsets of variables fixed at particular values while varying the values of other variables is also known in the constraint programming community as “Large Neighborhood Search” (LNS), whose terminology was introduced in Shaw (1998). Still more recently, the theme of identifying and exploiting variables by the criteria of being strongly determined and/or consistent, although again making use of different terminology, appears as a fundamental component of the framework proposed in Ahuja et al. (2002) for large scale neighborhood search, and also appears in

the innovative metaheuristic design proposed by Danna et al. (2003) for mixed integer programming problems.

1.3 Persistent Attractiveness

The notion of persistent attractiveness (e.g., Glover, 2000) underscores the utility of identifying assignments of values to variables that receive high evaluations, but which are not executed by a search method (as where, for example, other assignments may receive still higher evaluations). Assignments that often receive high evaluations are *persistent attractive*, and deserve to be the focus of strategies that impose such assignments just as if the variables qualified as “consistent” in the sense identified earlier. It is also relevant to keep track of variables that are *conditionally attractive*, i.e., that may rarely receive high evaluations, but that receive very high evaluations at critical points. These represent a form of strongly determined variables, because the fact that they rapidly became less attractive is a sign that, had they been given their high-evaluation assignments, they would have been disposed to change the structure of the solution in a significant way.

2. Fundamentals of an Adaptive Memory Projection Method

Adaptive Memory Projection Methods have several versions. We begin by examining a version that is easy to implement, and that employs the proposal of coordinating TS with the application of exact optimization procedures over sub-problems that are kept small enough that such problems can be solved efficiently by these procedures. An outline of the AMP method is as follows.

2.1 Overview of an Adaptive Memory Projection Approach

Step1. (Initiation) Generate a starting solution. (Alternatives for doing this are discussed later.)

Step2. (Search Phase) Apply a heuristic search for some limited number of iterations. Keep track of the variables whose values are changed, and also keep track of variables (and their assignments) that qualify as persistently and conditionally attractive. (Changes in assignments may be accomplished by multiple types of moves.)

*Step3. (Referent-Optimization Phase)*² Choose a subset of the variables recorded in Step 2 that were changed or that qualified as persistently or conditionally attractive. Apply an exact method to solve the *referent-problem* that allows these variables to be free, while remaining problem variables are held fixed at the values they received in the best solution found by the search in Step 2.

Step4. (Re-launch the Search) Using recency and frequency memory, and imposing associated restrictions (as subsequently identified), perform a new heuristic search starting from the solution obtained in Step 3.

Step5. (Diversification and Renewed Solution Pass) Drive the search into a new region, using longer term memory and standard TS diversification processes, and repeat Steps 2-4.

We amplify the steps of the preceding outline by several key observations.

- (1) A restricted form of the heuristic search of Step 2 can be applied by keeping the number of iterations small enough that all variables recorded (changed or identified as attractive) can be selected to be free in Step 3. If the heuristic search is performed for a larger number of iterations, then a screening step must be performed to select a preferred subset of variables to be treated as free. (The operation of holding certain variables fixed in Step 3 can of course be exploited by temporarily “reducing” the problem, i.e., by adjusting the constraint requirements and dropping the fixed variables from consideration.)
- (2) All variables changed in Step 2 are automatically included among candidates to become free variables in the next referent-optimization of Step 3; i.e., if a variable changes and then changes again to return to its original value, it still qualifies to be selected as a free variable of the next referent-optimization.
- (3) An advanced and intensive heuristic search may be performed to carry out Step 3 in place of an exact procedure. In such a process, and in successive applications of Step 3, such an approach may keep a longer term frequency memory $f(j)$ that records how often each variable x_j is a free variable for the referent-optimization step. Then the inclusion of x_j as a free variable can be penalized according to the size of $f(j)$ (e.g., as compared to the average $f(j)$ value), to achieve greater diversification.
- (4) Each new application of the improving heuristic at Step 4, after running the exact method at Step 3, generates new evaluations for assigning the variables particular values. Variables that receive the highest evaluations to be changed go on the list of variables to be considered as free variables

² This terminology comes from Glover and Laguna (1997), where this type of approach is viewed within a broader framework called “referent-domain optimization.” See also Mautor and Michelon (1997, 2001).

on the next execution of the exact method. These variables include both those that were selected to be changed and those that were not selected to be changed (but which nevertheless were strong candidates to be selected, e.g., that were strongly or persistently attractive).

The policy of discouraging the selection of variables that were free on the most recent preceding referent-optimization should also affect the evaluations on the current heuristic solution step. Thus, if a variable is tabu to change, its evaluation is considered to be unattractive, unless the associated move satisfies an aspiration criterion permitting its inclusion on the list of candidates to become free variables.

- (5) A diversification effect can be introduced within Step 3 by including one or more *pseudo-cuts* that assure a solution will be found that is different from the one found by the preceding heuristic solution effort.³

To illustrate, consider just two solutions, the one that initiates the search of Step 2 (or Step 4) and the one that is the best solution found during this step. (Subsequent comments address the situation where these solutions may be the same.) Let x_1, \dots, x_p be the variables whose values were increased, and let y_1, \dots, y_q be the variables whose values were decreased, in going from the initial solution to the best solution. Also, let x'_1, \dots, x'_p and y'_1, \dots, y'_q be the values of these variables in the initial solution.

Let *Increase* denote the amount by which the x variables increased and let *Decrease* denote the amount by which the y variables decreased. (Both of these are positive numbers.) For the values given to x and y in the best solution found during the search, the variables therefore satisfied:

$$(x_1 - x'_1) + (x_2 - x'_2) + \cdots + (x_p - x'_p) = \text{increase}$$

and

$$(y'_1 - y_1) + (y'_2 - y_2) + \cdots + (y'_q - y_q) = \text{Decrease}$$

The pseudo-cut that compels these variables to produce a change at least k less than the sum of *Increase* and *Decrease* is:

³ Pseudo-cuts (inequalities that may not be satisfied by optimal solutions) are also used within tabu search methods for mixed integer programming as a basis for generating moves. The tabu search processes that allow the foundations of previous moves to be selectively discarded, and that accordingly allow invalid or unproductive inequalities to be removed, prove especially useful in this context. (See the references cited in Chapter 6 of Glover and Laguna, 1997, particularly in connection with tabu branching.)

$$(x_1 - x'_1) + \cdots + (x_p - x'_p) + (y'_1 - y_1) + \cdots + (y'_q - y_q) \leq \quad (A)$$

Increase + Decrease – k

A possible value for k can be $(\text{Increase} + \text{Decrease})/2$, for example, although k should normally be restricted from being very large.) In the 0-1 context, $\text{Increase} = p$ and $\text{Decrease} = q$.

This same idea can be applied to include additional variables in (A), by allowing variables whose values do not change to be classified either as producing a 0 increase or a 0 decrease. Assigning a “no-change” variable to the *Increase* set (treating it as an x variable) tends to strengthen (A), while assigning it to the *Decrease* set (treating it as a y variable) tends to weaken (A). In either case, the expanded inequality is binding so long as k is positive.⁴

The pseudo-cut (A) can also be used in additional strategic ways. Specifically, even without first applying a search to change the initial solution, we can hypothesize that a selected subset of values may increase (or stay the same) and another may decrease (or stay the same), to identify the x and y variables. Then, an inequality for guiding the search can be generated by postulating values for *Increase* and *Decrease*, and choosing an associated value for k . More simply, we can directly choose a value $k^* = \text{Increase} + \text{Decrease} - k$ that represents the constant term (right hand side) of (A). An extreme example of this occurs for the case of 0-1 variables by specifying the x variables to be those for which $x_j = 0$ and the y variables to be those for which $y_j = 1$. Then the inequality for a chosen value of k^* becomes the same as the “local branching” inequality of Fischetti and Lodi (2002). The AMP framework that includes (A) therefore offers a set of strategies that subsume the local branching scheme, and suggests the merit of integrating adaptive memory projection with local branching ideas.⁵

- (6) In the situation where the heuristic search of Step 2 (or Step 4) does not identify a solution that is better than the starting solution for this step, the following approach can be used to introduce one or more pseudo-cuts in Step 3 to achieve a diversification effect. This type of approach is useful as well for the case where the starting and best solutions found in Step 3 are the same, and applies particularly to 0-1 problems.

⁴ The use of weakened versions of (A) provide interesting possibilities for making the search more flexible.

⁵ A more general approach that goes beyond local branching, and that likewise invites integration with adaptive memory projection, is provided by Surrogate Branching, as described in Glover, Fischetti and Lodi (2003).

Redefine x_1, \dots, x_p to be variables equal to their lower bounds (0), and y_1, \dots, y_q to be variables at their upper bounds (1) in the starting solution, where we now restrict consideration to those variables that did not change their values, i.e., whose values are the same in the starting solution and in the best solution obtained. (Hence, if the best solution obtained is no different from the starting solution, all variables are candidates to be considered.) We further restrict attention to variables that are included among the free variables for the next referent-optimization iteration. (For example, these can consist of variables that looked attractive to change at some point, although they were not changed. Or they could have been changed but then were changed back.)

Then choose small positive integer values of *Increase* and *Decrease* (where $Increase < p$ and $Decrease < q$) and impose one or both of the pseudo-cuts

$$x_1 + x_2 + \dots + x_p \geq Increase \quad (B1)$$

$$(1 - y_1) + (1 - y_2) + \dots + (1 - y_q) \geq Decrease \quad (B2)$$

An alternative is to impose the pseudo-cut inequality

$$\begin{aligned} x_1 + x_2 + \dots + x_p + (1 - y_1) + (1 - y_2) + \dots + (1 - y_q) \geq \\ Increase + Decrease \end{aligned} \quad (B3)$$

The upper bound of 1 in these inequalities can be replaced by a more general upper bound value U_j applicable to the associated variable.

We can expand the range of variables included in the foregoing inequalities by incorporating those whose values change, analogous to the previously indicated expansion of (A). We can similarly use these inequalities in additional strategies, by allowing alternative interpretations of the x and y variables.

