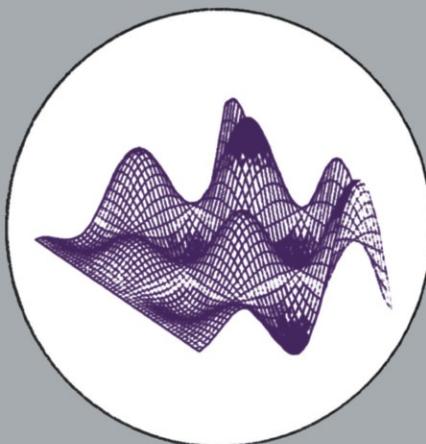


META-HEURISTICS: Theory & Applications

Edited by
Ibrahim H. Osman
James P. Kelly



KLUWER ACADEMIC PUBLISHERS

META-HEURISTICS:

Theory and Applications

META-HEURISTICS:

Theory & Applications

edited by

Ibrahim H. Osman
and
James P. Kelly



Kluwer Academic Publishers
Boston/London/Dordrecht

Distributors for North America:

Kluwer Academic Publishers
101 Philip Drive
Assinippi Park
Norwell, Massachusetts 02061 USA

Distributors for all other countries:

Kluwer Academic Publishers Group
Distribution Centre
Post Office Box 322
3300 AH Dordrecht, THE NETHERLANDS

Library of Congress Cataloging-in-Publication Data

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

ISBN-13: 978-1-4612-8587-8 e-ISBN-13: 978-1-4613-1361-8
DOI: 10.1007/978-1-4613-1361-8

Copyright © 1996 by Kluwer Academic Publishers

Softcover reprint of the hardcover 1st edition 1996

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, mechanical, photo-copying, recording, or otherwise, without the prior written permission of the publisher, Kluwer Academic Publishers, 101 Philip Drive, Assinippi Park, Norwell, Massachusetts 02061

Printed on acid-free paper.

Table of Contents

- Ch. 1 **Ibrahim H. Osman and James P. Kelly** "*Meta-Heuristics: An Overview*" 1

GENETIC ALGORITHMS

- Ch. 2 **David Levine** "*A Parallel Genetic Algorithm for the Set Partitioning Problem*" 23
- Ch. 3 **Zbigniew Michalewicz** "*Evolutionary Computation and Heuristics*" 37
- Ch. 4 **Heinz Mühlenbein and Hans-Michael Voigt** "*Gene Pool Recombination in Genetic Algorithms*" 53
- Ch. 5 **Mutsunori Yagiura and Toshihide Ibaraki** "*Genetic and Local Search Algorithms as Robust and Simple Optimization Tools*" 63

NETWORKS AND GRAPHS

- Ch. 6 **Geoff Craig, Mohan Krishnamoorthy and M. Palaniswami**
"*Comparison of Heuristic Algorithms for the Degree Constrained Minimum Spanning Tree*" 83
- Ch. 7 **Rafael Martí** "*An Aggressive Search Procedure for the Bipartite Drawing Problem*" 97
- Ch. 8 **Yazid M. Sharaiha and Richard Thaiss** "*Guided Search for the Shortest Path on Transportation Networks*" 115

SCHEDULING AND CONTROL

- Ch. 9 **H. Abada and E. El-Darzi** "*A Metaheuristic for the Timetabling Problem*" 133
- Ch. 10 **Peter Brucker and Johann Hurink** "*Complex Sequencing Problems and Local Search Heuristics*" 151

Ch. 11	Mauro Dell'Amico, Silvano Martello and Daniele Vigo	" <i>Heuristic Algorithms for Single Processor Scheduling with Earliness and Flow Time Penalties</i> "	167
Ch. 12	Gregor P. Henze, Manuel Laguna and Moncef Krarti	" <i>Heuristics for the Optimal Control of Thermal Energy Storage</i> "	183
Ch. 13	Helmut E. Mausser and Stephen R. Lawrence	" <i>Exploiting Block Structure to Improve Resource-Constrained Project Schedules</i> "	203
Ch. 14	Helena Ramalhinho Lourenço and Michiel Zwijnenburg	" <i>Combining the Large-Step Optimization with Tabu-Search: Application to The Job-Shop Scheduling Problem</i> "	219
Ch. 15	Takeshi Yamada and Ryohei Nakano	" <i>Job-Shop Scheduling by Simulated Annealing Combined with Deterministic Local Search</i> "	237