These same ideas can also be used to assure that new solutions generated in Step 3 are driven away from other solutions previously found, not just those of the immediately preceding execution of Step 2 (or 4).⁶

⁶We add a brief comment to amplify on the use of “persistently attractive” variables in the situation where Step 2 (or Step 4) fails to find an improved solution. In this case, free variables will be chosen from among those that had attractive

- (7) To apply Step 4 by re-launching the search, a natural strategy is to apply tabu restrictions to insure that at least one of the variables whose value is changed on any of the first several moves of the search must come from outside the set of variables that were free in the referent-optimization phase of Step 3. In other words, all free variables from this immediately preceding phase are treated as "tabu attributes" whose tenure lasts for a specified number of iterations. (This tenure may be as small as 2-4 iterations, or may be somewhat larger, e.g., 8 – 10 iterations, in a more strongly diversifying approach. Periodic shifts between tenure sizes are relevant, as in an adaptive tenure design.) A move is designated tabu if all of the variables it changes are tabu. A stronger tabu restriction can be imposed for the first few iterations (e.g., 1 to 3 iterations) by designating a move to be tabu if any of the variables it changes are tabu.
- (8) For 0-1 mixed integer programming problems, a strategy to establish greater diversification by Step 4 occurs by using "diversity thresholds." Let $x_j, j \in \text{Free}$ identify the set of variables that were free in the execution of Step 3 that immediately precedes the execution of Step 4, and let $x_j^*, j \in \text{Fixed}$, identify the associated set of fixed variables. Also, let x_j^* denote the values of these variables in the optimal solution obtained by Step 3 (where x_j^* is simply the fixed value of x_j in the case of a fixed variable).

evaluations during the pass (or that belonged to moves having attractive evaluations). Once the method reaches an improving phase, whether or not a solution better than the starting solution is obtained, the moves from this phase may be considered as sources for new free variables. Typically, not all moves that receive high evaluations (relative to alternative moves) will be chosen during an improving phase. Consequently, there can be more variables that qualify as "attractive" than those that belonged to moves that were chosen. (Some of these may have belonged to more than one move on the same iteration, and some may have belonged to different moves on different iterations.) The attractiveness of a variable may be measured by reference to the attractiveness of the moves that contain it. A variable that belongs to one of the most highly attractive moves at a given iteration, but then which vanishes from the attractive category, is also important to consider.

Diversity Threshold Procedure 1

Let $Free(1) = \{j \in Free : x_j^* = 1\}$ and $Free(0) = \{j \in Free : x_j^* = 0\}$.

Record $FreeSum(1) = \sum_{j \in Free(1)} x_j$ and $FreeSum(0) = \sum_{j \in Free(0)} x_j$.⁷

To generate a solution that departs from the solution of Step 3 we seek to avoid the situation where $FreeSum(1) = |Free(1)|$ and $FreeSum(0) = 0$. Define $FreeDiversitySum = |Free(1)| - FreeSum(1) + FreeSum(0)$. Identifying the number of variables $x_j, j \in Free$, such that $x_j \neq x_j^*$. Then we want to assure $FreeDiversitySum \geq FreeDiversityThreshold$ where $FreeDiversityThreshold = 2$ or 3 , etc. (larger for more diversification). This can be done by defining a move to be tabu if it will make $FreeDiversitySum$ fall below $FreeDiversityThreshold$ (For most simple types of moves, the condition only needs to be checked when $FreeDiversitySum$ drops as low as $FreeDiversityThreshold + 1$.)

Diversity Threshold Procedure 2

Let

$Fixed(1) = \{j \in Fixed : x_j^* = 1\}$

$Fixed(0) = \{j \in Fixed : x_j^* = 0\}$

Define $FixedSum(1)$ and $FixedSum(0)$ in the apparent way, and then define $FixedDiversitySum$ relative to these preceding values, also in the apparent way. We seek to assure

$$FixedDiversitySum \geq FixedDiversityThreshold$$

for a suitable (relatively small) value of $FixedDiversityThreshold$.

⁷These sums can be easily updated at each move by checking if a variable changed by the move is in $Free(1)$ or $Free(0)$. For example, to facilitate the update, keep an array $FreeMember(j)$ that receives the value 1 for $j \in Free(1)$, the value 0 for $j \in Free(0)$ and the value -1 for $j \in Fixed$.

This second approach is implicitly the basis for the tabu rule discussed in (7). That is, the tabu rule of (7) insures that *FixedDiversitySum* grows by at least 1 on each iteration, for some beginning number of iterations. Variants of these approaches can readily be constructed for problems involving integer variables other than 0-1 variables.

Combined Diversity Threshold Procedure

The two foregoing procedures can be combined to give a less restrictive approach. The combined procedure defines

$$\text{DiversitySum} = \text{FreeDiversitySum} + \text{FixedDiversitySum}$$

Then it is only necessary to select a (relatively small) value *DiversityThreshold* and require that every move assures

$$\text{DiversitySum} \geq \text{DiversityThreshold}$$

We can also stipulate that initial moves of the method are tabu unless they increase *DiversitySum* by at least 1 at each iteration – i.e., requiring that the initial moves must increase *DiversitySum* by at least 1 until *DiversitySum* reaches some minimum value (for example, 1 or 2 more than *DiversityThreshold*). This approach offers an alternative to the tabu rule of (7) that may possibly yield better results.

- (9) The re-launched heuristic search of Step 4 may not find any improving moves to begin, and should operate by the usual tabu search design of selecting best admissible (non-tabu) moves from an intelligently generated candidate list, even if the selected moves cause the solution to deteriorate. The heuristic must be run for enough iterations to give a basis for selecting a proper number of new free variables for the next referent-optimization. As it proceeds, the heuristic continues to create new tabu restrictions in a standard way, to avoid cycling.
If the method is in an improving sequence at the point where enough new free variables can be selected for the next referent-optimization phase, the sequence is allowed to continue its improvement until reaching a new local optimum. (This may be viewed as a special aspiration criterion that accepts all improving moves at this point.)
- (10) To spur additional diversification, a “weak tabu condition” can be implemented whenever improving moves exist by discouraging (but not preventing) the choice of improving moves that change the value of some variable to a value it had in the referent-solution. Other tabu restrictions

continue to operate normally in this phase. When a non-improving stage is entered during the search, then the weak tabu condition can be strengthened for a small number of iterations (2-4) to prevent moves that change the value of any variable back to the value it received in the referent-solution.

- (11) The preceding approach can be applied by eliminating the use of the heuristic method in Steps 1 and 4. In other words, the procedure can simply use an adaptive memory design from a variant of TS that progressively selects different subsets of variables to be free, and then immediately undertakes the next referent-optimization phase in Step 3. In essence, the heuristic approach is not precisely discarded, but it only identifies variables that may be considered conditionally attractive as a basis for constituting the new set of free variables. (These variables are evaluated as candidates to receive new assignments, but no steps are performed to actually execute such assignments.) The conditionally attractive evaluation can be made from information provided by the exact method itself. For example, if the exact method solves linear programming sub-problems, as in a B&B approach, then a form of sensitivity or post-optimizing “look-ahead” analysis may be performed to help identify new candidates to constitute the next set of free variables.

More general TS diversification approaches can of course also be incorporated into Step 5. An adaptive memory projection approach can readily be embedded within a constructive method for generating a solution, for the case of diversification methods based on re-starting. Consequently, such a projection method can be used to drive a multi-start method, and can also be used (as above) as part of a process that progressively amends a current solution rather than constructing (or re-constructing) a solution from scratch.

3. Constructive Variant of an Adaptive Memory Projection Method

A constructive variant of an adaptive memory projection method, to be incorporated in a multi-start procedure, can be briefly sketched as follows by making use of ideas already discussed.

3.1 Constructive (Multi-Start) Procedure

Step 1. Generate a solution constructively (as by a strategy for successively choosing values to assign to variables), keeping track of persistently and conditionally attractive value assignments, as well as assignments actually made.

Step 2. Select a subset of variables that include some of those selected to receive specific values during the construction process and also some of those that were persistently and conditionally attractive. Then apply an exact method to solve the sub-problem that allows these selected variables to be free, while remaining problem variables are held fixed.

Step 3. Using an adaptive memory design to avoid duplicating sets of variables recently chosen, return to Step 3 to select a new subset of variables and again apply an exact method with these variables treated as free.

Early stages of such an approach can select subsets of variables in Step 2 that were chosen successively to receive their assigned values, and that first became attractive (if they qualify as persistently or conditionally attractive) during this sequence of assignments. Later applications of the step can reasonably mix the subsets to include variables assigned in non-consecutive orders. Once enough values are changed, the method reverts to become the same as the method originally described (which does not specifically make reference to constructive processes).

The observations of Section 2 are relevant for the constructive multi-start variant as well.

4. Adaptive Memory Projection and Target Analysis

Some concluding comments are relevant for applying the AMP approach, when a basic heuristic is used to identify a particular region to explore, and then another more advanced solution procedure is used to examine this region in depth.

Consider the use of a search neighborhood in which the moves consist of flipping values of 0-1 variables. Assume we have a corresponding MIP formulation of the problem to be solved, so that an exact 0-1 MIP method can take the role of the advanced method. Then the AMP approach can be used in a couple of straightforward ways as follows.

(1) First apply the current heuristic method. Maintain a record of k variables (e.g., $k = 20$)⁸ that have changed their values in the most recent moves prior to reaching a local optimum, or prior to reaching some other intervention point that seems useful. (Fewer than k moves may be involved in identifying such variables, if a move modifies more than a single variable at a time.)

Also record an additional k variables that received high evaluations to change their assigned values, but not quite high enough to result in choosing the variables to receive such a changed assignment. Then apply the AMP approach by solving an MIP sub-problem over these

⁸ In some applications, k may usefully be selected to be much larger, as illustrated by the work of Danna and Perron (2003).

$2k$ variables, which are allowed to be free while the other variables are fixed at the values assigned by the local optimum. This gives a solution at least as good as the one the heuristic found. (Instead of basing the process on a record of most recently changed variables, it can be based on a record of other critical changes following earlier suggestions.)

- (2) Further exploit the preceding strategy by means of target analysis (Chapter 9 of Glover and Laguna, 1997), to learn how to improve the basic heuristic. In this case the AMP approach is applied by allowing more variables to be included in the sub-problem solved by the MIP method than would normally be desirable to include in this problem. (For example, this might involve choosing up to $4k$ variables for the sub-problem solved by the MIP method.)

Then, target analysis can operate by examining the cases where the MIP gives a better solution than found by the heuristic. These cases yield information about how to improve the heuristic by changing its choice rules to make better decisions.

To illustrate, it can be valuable to know if the variables that allow the MIP method to get better results are primarily those whose values were changed by the heuristic (a) multiple times, (b) at least once, or (c) never, but which received a high evaluation (or high average evaluation) in favor of receiving new value assignments in spite of not being selected for such a purpose. This knowledge provides a foundation for tests to further pinpoint the characteristics of the most important variables.

It is similarly relevant to identify variables that receive different values in the MIP solution than in the heuristic solution, thereby making it possible to investigate whether the heuristic evaluations should be amended by reference to associated historical information.

Such information may be embodied in the frequency that a variable receives an evaluation at a certain basic level in favor of being assigned a given value 0 or 1, and also embodied in the frequency that the variable receives an evaluation at a higher level in favor of such a value assignment. Target analysis is then used to identify whether some mix of these two types of frequencies supports the choice $x_j = 0$ or $x_j = 1$.

In the case where many alternative choices may exist to be evaluated, target analysis can be kept manageable by focusing on a selected subset of key variables. These may be a collection of variables that receive the highest evaluations according to particular decision rules to be analyzed, or they may be divided among variables whose evaluations by such rules predict appropriate values to be assigned and those whose evaluations predict inappropriate values. In this type of process multiple decision rules can be analyzed simultaneously, rather than requiring each one to be run separately. Thus, it may be discovered that one rule is more accurate under certain

conditions and another is more accurate under other conditions – an outcome that would not be discovered except by the use of target analysis, since in standard testing only the quality of the final solution would be known, without providing an ability to assess the merit of component choices (and to do so in relation to the settings in which these choices arise).

It is important to realize that clues of the type indicated only have to give approximate rules for identifying good values to be assigned to the variables. Suppose, for instance, the MIP solution method is applied to a larger subproblem than normally would be considered, as in the approach described in (1), and gives values to 10 variables that are different than found by the heuristic. A rule may be regarded as only "halfway accurate" if it instead selects 20 variables as candidates to change their values. But if these 20 variables include the 10 that the MIP method determined should be changed, then the rule is extremely powerful – because the MIP method can be applied to a 20 variable problem and still do as well as applying it to a considerably larger problem. In short, if the rule is used in conjunction with the AMP method, then the method will always find the desired solution. Clearly, a good AMP method can result from a rule that is not this effective. But target analysis can be a useful supplement in the design stage, in order to devise a rule for the AMP method that performs better than those that might otherwise be employed.

References

- Ahuja, R.K., O. Ergun, J.B. Orlin and A.P. Punnen (2002) "Survey of Very Large-Scale Neighborhood Search Techniques," *Discrete Applied Mathematics* 123:75–102.
- Danna, E. and L. Perron (2003) Structured vs. Unstructured Large Neighborhood Search: A Case Study on Job-Shop Scheduling Problems with Earliness and Tardiness Costs, ILOG Technical Report, ILOG, S.A.
- Danna, E., E. Rothberg and C. Le Pape (2003) Exploring Relaxation Induced Neighborhoods to Improve MIP Solutions. ILOG Technical Report, ILOG, S.A.
- Fischetti, M. and A. Lodi (2002) "Local Branching," Research Report, DEI, University of Padova and DEIS, University of Bologna.
- Glover, F. (1977) "Heuristics for Integer Programming Using Surrogate Constraints," *Decision Sciences*, 8(1):156–166.
- Glover, F. (2000) Multi-Start and Strategic Oscillation Methods – Principles to Exploit Adaptive Memory. Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research, M. Laguna and J.L. Gonzales Velarde, eds., Kluwer Academic Publishers, 1–24.
- Glover, F. and M. Laguna (1997) *Tabu Search*, Kluwer Academic Publishers.

- Glover, F., M. Fischetti and A. Lodi (2003) Surrogate Branching Methods for Mixed Integer Programming. Report HCES-04-03, Hearin Center for Enterprise Science, University of Mississippi.
- Mautor, T. and P. Michelon (1997) Mimausa: A New Hybrid Method Combining Exact Solution and Local Search. MIC'97, 2nd Methaheuristics International Conference, Sophia Antipolis.
- Mautor, T. and P. Michelon (2001) Mimausa: An Application of Referent Domain Optimization. Technical Report, Laboratoire d'Informatique d'Avignon.
- Shaw, P. (1998) Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In M. Maher and J.F. Puget, eds., Proceeding of CP '98, Springer-Verlag, 417–431.

Chapter 20

RAMP: A NEW METAHEURISTIC FRAMEWORK FOR COMBINATORIAL OPTIMIZATION

César Rego

Hearin Center for Enterprise Science, School of Business Administration, University of Mississippi, University, MS 38677, USA, crego@bus.olemiss.edu;

Abstract: We propose a new metaheuristic framework embodied in two approaches, Relaxation Adaptive Memory Programming (RAMP) and its primal-dual extension (PD-RAMP). The RAMP method, at the first level, operates by combining fundamental principles of mathematical relaxation with those of adaptive memory programming, as expressed in tabu search. The extended PD-RAMP method, at the second level, integrates the RAMP approach with other more advanced strategies. We identify specific combinations of such strategies at both levels, based on Lagrangean and surrogate constraint relaxation on the dual side and on scatter search and path relinking on the primal side, in each instance joined with appropriate guidance from adaptive memory processes. The framework invites the use of alternative procedures for both its primal and dual components, including other forms of relaxations and evolutionary approaches such as genetic algorithms and other procedures based on metaphors of nature.

Keywords: RAMP, Scatter Search, Surrogate Constraints, Lagrangean Relaxation, Cross-Parametric Relaxation, Subgradient Optimization, Adaptive Memory, Metaheuristics, Combinatorial Optimization.

1. Introduction

Adaptive memory programming (AMP) has been the source of numerous important developments in metaheuristics in the last decade. The term refers to the appropriate integration of memory structures to effectively explore the solution space in optimization algorithms. Because AMP is the foundation of tabu search (TS), which appeared as the first method specially designed to exploit adaptive memory, the terms TS and AMP have often been used interchangeably. However, more recently the principles of AMP as

introduced in tabu search have likewise been used to enhance other approaches as in the creation of hybrid algorithms that incorporate tabu search components. Many important examples exist of methods that integrate genetic algorithms and evolutionary computation methods with adaptive memory, but perhaps the most prominent example and successful use of adaptive memory outside of tabu search are the recent developments in scatter search and its generalization, the path-relinking approach.

On the other hand, relaxation techniques have been widely used in combinatorial optimization to provide bounds for tree search procedures as well as to produce heuristic algorithms. These techniques are based upon the solution of an auxiliary (or relaxed) problem derived from the original by fully dropping or diminishing the restrictiveness of some constraints. Since a feasible solution to the original problem is usually not available from the solution of the relaxed problem, constructive and local search procedures are typically used to search for complementary feasible and possibly improved solutions. The effectiveness of the improvement methods employed is critical for the performance of these relaxation-based approaches. Yet surprisingly, only relatively rudimentary forms of search strategies can be found in the literature on these methods, and are usually limited to descent algorithms involving greedy strategies and very simple neighborhoods. This is a critical gap in the current state-of-the-art of approximation algorithms since it is well established that in many cases in the history of optimization dual approaches outperform their primal counterpart. Therefore, these findings provide a significant motivation for the next generation of metaheuristics to take advantage of duality as well as the creation of metaheuristic approaches that can exploit the primal-dual relationships as a means for bridging the duality gap in combinatorial optimization.

This paper takes a first step toward integrating these two key developments by proposing a unified framework for the design of dual and primal-dual metaheuristics that take full advantage of adaptive memory programming. Such an initiative is influenced and reinforced by the following remarks.

An important conceptual difference exists between local search methods and relaxation techniques. Local search methods are characteristically *restrictive* approaches that confine the solution space to the solutions that are made accessible by the neighborhood structure employed. By contrast, relaxation techniques enlarge the solution space to include infeasible solutions that fall within the boundaries of the *relaxation* utilized. Ultimately, even in cases where local search methods allow for crossing feasibility

boundaries as a form of strategic oscillation, neighborhood search and relaxation heuristics are conceptually complementary approaches for handling optimization models and generating search directions.

Local search methods are typically *primal* approaches in the sense that they explore the solution space by exploiting the context of the original problem to be solved. By contrast, relaxation heuristics work by primarily solving a distinct problem associated with the type of the relaxation. In particular, Lagrangean and surrogate constraint relaxation approaches are based on optimizing the Lagrangean or the surrogate *dual* problem, respectively, and then finding a complementary feasible primal solution. However, when it comes to integer programs the optimization of the dual problem relaxation does not necessarily result in a complementary feasible primal solution, and in many cases these methods do not converge in a reasonable amount of time, thus producing solutions with large duality gaps.

Our fundamental premise is that solving the Lagrangean or the surrogate problem affords relevant insights for the creation of adaptive memory structures by gathering information that cannot be obtained by primal based approaches. Consequently, a method that effectively gathers information from the primal and dual sides fulfills the concept of adaptive memory programming and suitably may provide a unified framework for solving difficult combinatorial optimization problems.

We propose two metaheuristic approaches based on these observations. The first approach couples surrogate and Lagrangean relaxations with tabu search and path-relinking as a means to create a Relaxation AMP (RAMP) method. In addition, we introduce a special relaxation technique giving rise to a cross-parametric relaxation method (CPRM) that combines the notion of *parametric subgradients* from surrogate constraint duality theory (Glover 1975) with the Lagrangean/Surrogate relaxation technique introduced in Narciso and Lorena (1999). More precisely, CPRM combines Lagrangean and surrogate relaxations by using a Lagrangean based subgradient search within a surrogate constraint framework to generate good surrogate constraints.

Although the RAMP approach constitutes a stand alone metaheuristic, we integrate this approach with scatter search and path-relinking to create a Primal-Dual metaheuristic approach, called PD-RAMP, which constitutes a major contribution of this paper. While RAMP is primarily a dual approach, PD-RAMP exploits the primal-dual relationships more thoroughly. For economy of terminology, we will refer to both forms of the RAMP method

(the basic RAMP method and PD-RAMP) simply as RAMP, without added qualification, whenever no specific details are necessary regarding the level of the approach utilized.

The remainder of this paper is organized as follows. Section 2 reviews classical relaxation procedures that are deemed relevant in the context of this research and discusses the cross-parametric relaxation method. Section 3 describes the dual and primal-dual components of the RAMP method. Section 4 presents concluding remarks and perspectives for further developments.

2. Fundamentals of Cross-Parametric Relaxation

Surrogate constraints (SC) and Lagrangean relaxation (LR) form the building blocks of the cross-parametric relaxation method (CPRM).