SIMULATED ANNEALING

Ch. 16	Mark A. Fleischer & Sheldon H. Jacobson	" <i>Cybernetic Optimization by Simulated Annealing: An Implementation of Parallel Processing Using Probabilistic Feedback Control</i> "	249
Ch. 17	Jean-Luc Lutton and Emmanuelle Philippart	" <i>A Simulated Annealing Algorithm for the Computation of Marginal Costs of Telecommunication Links</i> "	265
Ch. 18	Norman M. Sadeh and Sam R. Thangiah	" <i>Learning to Recognize (Un) Promising Simulated Annealing Runs: Efficient Search Procedures for Job Shop Scheduling and Vehicle Routing</i> "	277
Ch. 19	Mike B. Wright and Richard C. Maret	" <i>A Preliminary Investigation into the Performance of Heuristic Search Methods Applied to Compound Combinatorial Problems</i> "	299

TABU SEARCH

Ch. 20	Ahmed S. Al-Mahmeed	" <i>Tabu Search, Combination and Integration</i> "	319
---------------	----------------------------	---------------------------------------------------------------	-----

Ch. 21	Roberto Battiti, Giampietro Tecchiolli and Paolo Tonella " <i>Vector Quantization with the Reactive Tabu Search</i> "	331
Ch. 22	Diane Castelino and Nelson Stephens " <i>Tabu Thresholding for the Frequency Assignment Problem</i> "	343
Ch. 23	Mauro Dell'Amico and Francesco Maffioli " <i>A New Tabu Search Approach to the 0-1 Equicut Problem</i> "	361
Ch. 24	Kathryn A. Dowsland " <i>Simple Tabu Thresholding and the Pallet Loading Problem</i> "	379
Ch. 25	Fred Glover and Gary A. Kochenberger " <i>Critical Event Tabu Search for Multidimensional Knapsack Problems</i> "	407
Ch. 26	Fred Glover, John M. Mulvey and Kjetil Hoyland " <i>Solving Dynamic Stochastic Control Problems in Finance Using Tabu Search with Variable Scaling</i> "	429
Ch. 27	S. Hanafi, A. Freville and A. El Abdellaoui " <i>Comparison of Heuristics for the 0-1 Multidimensional Knapsack Problem</i> "	449
Ch. 28	Arne Løkketangen and Fred Glover " <i>Probabilistic Move Selection in Tabu Search for Zero-One Mixed Integer Programming Problems</i> "	467
Ch. 29	Lutz Sondergeld and Stefan Voß " <i>A Star-Shaped Diversification Approach in Tabu Search</i> "	489
Ch. 30	Michel Toulouse, Teodor G. Crainic and Michel Gendreau " <i>Communication Issues in Designing Cooperative Multi-Thread Parallel Searches</i> "	503
Ch. 31	Fan T. Tseng " <i>A Study on Algorithms for Selecting r Best Elements from An Array</i> "	523
Ch. 32	Vicente Valls, M. Ángeles Pérez, M. Sacramento Quintanilla " <i>A Modified Tabu Thresholding Approach for the Generalised Restricted Vertex Colouring Problem</i> "	537

Ch. 33 David L. Woodruff	<i>"Chunking Applied to Reactive Tabu Search"</i>	555
Ch. 34 Martin Zachariasen and Martin Dam	<i>"Tabu Search on the Geometric Traveling Salesman Problem"</i>	571

TRAVELING SALESMAN PROBLEMS

Ch. 35 Irène Charon and Olivier Hudry	<i>"Mixing Different Components of Metaheuristics"</i>	589
Ch. 36 H.M.M. ten Eikelder, M.G.A. Verhoeven, T.W.M. Vossen and E.H.L. Aarts	<i>"A Probabilistic Analysis of Local Search"</i>	605
Ch. 37 Jean-Yves Potvin and François Guertin	<i>"The Clustered Traveling Salesman Problem: A Genetic Approach"</i>	619

VEHICLE ROUTING PROBLEMS

Ch. 38 Richard W. Eglese and Leon Y. O. Li	<i>"A Tabu Search Based Heuristic for Arc Routing with a Capacity Constraint and Time Deadline"</i>	633
Ch. 39 Hassan Ghaziri	<i>"Supervision in the Self-Organizing Feature Map: Application to the Vehicle Routing Problem"</i>	651
Ch. 40 César Rego and Catherine Roucairol	<i>"A Parallel Tabu Search Algorithm Using Ejection Chains for the Vehicle Routing Problem"</i>	661
Ch. 41 Paolo Toth and Daniele Vigo	<i>"Fast Local Search Algorithms for the Handicapped Persons Transportation Problem"</i>	677