Throughout this paper we define specific problems by reference to their value functions. Following this convention, consider the general 0-1 integer linear programming problem P defined by

$$v(P) = \text{Min} \{cx \mid Ax \leq b, Dx \leq e, x \in \{0,1\}\}$$

and assume that the constraints $Ax \leq b$ are the ones that make the problem difficult to solve (i.e., the form of the problem that excludes these constraints can be solved efficiently by known methods).

Lagrangean relaxation

The Lagrangean relaxation of P is obtained by *dualizing* the constraints $Ax \leq b$ to form the integer programming problem LP^λ defined by

$$v(LP^\lambda) = \text{Min} \{cx - \lambda(Ax - b) \mid Dx \leq e, x \in \{0,1\}\},$$

where λ represents a nonnegative vector of multipliers with one component for each row of A . The dual problem in this case consists of finding such a λ that maximizes the value $v(LP^\lambda)$. A solution to LP^λ for any given vector $\lambda \geq 0$ provides a lower bound on the primal objective function $v(P)$ – a result known as *weak Lagrangean duality*. Whenever there is a *duality gap*¹, i.e., whenever the optimum values for the primal and dual problems are not the same, it is not possible to determine optimality for an integer linear programming problem by means of solving the dual. *Strong Lagrangean*

¹ The *duality gap* is sometimes called the *integrality gap* when referring to the dual of the linear programming relaxation of P .

duality, that gives rise to the *optimality conditions* for an Lagrangean primal-dual solution², includes *complementary slackness conditions*—i.e. for a given λ a primal solution x must satisfy $\lambda(Ax - b) = 0$. If the optimal primal dual solution fails to satisfy the complementary slackness conditions, the solution is a *near-optimal solution* to P with a duality gap $v(P) - v(LP^\lambda) = \lambda(Ax - b)$. Determining an optimal primal-dual solution underlies finding the optimal multipliers for LP^λ that result from solving the Lagrangean dual of the primal problem P , which may be more explicitly defined by

$$v(D^\lambda) = \text{Max} \{v(LP^\lambda) \mid \lambda \geq 0\}.$$

Although the problem is only restricted by non-negativity constraints that do not offer difficulties, it contains a nonlinear function with an implicit Maxmin objective that is rather costly to evaluate. In fact the problem is equivalent to

$$v(D^\lambda) = \text{Max}_{\lambda \geq 0} \text{Min}_{Dx \leq e} \{cx - \lambda(Ax - b) \mid x \in \{0,1\}\}.$$

Therefore, to determine $v(LP^\lambda)$ for a given λ , one must solve the Lagrangean problem LP^λ . A useful property is that $v(LP^\lambda)$ is a continuous piecewise linear concave function of λ (represented by a finite number of linear functions $cx - \lambda(Ax - b)$, one for each Lagrangean dual solution). The function is differentiable except at points λ where LP^λ has alternative optimal solutions. Subgradient optimization, which extends the gradient concept to nondifferentiable concave/convex functions, has proved effective to find optimal or near-optimal Lagrangean dual solutions. Likewise the method can be used to generate dual solutions for other types of relaxations as will be discussed later.

Surrogate constraint relaxation

A surrogate relaxation of P consists of replacing the constraints $Ax \leq b$ by a nonnegative linear combination of these constraints weighted by a vector of multipliers w . This replaces the constraints $Ax \leq b$ by a single *surrogate constraint*, $w(Ax - b) \leq 0$, thus producing the *surrogate problem* SP^w defined by

² A Lagrangean primal-dual solution is an optimal solution for LP^λ that also satisfies the primal $Ax \leq b$ constraints.

$$v(SP^w) = \text{Min} \{cx \mid w(Ax - b) \leq 0, Dx \leq e, x \in \{0,1\}\}.$$

Since SP^w is a relaxation of P (for w nonnegative), $v(SP^w)$ cannot exceed the optimal objective function value for P and it approaches this value more closely as $w(Ax - b) \leq 0$ becomes a more accurate representation of the polyhedron defined by the constraints $Ax \leq b$. The associated surrogate dual is the one that yields

$$v(D^w) = \text{Max} \{v(SP^w) \mid w \geq 0\}.$$

Although surrogate constraint relaxation has not been employed as widely as Lagrangean relaxation, it is theoretically more powerful—the surrogate duality gap is always at least as good, and often better than, the Lagrangean duality gap. In fact, the Lagrangean dual value can be equal to the surrogate dual value only if the complementary slackness conditions hold for every Lagrangean multiplier (Greenberg and Pierskalla 1970), which is very unlikely for most integer programming problems. Another important result is that, as surrogate constraints incorporate the corresponding original primal constraints as a special case, any optimal solution to the surrogate problem that is feasible for the primal is automatically optimal for the primal. This result contrasts advantageously with Lagrangean relaxation because no complementary slackness conditions are required to determine optimality.

Subgradient Optimization

The subgradient method is a generalization of the gradient method to nondifferentiable concave/convex functions. Subgradient optimization is a well established technique to solve the Lagrangean dual and likewise has been recently proved effective in finding good weights for surrogate constraint relaxations (see for example, Lorena and Lopes 1994).

Let λ^* denote the optimal solution for the Lagrangean dual problem D^λ . Beginning at some point λ^0 (e.g. $\lambda^0 = 0$) the method iteratively generates a sequence of points

$$\lambda^{k+1} = \lambda^k + \theta^k(b - Ax^k),$$

where θ^k is a positive scalar called the step size. If θ^k is small enough, the point λ^{k+1} will be closer (in a Euclidian distance sense) to λ^* than λ^k . In fact, although each step of the subgradient method does not guarantee an increase in $v(LP^\lambda)$, it can be shown (see Held, Wolfe and Crowder 1974)

that under the assumption that $\lim_{k \rightarrow \infty} \theta^k = 0$ and $\sum_{k=0}^{\infty} \theta^k = \infty$, the sequence of $v(LP^{\lambda^k})$ values converges to $v(D^{\lambda^*})$. Therefore, the question is how to select θ^k in order to guarantee convergence. It is common to compute θ^k as

$$\theta^k = \frac{\beta^k [v(D^\lambda) - v(LP^\lambda)]}{\|b - Ax^k\|^2},$$

where β^k is a parameter between 0 and 2 and $v(D^\lambda)$ is an upper bound on the optimal value $v(D^{\lambda^*})$, with x^k being an optimal solution for LP^λ . In practice, by weak duality $v(D^\lambda)$ is replaced by the value of a feasible solution to P and parameter β^k is initialized by setting $\beta^0 = 2$ and halving its current value whenever $v(D^\lambda)$ has not improved for a fixed number of iterations. Likewise, the method stops when no substantial improvement has been verified for a predefined number T of iterations, e.g. $|v(LP^{\lambda^k}) - v(LP^{\lambda^{k+1}})| < \varepsilon$ for a given $\varepsilon > 0$ and $1 \leq t \leq T$.

Cross-Parametric Relaxation

Cross-parametric relaxation combines surrogate and Lagrangean relaxations coupled with Lagrangean-based subgradient search to generate good surrogate constraints. In terms of graph theory, the method can be defined as using a classical subgradient search with a *Lagrangean substitution* as a way to produce *parametric subgradients*, as defined in Glover (1975).

The parametric subgradient feature of the method can be sketched as follows. The subgradient method is used to generate the vector of surrogate multipliers for the relaxed constraints of the primal to create the corresponding surrogate problem. Then, the surrogate vector is used as a *parameter* vector in the subgradient search carried out on the Lagrangean relaxation of the surrogate problem aimed at determining a surrogate dual solution. It should be noticed here that if only the surrogate constraint is relaxed the Lagrangean multiplier is a scale factor rather than a vector. However, there are cases where relaxing more than one component under this Lagrangean substitution framework may be advantageous as will be discussed later. In any event, the new surrogate dual solution and the primal solution (obtained by projecting the surrogate dual solution on the primal-feasible region) yield the new lower and upper bounds, respectively. These

bounds in turn are used in computing the next subgradient-based multipliers to generate a new surrogate problem, thus completing the loop for one iteration of a parametric subgradient search.

In fact, the method is parametric in multiple senses: first in the sense that the subgradient search itself is a search for good parameter (weight) values; second in the sense that the method is substituting the classical Lagrangean based subgradient search inside the surrogate constraint framework to provide what are called parametric subgradients. Although the resulting surrogate relaxation is not guaranteed to produce a smallest duality gap, this method is usefully designed to yield an effective and efficient dual approach for solving difficult integer programming problems.

In other words, the cross-linking of Lagrangean and surrogate relaxations by using (1) Lagrangean-based subgradient directions for the solution of the surrogate dual and (2) subgradient-based surrogate weights for the surrogate relaxation, gives rise to what we call *cross-parametric relaxation*. The method can be viewed as a generalization of the Lagrangean/surrogate relaxation considered in Narciso and Lorena (1999) and the parametric subgradient method as just described.

More formally, the cross-parametric relaxation can be written as

$$v(L_\lambda SP^w) = \text{Min} \{cx - \lambda w(Ax - b) \mid Dx \leq e, x \in \{0,1\}\},$$

where w is a vector of surrogate multipliers and λ is a scale factor representing the Lagrangean multiplier associated with the surrogate constraint.

The corresponding cross-parametric dual problem is the optimization problem in λ and w

$$v(D^{\lambda w}) = \text{Max} \{v(L_\lambda SP^w) \mid \lambda, w \geq 0\}.$$

It is immediate that by setting $\varphi = \lambda w$, problem $D^{\lambda w}$ is the Lagrangean dual D^φ , thus identifying the same optimal dual solution.

The purpose of the cross-parametric relaxation is to approach a solution for D^φ through a *decomposition method* consisting of solving a sequence of locally optimal dual problems

$$v(D_\lambda^w) = \text{Max} \{v(L_\lambda SP^w) \mid \lambda \geq 0\},$$

in which w is a *parameter* associated with a vector of surrogate multipliers determined by the subgradient method (or some pre-defined vector used to initialize the method). In other words, a solution to D_λ^w for a fixed w corresponds to a surrogate dual solution. The method progresses by maintaining an appropriate interaction between the multipliers λ and w in such a way that changes in λ leading to surrogate dual solutions implicitly induce changes in w . This interaction materializes by cross-linking the Lagrangean and surrogate approaches, using Lagrangean based subgradient search within a surrogate constraint relaxation and using subgradient directions (inferred by surrogate dual values and the corresponding projected primal solution values) to determine new surrogate multipliers.

It should be stressed that even though the cross-parametric and traditional Lagrangean relaxations are theoretically equivalent, the optimization processes underlying the two methods are significantly different, yielding different patterns and rates of convergence. This derives from the fact that cross-parametric relaxation is based on optimizing a composite dual variable $\varphi = \lambda w$ that reflects the interaction between two interrelated dual variables. This process rests on the solution of surrogate dual problems that do not arise in traditional Lagrangean relaxation. Also, since no assumption is made about the value of w when choosing λ , it is very unlikely that a Lagrangean substitution search over φ for a fixed w will lead to the same dual solutions produced by subgradient directions over λ without the parameter w . Therefore, one can expect that only for optimal combinations of w and λ , problems D_λ^w and D^φ produce the same bounds. In consequence, these composite relaxation strategies embedded in the cross-parametric relaxation approach can find potentially better local bounds than traditional Lagrangean relaxation alone.

A general framework for the cross-parametric relaxation approach is depicted in the flowchart of Figure 1. In the diagram $X(\cdot)$ denotes the set of feasible solutions for a problem “.”. Also, x_λ and x_w represent solutions to the problems $L_\lambda SP^w$ and SP^w , respectively. Consequently, it is assumed that if x_λ is feasible for SP^w , both x_λ and x_w stand for the same solution.

As the figure shows, the information contained in each surrogate solution may be used in a constructive process for building a feasible solution to the primal problem. In addition, an improvement method is used in an attempt to find an enhanced solution, which thereby may be used to replace the current upper bound for the next subgradient iteration. By this means, a sequence of Lagrangean multipliers generates lower bounds, while a sequence of feasible

and enhanced solutions defines upper bounds. Without loss of generality, we will refer to the block “constructive plus improvement” simply as the *projection method*, because of its primary function of projecting a dual solution into primal feasible space. Some advances on the design of effective projection methods will be given in the next section.

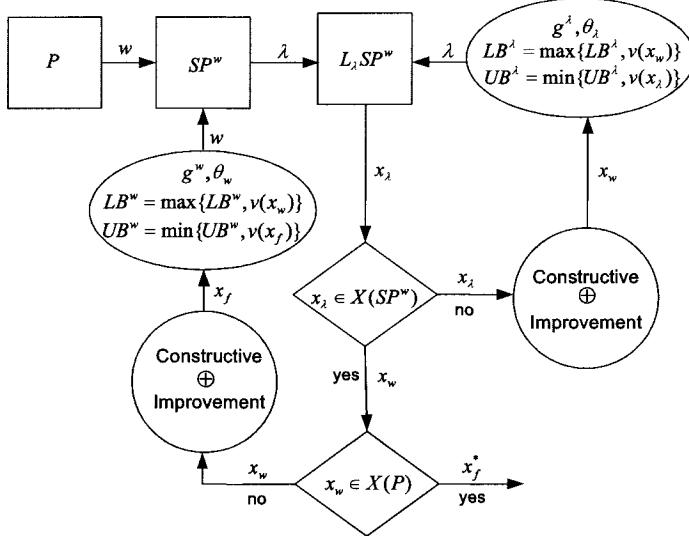


Figure 1. Cross-Parametric Relaxation Procedure

To conclude this section it is useful to say a few words about the rationale of the cross-parametric relaxation. Besides the conceptual relaxation design discussed earlier concerning some potentially misleading similarities with traditional Lagrangean subgradient search, cross-parametric relaxation rests on theoretical foundations stemming from properties of surrogate constraints. Consequently, they are exploited using concepts tied to surrogate duality theory, in particular those associated with the notion of parametric subgradients (Glover 1975). More precisely, cross-parametric relaxation is founded on the following observations.

Theoretically, a solution for a surrogate problem can be difficult to obtain. In fact, if the primal problem P is NP-complete, any surrogate problem SP^w derived from it is also NP-Complete. Yet in practice a problem relaxation having fewer constraints than the original problem is usually easier to solve. Therefore one must be concerned with finding an appropriate surrogate relaxation for which effective methods exist to solve the

corresponding surrogate problem. In an extreme situation, if all constraints are relaxed to become replaced by the surrogate constraint, the surrogate problem becomes a knapsack problem. Although the knapsack problem is NP-Complete, in many cases it can be solved efficiently by specialized algorithms, particularly by drawing on the work of Pisinger (1999a, 1999b, 2004), and Martello, Pisinger and Toth (2000). Nevertheless, from the theory of NP-Completeness one might find instances (not necessarily very large) for which finding an optimal solution may be computationally very expensive. Cross-parametric relaxation becomes especially relevant in such a case. When the surrogate problem is difficult to solve, the step of relaxing the surrogate constraint in Lagrangean fashion, especially using parametric subgradients, may produce an effective procedure to find optimal or near-optimal surrogate dual solutions. This assumption is in part supported by the interesting results obtained by Lorena and Lopes (1994), and Narciso and Lorena (1999) in their applications to the set covering and generalized assignment problems, employing a special case of the cross-parametric relaxation proposed here and relatively simple greedy heuristic projection methods.

The motivation for the use of parametric subgradients is also reinforced by the fact that in many practical situations, problems are so complex that the corresponding formulations may involve a relatively large set of difficult constraints of the type exemplified by $Ax \leq b$ in our general formulation. In these circumstances, a surrogate constraint model can be generated by grouping sets of constraints of the same type, such as those modeling similar requirements as a way to create multiple surrogate constraints. Under this framework a cross-parametric relaxation may involve the Lagrangean relaxation of all the surrogate constraints, and in this case λ is no longer a scale factor but a vector of multipliers; hence subgradient optimization becomes particularly relevant to perform a multiple-dimensional search of improved dual solutions. The potential value of generating multiple surrogate constraints has been suggested in Glover (1965, 1968) and cross-parametric relaxation affords a means to exploit such a possibility and develop a more powerful relaxation-based algorithm.

3. The RAMP Method

The RAMP method is presented in two stages, representing different levels of sophistication. The first level focuses on exploiting a dual framework, which is confined to creating an adaptive memory relaxation based on the type of relaxations described in Section 2, making use of tabu search and path-relinking strategies. The second level of sophistication

embodies the primal-dual framework, which establishes the relationship between the dual approach (defined in the first level of sophistication) and a primal approach represented by scatter search and path-relinking. The role of path-relinking at this second level is different from the one represented in the first level, as will be clear later in this section.

3.1 Relaxation Adaptive Memory Programming

We begin by noting some similarities and contrasts between a subgradient search and a path-relinking approach, which gives the foundation for a path-relinking projection method as a fundamental component for a more advanced Relaxation AMP approach.

Subgradients of the Lagrangean dual function $v(D^\lambda)$ at each investigated multiplier λ^{k+1} define convex linear combinations of vectors $(Ax^k - b)$ for solutions x^k solving LP^λ . From this perspective, subgradient optimization with Lagrangean substitution is a search method that links partially feasible primal solutions (that only satisfy $Dx \leq e$) to the subset of solutions that are primal feasible (i.e. that also satisfy $Ax \leq b$). Thus, from a more abstract perspective we may consider that the inequality system $Ax \leq b$ implicitly defines a reference set of solutions that the method is moving towards. This contrasts with a path-relinking approach where the reference set is explicitly defined by solutions that have already been visited. Also, we may say that solutions are linked to one another by means of subgradient directions.

In a path-relinking approach, moves from one solution to the next are derived from weighted transformations that drive the search toward a subset of solutions that meet certain requirements (defined by the level of importance, with regard to optimality, of each of their own attributes). By analogy, subgradient directions are derived from weighted combinations generated with the goal of converging to the best solutions that possess the requirements (or attributes) specified by $Ax \leq b$. In this sense, the method can be viewed as a special case of a path-relinking approach where a move from one solution to another in the path is given by a subgradient direction. During a path-relinking trajectory each move typically avoids reconsidering attributes that have been previously deleted (or rejected) throughout the path—this is implemented by defining appropriate *tabu restrictions* associated with the type of neighborhood utilized. Similarly, the subgradient approach creates dual solutions (representing Lagrangean multipliers) that are meant to penalize violated primal constraints, thus keeping the method from

generating the same sets of solution attributes that would reiterate over repeated $(Ax^k - b)$ vectors—this is accomplished by choosing adequate *step sizes* to ensure appropriate convergence.

The foregoing observations underlie the foundation of a path-relinking projection method, as amplified in the following discussion.

Simple Relaxation AMP.

Subgradient directions pointing to solutions that satisfy $Dx \leq e$ are attractive to provide starting points for a method that projects these solutions on the domain of feasibility of the original primal problem P . We emphasize that the type of projection methods that are referred to in the context of this paper are heuristic rather than exact and in particular are structured to exploit the notion of adaptive memory programming. As a prelude to other more elaborate strategies, we start by proposing the use of frequency-based tabu search memory to construct a primal solution from the dual solution and also to improve this solution further after reaching primal feasibility. Lower and upper bounds are then updated based on the new values of the dual and primal solutions respectively, and these values are used to compute a new subgradient direction from the dual, if it applies.

Even though this method constitutes the simplest form of a Relaxation AMP proposed here, we have found it can be quite effective compared to currently popular state-of-the-art constructive or local search projection methods based on simple greedy approaches. Also, the Simple Relaxation AMP can be enhanced by the use of neighborhood structures incorporating adaptive and dynamic search such as filter-and-fan or ejection chain methods. (For a detailed description of these methods see Rego and Glover (2002). Recent applications of filter-and-fan and ejection chain methods are reported in Greistorfer, Rego and Alidaee (2003), Renato, Rego and Gamboa (2004), and Rego, Li and Glover (2004).)

Advanced Relaxation AMP.

We now describe a more advanced approach that establishes a stronger connection between the primal and dual. To do this, we retain a collection of *elite* primal solutions selected among those projected from the dual to define a reference set for a *path-relinking projection method*, generating paths between dual solutions and their primal counterpart elite solutions. Here, the

term “elite” is employed as in tabu search to refer to the output of a reference set update method utilizing appropriate measures of solution quality and diversity. An abbreviated version of path relinking projection consists of keeping a single solution in the reference set represented by the best upper bound found so far.

The method is structured in two phases. A first phase creates an initial reference set based upon a tabu search projection method, with the difference that, for the purpose of speeding up this initialization phase, only a rudimentary long term memory is employed, or even none at all if short-term memory is sufficient to achieve a reference set with desired levels of diversity and solution quality. The second phase replaces the constructive phase of the tabu search projection method by a path-relinking projection method to transform dual solutions (generated by the subgradient search) into primal feasible solutions. The analogy between the subgradient search and the path-relinking strategy constitutes a primary motivation for the development of this path-relinking projection method. The method operates by starting from a trial solution obtained from the dual and moving toward pre-defined subsets of guiding solutions in the current reference set. The definition of the aforementioned subset is established by an appropriate subset generation method as considered in the classical path-relinking/scatter search template (Glover 1997). To take fuller advantage of adaptive memory programming, tabu search is used as an improvement method over the best primal solutions obtained after combinations. A useful variation of such an improvement method can be based on integrating path-relinking and ejection chain strategies, as initially proposed in Rego and Glover (2002) and recently demonstrated to be effective for the solution of generalized assignment problems by Yagiura, Ibaraki, and Glover (2004a).

We should note that in these Relaxation AMP methods, the primal algorithm is confined to an improvement method executed by a tabu search procedure and a selection process empowered by a reference set update method. Therefore, it is considered a dual-based metaheuristic approach. A more sophisticated method that takes full advantage of the primal-dual relationships and adaptive memory programming is the Primal-Dual RAMP approach presented next.

3.2 Primal-Dual Relaxation Adaptive Memory Programming

Primal-Dual Relaxation Adaptive Memory Programming (PD-RAMP) goes beyond the customary notions of primal-dual relationships used in linear programming (where the primal and dual have the same optimum values). The method is enhanced by adaptive memory strategies that affect both sides of the primal-dual connection, as a means for bridging the duality gap that exists in combinatorial optimization.

More precisely, PD-RAMP refers to the exploitation of primal-dual relationships, by an adaptive process that takes advantage of the primal side using scatter search and path relinking, and takes advantage of the dual side using surrogate constraint and Lagrangian relaxations. It extends the basic (first level) form of the RAMP method by interconnecting its dual component with an appropriate primal component that utilizes adaptive memory programming more thoroughly. This is accomplished by integrating memory and learning through the use of a reference set of solutions that is common to and updated by both the primal and the dual approaches.

The main function of the dual approach is to generate new solutions for the reference set as specified by the first level RAMP approach. For a better understanding of the conceptual design of the PD-RAMP approach, we underline a few important concepts concerning the foundations of the surrogate constraint relaxations and contrast them with those of scatter search. (Similar concepts hold for surrogate constraints and the cross-parametric relaxation, therefore for the sake of simplifying the explanation we restrict attention to surrogate constraints.)

As previously noted, surrogate constraint approaches explore the solution space by generating solutions that are derived from the creation of nonnegative linear combinations of constraints. The process seeks to capture relevant information contained in individual constraints and integrate it into new surrogate constraints. The result is to generate composite decision rules (characterized by each surrogate problem) leading to associated new trial solutions. By contrast, scatter search combines vectors of solutions rather than combining vectors of constraints, and likewise is organized to capture information not contained separately in the original vectors. These observations set the stage for the primal-dual metaheuristic approach identified by the PD-RAMP method. Although there are other possible and also interesting variations to create primal-dual strategies under the RAMP

model, we start off considering scatter search and surrogate constraints as the basic methods for the primal and dual approaches, respectively.

The method starts by creating an initial reference set of solutions produced by a surrogate approach as laid down by the initial phase of the RAMP approach. At this stage the reference set contains relevant information that was made available by the dual approach. Although the surrogate constraint model is provided with the ability to integrate information extracted from individual constraints, each piece of information, generated at each iteration of the surrogate constraint method, is represented in a single solution that results from solving the associated surrogate problem (and the application of the complementary projection method). These solutions may be viewed as individual memory structures that can be integrated to create more complex compound memory structures. This result is accomplished by the scatter search method through generating weighted combinations of these solutions (and so their attributes) to create new composite solutions. The method is structured to alternate between the primal and dual approaches, both updating the reference set in an evolutionary fashion.

Figure 2 provides an illustration of the general RAMP model that is completed by the Primal-Dual RAMP approach.

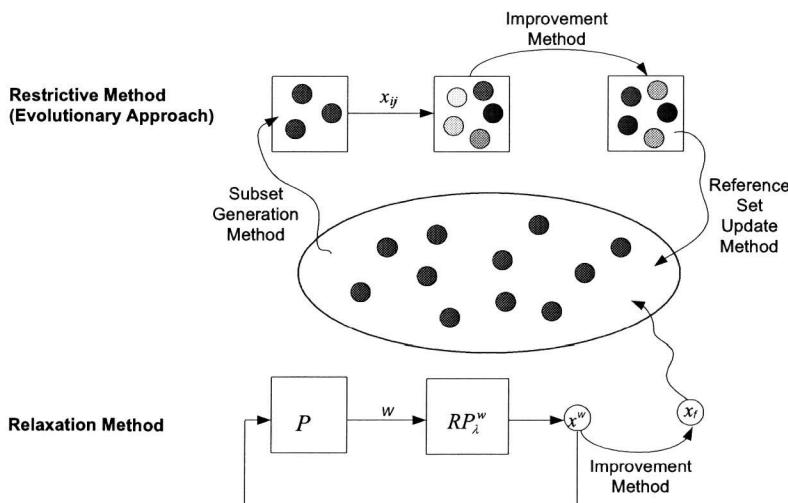


Figure 2. Relational AMP model

For the sake of providing a more comprehensive illustration, the complete PD-RAMP method uses cross-parametric relaxation in the primal side and scatter search in the dual side, though other variations of the method are possible. For example, scatter search can be naturally replaced by its path-relinking generalization. Ultimately a hybrid approach can be created by using other evolutionary approaches such as genetic algorithms (GAs) or evolution strategies (ESs), preferentially provided with adaptive memory components. Likewise, traditional surrogate constraint and Lagrangean relaxation can be used as stand alone dual procedures rather than combined in a cross-parametric approach. As far as the improvement method is concerned, there are advantages for using the adaptive memory orientation of tabu search. In addition, as described earlier path-relinking is also particularly relevant to create a projection method to bridge the gap between dual and primal solutions.

4. Conclusions and Perspectives

The complete RAMP method (embodying its primal-dual form as well as its more basic form) is organized to take advantage of a number of existing methods, using them as components or building blocks to form more advanced search strategies. The resulting procedure affords advantages that cannot be obtained by more customary metaheuristic approaches alone, or even by current hybrid methods. The component methods within RAMP are strategically articulated in a unified design rather than used as independent add-on components to be called whenever other components are unable to make progress in finding improved solutions. The cross-parametric relaxation method, the path-relinking projection method (in connection with subgradient optimization) and the integration of surrogate constraints and scatter search constitute primary factors of such an articulated methodology.

A key contribution of the RAMP method is its capability to exploit duality as well as primal-dual relationships that have been largely neglected in the field of metaheuristics. Moreover, the method includes a learning process that relies on adaptive memory rather than kicking-off or re-starting the search through randomized-based processes. Although memory is explicitly structured by reference to solution attributes, the RAMP method implicitly includes an automatic learning process embodied in an evolutionary framework that is enhanced by a cohesive integration of primal and dual approaches.

Finally, the variety of possibilities to create intriguing variations of RAMP algorithms opens new doors for research in metaheuristics. Opportunities for advances emerge from discoveries about primal-dual coordination strategies, including the coordination of parallel primal dual search. Advances can also be derived from combinations of scatter search and path-relinking strategies (or other evolutionary approaches) on the primal side and Lagrangean, surrogate or cross-parametric relaxations on the dual side. New neighborhood structures may likewise be exploited in RAMP strategies. In this connection, recent advances in ejection chain methods and combinations of these with path-relinking and filter-and-fan approaches (Rego and Glover, 2002) reveals an interesting path of research for creating advanced variants of RAMP.

Preliminary computational results obtained by the RAMP and PD-RAMP methods are very encouraging. Applied to two challenging and well-studied problems, the Generalized Assignment Problem (GAP) and the Multi-Resource Generalized Assignment Problem (MRGAP), the RAMP method yields results rivaling the best in the literature, while the PD-RAMP method dominates the performance of the previously best methods for these problems. In particular, the PD-RAMP finds solutions that are always as good or better than the leading approaches of Yagiura, Ibaraki and Glover (2004a, 2004b) for the GAP problem and of Yagiura et al. (2004) for the MRGAP problem, while using 10% on average of the time required by these algorithms. (The referenced methods achieve their leading position by also using path-relinking and ejection chains, but without embedding them within the primal-dual RAMP framework proposed here.) Compared to CPLEX 8.1, the PD-RAMP approach finds optimal solutions to those problems that are simple enough to enable CPLEX to solve them to optimality within a reasonable amount of time, but generally requires only 5% of the solution time required by CPLEX to obtain these solutions. For larger and more difficult problems, PD-RAMP obtains solutions whose quality is superior to that of the best solutions CPLEX is able to find when it is allowed to run up to 70 times longer than PD-RAMP. In particular, PD-RAMP finds the optimal solution for about 85% of these larger problems tested while CPLEX succeeds in finding optimal solutions for less than 19% of these problems under its significantly larger allotted time span. More detailed results can be found in Rego et al. (2004), currently in process.

References

- Duarte, R., C. Rego and D. Gamboa (2004) "A Filter and Fan Approach for the Job Shop Scheduling Problem: A Preliminary Study," in Proceedings: International Conference on Knowledge Engineering and Decision Support, 401-406.
- Glover, F. (1965) "A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problem," *Operations Research*, 13:879-919.
- Glover, F. (1968) "Surrogate Constraints," *Operations Research*, 16(4):741-749.
- Glover, F. (1975) "Surrogate Constraint Duality in Mathematical Programming," *Operations Research*, 23(3):434-451.
- Glover, F. (1997) "A Template for Scatter Search and Path Relinking," in Lecture Notes in Computer Science, 1363, J.K. Hao, E. Lutton, E. Ronald, M. Schoenauer, D. Snyers (Eds.), 13-54.
- Greenberg, H.J. and W.P. Pierskalla (1970) "Surrogate Mathematical Programming," *Operations Research*, 18:1138-1162.
- Greistorfer, P., C. Rego and B. Alidaee "Simple Filter-and-Fan Approach for the Facility Location Problem," Research Report, Hearin Center for Enterprise Science, School of Business Administration, University of Mississippi, USA.
- Held, M., P. Wolfe and H.P. Crowder (1975) "Validation of Subgradient Optimization," *Mathematical Programming*, 6:62-88.
- Lorena, L.A.N. and M.G. Narciso (1999) "Lagrangean/Surrogate Relaxations for the Generalized Assignment Problems," *European Journal of Operational Research*, 114(1): 165-177.
- Lorena, L.A.N. and F.B. Lopes (1994) "A Surrogate Heuristic for Set Covering Problems," *European Journal of Operational Research*, 79:138-150.
- Martello, S., D. Pisinger and P. Toth (2000), "New Trends in Exact Algorithms for the 0-1 Knapsack Problem," *European Journal of Operational Research*, 123:325-332.
- Pisinger, D. (1999a) "Core problems in Knapsack Algorithms," *Operations Research*, 47:570-575.
- Pisinger, D. (1999b) "An Exact Algorithm for Large Multiple Knapsack Problems", *European Journal of Operational Research*, 114:528-541.
- Pisinger, D. (2004) "Where are the hard knapsack problems?," *Computers and Operations Research*, to appear.
- Poljak, B.T. (1969) "Minimization of Unsmooth Functionals," USSR Computational Mathematics and Mathematical Physics, 9:14-29.
- Rego, C. and F. Glover (2002) "Local Search and Metaheuristics for the Traveling Salesman Problem," in book The Traveling Salesman Problem and its Variations, G. Gutin and A. Punnen Editors, Kluwer Academic Publishers, 309-368.
- Rego, C., H. Li, and F. Glover (2004) "Adaptive and Dynamic Search Neighborhoods for Protein Folding in a 2D-HP Lattice Model", Research Report, Hearin Center for Enterprise Science, School of Business Administration, University of Mississippi, USA.

- Rego, C., L. Sagbansua, B. Alidaee, and F. Glover (2004) "RAMP and Primal-Dual RAMP Algorithms for the Multi-Resource Generalized Assignment Problem," Research Report, Hearn Center for Enterprise Science, School of Business Administration, University of Mississippi, USA.
- Yagiura, M., T. Ibaraki, and F. Glover (2004a) "A Path Relinking Approach with Ejection Chains for the Generalized Assignment Problem," *European Journal of Operational Research*, to appear.
- Yagiura, M., T. Ibaraki and F. Glover (2004b) "An Ejection Chain Approach for the Generalized Assignment Problem," *INFORMS Journal on Computing*, 16(2):133-151.
- Yagiura, M., S. Iwasaki, T. Ibaraki and F. Glover (2004) "A Very Large-Scale Neighborhood Search Algorithm for the Multi-Resource Generalized Assignment Problem," *Discrete Optimization*, 1:87-98.

Index

A

- Adaptive Memory, 1, 12, 14, 22, 149, 151, 155, 213, 214, 217, 218, 220, 223, 227, 232, 288, 289, 290, 293, 391, 413, 425, 426, 427, 428, 431, 436, 437, 441, 442, 443, 451, 453, 454, 455, 457
Algebraic Modeling Language, 37, 51
Aspiration Criterion, 149, 155, 198, 199, 201, 248, 285, 347, 430, 435
Aspiration Plus, 361

B

- Branch and Bound, 31, 43, 122, 166, 248, 257, 258, 274, 364, 374
Branch and Cut, 62, 257
Branching, 256, 258, 360, 361, 363, 365, 430, 431

C

- Cancellation Sequence Method, 400
Candidate Lists, 209, 251, 360, 361, 362, 435
Chunking, 373, 374, 404, 410, 412
Clustering, 14, 407, 418, 427
Combinatorial Implosion, 427
Combinatorial Problems
 Aerial Fleet Refueling, 304, 317
 Bandwidth Reduction, 265, 273
 Bin Packing, 367
 Course Scheduling, 382, 383
 Covering Tour, 59, 74, 88
 Cutting Planes, 59, 71, 72, 73, 77, 81, 84, 86, 89, 425

- Generalized Traveling Salesman, 62
Graph Partitioning, 93, 115, 293
Linear Ordering, 229, 236, 242, 245, 265, 270, 273, 274, 416
Multicommodity, 290, 294
Set Covering, 6
Single Machine-Sequencing, 273, 274
Steiner Problem, 411
Traveling Salesman, 2, 6, 59, 61, 146, 273, 274, 303, 304
Vehicle Dispatching, 290, 351
Vehicle Routing, 1, 2, 3, 4, 6, 21, 22, 65, 145, 319, 350, 367, 397, 409
Complementary Slackness, 445, 446
Conditionally Attractive, 428, 429, 436, 437
Consistent Variables, 426, 427
Constraint Programming, 357, 427
Convex Combination, 14, 77, 230
Convex-Hull, 14, 15, 16
Co-operation, 283, 288, 289, 290, 291, 293, 294, 296
Co-operative Search, 293
Critical Event, 13, 19

D

- Decision Support, 329, 330, 339, 373, 374, 376
Decomposition Method, 448
Directional Rounding, 73, 78, 79
Distance, 4, 13, 14, 27, 28, 32, 33, 34, 35, 37, 38, 41, 46, 50, 59, 60, 61, 69, 80, 106, 145, 146, 172, 177, 178, 179, 180, 183, 184, 240, 252, 256, 266, 274, 304, 307, 308, 343, 351, 376, 377, 378, 380,

- 381, 382, 383, 387, 394, 395, 397, 401, 402, 403, 404, 405, 406, 410, 411, 412, 413, 414, 415, 419, 446
- Diversification, 3, 4, 8, 9, 10, 11, 12, 14, 19, 22, 68, 69, 71, 72, 74, 77, 78, 80, 95, 105, 106, 107, 108, 109, 111, 113, 115, 147, 151, 153, 155, 165, 166, 167, 170, 176, 183, 200, 213, 214, 219, 220, 222, 223, 229, 233, 234, 235, 236, 237, 239, 240, 241, 247, 248, 252, 254, 255, 256, 265, 267, 269, 281, 285, 294, 312, 322, 348, 374, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 409, 410, 411, 412, 414, 415, 416, 417, 418, 419, 425, 426, 427, 429, 430, 431, 433, 434, 435, 436
- Diversification Generator, 236, 237, 240, 269, 348, 412, 417
- Dual, 1, 194, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458
- Duality Gap, 264, 442, 443, 444, 448, 455
- ## E
- Ejection Chains, 147, 148, 152, 153, 154, 155, 289, 368, 453, 454, 458
- Elite Solutions, 107, 184, 313, 366, 387, 388, 389, 391, 392, 394, 396, 407, 411, 412, 413, 414, 417, 418, 419, 453
- Evolution Strategies, 457
- Evolutionary Algorithms, 1, 351, 391
- Evolutionary Approach, 22, 29, 441, 457, 458
- Evolutionary Methods, 1, 29, 229, 264, 351
- ## F
- Facility Location, 191, 192, 193, 194, 195, 196, 201, 209, 333, 334, 394
- Filter-and-Fan, 453, 458
- Flow Shop, 177, 180
- Frequency Memory, 235, 236, 237, 427, 429
- ## G
- Generalized Assignment, 6, 8, 407, 451, 454, 458
- Generalized Reduced Gradient, 26, 30
- Genetic Algorithms, 12, 29, 146, 149, 244, 263, 264, 267, 270, 283, 294, 295, 351, 388, 391, 405, 407, 441, 442, 457
- Gradient Methods, 25, 446
- Graph Theory, 1, 4, 96, 447
- GRASP, 214, 217, 218, 236, 360
- Greedy, 93, 94, 96, 110, 111, 112, 113, 114, 115, 193, 194, 200, 201, 204, 218, 219, 236, 250, 365, 366, 367, 390, 409, 413, 414, 416, 417, 442, 451, 453
- Green Logistics, 333, 341, 351
- Group Theoretic Tabu Search, 303, 319, 320
- Group Theory, 303, 304, 305, 306, 307, 308, 309, 311, 314, 316, 323, 324
- ## H
- Hamming Distance, 106, 377, 380, 406
- Hash Functions, 18, 70, 182, 407, 415
- Heuristics, 2, 3, 9, 22, 25, 29, 31, 59, 61, 62, 67, 75, 76, 77, 81, 82, 86, 96, 97, 137, 146, 151, 171, 191, 192, 193, 194, 195, 196, 199, 200, 202, 203, 204, 205, 206, 208, 209, 214, 217, 218, 219, 220, 224, 229, 256, 264, 284, 288, 296, 304, 311, 330, 339, 342, 345, 350, 359, 361, 364, 366, 367, 389, 390, 391, 393, 395, 405, 411, 415, 428, 429, 430, 431, 435, 436, 437, 438, 439, 442, 451, 453
- ## I
- Information Technology, 193, 339

Integer Programming, 63, 65, 68, 70, 229, 247, 249, 258, 264, 304, 347, 360, 425, 444, 446, 448

Integrality Gap, 444

Intensification, 12, 14, 19, 22, 95, 105, 106, 108, 113, 145, 147, 149, 150, 151, 154, 229, 232, 233, 239, 244, 247, 248, 249, 252, 253, 254, 255, 256, 281, 285, 322, 387, 388, 389, 391, 392, 393, 394, 395, 396, 407, 408, 409, 411, 412, 416, 418, 419, 425, 426, 427

Inventory Management, 333, 337

Iterated Local Search, 329, 331, 345, 346, 352

J

Job Shop, 117, 165, 177, 180, 289

K

Knapsack Problem, 6, 388, 451

L

Lagrangean Dual, 445, 446, 448, 452

Lagrangean Multiplier, 264, 446, 447, 448, 449, 452

Lagrangean Substitution, 447, 449, 452

Large Neighborhood Search, 427

Large Scale Neighborhood Search, 427

Layered Network, 74

Linear Combination, 3, 15, 16, 17, 18, 20, 29, 72, 76, 77, 216, 235, 264, 270, 391, 411, 445, 452, 455

Linear Programming, 230, 248, 358, 425, 436, 444, 455

Local Search, 9, 11, 22, 27, 67, 68, 69, 70, 71, 72, 75, 76, 78, 80, 109, 117, 122, 134, 147, 168, 181, 183, 194, 213, 214, 216, 217, 218, 219, 220, 227, 235, 237, 239, 263, 266, 267, 268, 269, 275, 276, 277, 278, 281, 284, 289, 292, 293, 304,

342, 346, 349, 357, 359, 365, 390, 391, 392, 407, 409, 411, 417, 442, 443, 453

Logical Restructuring, 360, 367, 368

Logistics, 2, 226, 303, 320, 324, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 344, 349, 350, 351, 352

Logistics Management, 329, 330, 332, 342, 344, 349, 352

Long Term Memory, 145, 147, 149, 150, 151, 192, 198, 199, 200, 201, 203, 204, 208, 285, 413, 414, 454

M

Mahalanobis Distance, 378, 379, 380, 381, 384, 403, 404

Material Requirement Planning, 338

Maxmin Criterion, 13, 240, 241

Memetic Algorithms, 407

Metaheuristics, 1, 29, 62, 88, 192, 194, 257, 283, 284, 287, 288, 290, 291, 292, 294, 295, 296, 297, 303, 304, 305, 306, 307, 308, 314, 315, 316, 317, 323, 324, 329, 330, 331, 339, 342, 343, 346, 347, 348, 349, 357, 359, 382, 384, 387, 388, 390, 401, 406, 409, 425, 426, 428, 441, 442, 443, 454, 455, 457

Mixed Integer Nonlinear Programming, 25

Mixed Integer Programming, 247, 304, 404, 412, 425, 426, 428, 430, 433

m-TSP, 304, 307, 308, 311, 315

Multi-Resource Generalized Assignment, 458

Multistart, 25, 26, 27, 28, 31, 93, 289, 291, 425, 436, 437

Multistart Heuristic, 25, 26, 289

Multi-Start Methods, 27, 425, 436

N

Neighborhoods, 11, 97, 98, 99, 102, 103, 104, 109, 111, 166, 168, 169, 170, 175, 177, 183, 184, 217, 218, 222, 223, 248, 250, 251, 269, 270, 277, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 342, 348, 349, 357, 360, 361, 391, 394, 397, 398, 399, 406, 408, 409, 411, 414, 416, 418, 426, 437, 442, 443, 452, 453, 458

Network Design, 59, 290, 294, 333, 334, 335, 343

Neural Networks, 146, 166, 194, 229

Non-Convex Combination, 231

NP-Complete, 2, 215, 450, 451

NP-Hard, 59, 60, 117, 165, 194, 274, 406

O

Operations Research, 231, 303, 349, 364

Optimization, 1, 2, 25, 29, 51, 105, 151, 165, 166, 180, 191, 201, 229, 232, 237, 264, 270, 323, 330, 333, 334, 342, 343, 344, 345, 349, 352, 365, 370, 373, 374, 375, 376, 380, 384, 393, 394, 407, 408, 409, 412, 425, 426, 428, 429, 430, 432, 433, 435, 436, 441, 442, 443, 448, 449

Combinatorial, 1, 4, 6, 117, 121, 146, 147, 165, 194, 213, 229, 263, 265, 273, 303, 319, 346, 349, 357, 358, 406, 441, 442, 443, 455

Discrete, 1, 153, 233, 358

Global, 25, 31, 38, 47, 51, 357

Multi-Criteria, 374

Nonlinear, 229, 238, 243, 270

Subgradient, 194, 441, 445, 446, 451, 452, 457

Optimization Solvers

CPLEX, 44, 80, 93, 95, 97, 111, 113, 458

Frontline, 28, 31, 51

GAMS, 25, 37, 38, 39, 43, 46, 49, 51, 52, 53, 54, 55, 56

GLPK, 248, 249, 256, 257, 258, 259

MINOS, 30

OptQuest, 25, 26, 29, 31, 32, 33, 34, 35, 37, 38, 40, 42, 43, 44, 46, 48, 50, 51, 345, 350

Xpress-MP, 257, 258

OptTeck Systems, 345

P

Parallel Computation, 283, 392

Parallel Metaheuristics, 284, 285, 290, 294, 296, 297

Parallelization Strategies, 283, 287, 290, 296

Parametric Subgradients, 443, 447, 448, 450, 451

Path Relinking, 95, 105, 106, 115, 165, 166, 177, 184, 283, 284, 290, 295, 296, 312, 315, 365, 368, 387, 388, 390, 391, 411, 413, 441, 454, 455

Path-Relinking Projection Method, 452, 453, 454, 457

PD-RAMP, 441, 443, 444, 455, 457, 458

Penalty Values, 19

Permutation Problems, 1, 6, 8, 232, 233, 244, 263, 265, 266, 269, 270, 275, 408

Persistently Attractive, 428, 430, 432

Pool, 146, 152, 290, 294, 295, 374, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 399, 401, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419

Population, 2, 11, 29, 32, 33, 34, 35, 37, 44, 46, 50, 149, 150, 263, 264, 276, 294, 295, 347, 348, 373, 374, 375, 378, 380, 384, 387, 388, 391, 392, 401, 402, 403, 407, 408, 413, 414, 416

Primal, 1, 44, 89, 172, 185, 441, 442, 443, 444, 445, 446, 447, 449, 450, 452, 453, 454, 455, 456, 457, 458

Primal-Dual RAMP, 454, 456, 458

Primal-Dual Solution, 445

Principal Components, 374, 380, 384

Principle of Congenial Structures, 367

Product Design, 333, 337

Projection, 50, 79, 180, 256, 380, 425, 426, 427, 428, 431, 436, 437, 450, 451, 453, 454, 456, 457

Projection Method, 425, 426, 427, 428, 436, 450, 451, 453, 454, 456, 457

Pseudo-Cuts, 430, 431, 432

Q

Quadratic Assignment, 149, 287, 406

Quadratic Programming, 30

R

RAMP, 441, 443, 444, 451, 455, 456, 457, 458

Reactive Tabu Search, 325, 396, 411

Recency, 198, 226, 227, 232, 396, 400, 413, 429

Recovering Beam Search, 360, 370

Reference Set, 3, 4, 11, 12, 13, 14, 15, 16, 19, 21, 29, 35, 67, 68, 69, 70, 71, 72, 73, 74, 76, 77, 78, 80, 81, 88, 108, 109, 232, 234, 235, 237, 238, 239, 240, 241, 242, 243, 266, 267, 290, 295, 296, 347, 348, 387, 388, 391, 412, 417, 452, 453, 454, 455, 456

Relaxation

Cross-Parametric, 441, 443, 444, 447, 448, 449, 450, 451, 455, 457, 458

Lagrangean, 2, 264, 441, 443, 444, 446, 447, 449, 451, 457

Linear Programming, 65, 67, 72, 82, 89, 358, 444

Surrogate, 2, 265, 443, 445, 448, 450

Relaxation Adaptive Memory

Programming, 441, 452, 455

Restricted Search Space, 425

Reverse Elimination Method, 396, 400

Reverse Logistics, 341

S

Satisfiability, 213, 214, 215

Scatter Search, 1, 2, 3, 4, 6, 7, 8, 10, 11, 12, 14, 18, 21, 22, 25, 26, 29, 59, 62, 67, 68, 71, 73, 78, 79, 80, 81, 84, 86, 88, 89, 93, 95, 105, 108, 115, 229, 230, 231, 232, 233, 235, 236, 237, 238, 239, 242, 243, 244, 245, 263, 264, 265, 267, 268, 270, 273, 275, 280, 283, 284, 290, 295, 296, 329, 331, 345, 347, 348, 352, 353, 373, 374, 381, 382, 384, 387, 388, 391, 411, 412, 414, 416, 441, 442, 443, 452, 454, 455, 456, 457, 458

Scheduling, 6, 117, 118, 122, 165, 173, 187, 193, 213, 229, 233, 238, 245, 274, 288, 289, 320, 322, 323, 331, 335, 336, 338, 339, 351, 367, 382, 413, 414

Sequential Fan, 287, 364

Sequential Fan Candidate List, 287, 364

Short Term Memory, 145, 147, 149, 198, 199, 200, 206, 239, 248, 254, 346, 396, 400, 413, 414, 454

Shortest Path, 74, 75

Similarity, 325, 387, 397, 401, 403, 405, 413, 418

Simplex Method, 425

Simulated Annealing, 146, 166, 283, 295, 342, 391, 396

SONET, 93, 94, 96, 115

SONET Ring Assignment, 93, 94

Spanning Tree, 368, 369

Star Path, 73, 78, 79, 80, 81, 86, 89

Strategic Oscillation, 12, 19, 95, 103, 104, 105, 115, 426, 443

Strong Lagrangean, 444

Strongly Determined, 426, 427, 428
 Structured Weighted Combinations, 233
 Subgradient Method, 446, 447, 448, 449
 Successive Filter, 363
 Supply Chain, 191, 329, 330, 331, 332,
 333, 334, 335, 336, 337, 338, 339, 340,
 341, 342, 349, 352, 353
 Surrogate Constraints, 1, 213, 214, 216,
 217, 218, 219, 223, 226, 227, 264, 441,
 443, 444, 445, 446, 447, 448, 449, 450,
 451, 455, 456, 457
 Surrogate Dual, 443, 446, 447, 448, 449,
 450, 451
 Surrogate Problem, 443, 445, 446, 447,
 448, 450, 451, 455, 456

T

Tabu List, 99, 100, 101, 108, 118, 121,
 132, 133, 134, 135, 136, 142, 169, 170,
 171, 174, 322, 347, 400, 410
 Tabu Restrictions, 433, 435, 436, 452
 Tabu Search, 1, 12, 19, 93, 94, 95, 97,
 101, 103, 105, 107, 115, 117, 118, 121,
 122, 134, 136, 142, 145, 146, 147, 148,
 153, 154, 155, 165, 166, 170, 171, 191,
 192, 194, 195, 196, 200, 201, 202, 203,
 204, 205, 206, 207, 208, 209, 219, 220,
 232, 233, 239, 244, 247, 248, 249, 250,
 251, 252, 253, 254, 255, 256, 257, 258,
 259, 260, 274, 283, 284, 285, 286, 287,
 288, 289, 290, 291, 292, 294, 295, 296,
 297, 303, 312, 329, 331, 342, 345, 346,
 347, 349, 350, 351, 352, 353, 357, 359,
 387, 390, 391, 410, 413, 418, 425, 426,
 427, 430, 435, 441, 442, 443, 451, 453,
 454, 457
 Tabu Status, 132, 133, 134, 149, 174, 198,
 199, 201, 285, 347, 400
 Tabu-Tenure, 99, 100, 150, 152, 153, 220,
 221, 222, 251
 Tardiness, 233

Target Analysis, 425, 437, 438, 439
 Taxonomy, 285, 286, 289, 290, 334
 Time Windows, 65, 154, 290, 294, 321,
 409
 Transportation, 2, 61, 146, 320, 333, 335,
 336, 337, 340, 341, 343, 351, 414

U

Uncapacitated Facility Location, 191, 192

V

Variety Metrics, 373

W

Warehouse Location, 193, 334, 335
 Weak Lagrangean Duality, 444