Preface

The objective of this volume is to consolidate works in Operations Research, Management Science, Artificial Intelligence, Computer Science, and related fields to further the understanding of basic principles and new developments in Metaheuristics, including Genetic Algorithms, Neural Networks, Simulated Annealing, Tabu Search, and other approaches for solving hard combinatorial optimization. The included papers are a reviewed subset of the papers presented at the 1995 Meta-Heuristics International Conference (MIC) held in Breckenridge, Colorado, USA, 22-26 July 1995.

MIC-95 Committee:

Fred Glover General Chair University of Colorado Boulder, Colorado	James P. Kelly Organizing Chair University of Colorado Boulder, Colorado	Ibrahim H. Osman Programme Chair University of Kent Canterbury Kent, UK
-----------------------------------------------------------------------------	-----------------------------------------------------------------------------------	----------------------------------------------------------------------------------

The International Scientific Program Committee members are:

Alain Hertz, Switzerland	Heinz Mühlenbein, Germany
Bennet Fox, USA	Jan Karel Lenstra, Netherlands
Bruce L. Golden, USA	Jun Wang, USA
Catherine Roucairol, France	Luk N. Van Wassenhove, France
Celso R. Ribeiro, Brazil	Martine Labb��, Belgium
Chris Potts, UK	Paolo Toth, Italy
Colin Reeves, UK	Rainer Burkard, Austria
Darrell Whitley, USA	Ramesh Sharda, USA
David Johnson, USA	Richard Eglese, UK
Dominique de Werra, Switzerland	Silvano Martello, Italy
Erwin Pesch, Germany	Stefan Vo��, Germany
Emile Aarts, Netherlands	Teodor G. Crainic, Canada
Francesco Maffioli, Italy	Toshihide Ibaraki, Japan
Gilbert Laporte, Canada	Zbigniew Michalewicz, USA
Harvey Greenberg, USA	

We would like to thank all the referees:

E. H.L. Aarts	J. Hurink	C. Rego
D. Abramson	T. Ibaraki	C. C. Ribeiro
R. Alvares	L. Ingber	E. Rolland
R. Battiti	D. S. Johnson	C. Roucairol
A. Van Breedam	M. Krishnamoorthy	R. A. Russell
P. Brucker	M. Labb��	F. Semet
J. Chakrapani	M. Laguna	Y. M. Sharaiha
I. Charon	G. Laporte	R. Sharda
A. Colorni	J. K. Lenstra	J. Skorin-Kapov
D. Costa	D. Levine	P. Soriano
T. Crainic	A. L��kketangen	E. Taillard
M. Dell'Amico	H. Ramalhinho Louren��o	G. Tecchiolli
P. Djang	L. Lu	S. R. Thangiah
M. Dorigo	J.-L. Lutton	B. Thiesse
U. Dorndorf	F. Maffioli	P. Toth
K. A. Dowsland	V. Maniezzo	M. Toulouse
R. W. Eglese	S. Martello	M. Trubian
E. El-Darzi	H. Mausser	V. Valls
E. Faulkenauer	Z. Michalewicz	M. G. A. Verhoeven
C. Fleurent	H. M��hlenbein	D. Vigo
B. Fox	W. Nujiten	S. Vo��
F. Glover	I. H. Osman	J. Wang
B. L. Golden	M. Otto	D. Whitley
P. Greistorfer	E. Pesch	D. Woodruff
F. Guertin	M. Pirlot	M. Wright
A. Hertz	C. N. Potts	J. Xu
G. Hoscheit	J.-Y. Potvin	T. Yamada
J. Huh	C. R. Reeves	S. Zenios

We would like to thank the College of Business at the University of Colorado and, the Institute of Mathematics and Statistics at Kent University, United Kingdom, for providing staff and financial support. We also would like to thank Gary Folven and Carolyn Ford of Kluwer Academic Publishers for their editorial assistance in producing this volume. Finally, we especially wish to thank Winnie Bartley and Dolores Randall of the University of Colorado for their invaluable assistance in producing this book.

Editors: I.H. Osman and J.P. Kelly

Meta-Heuristics: An Overview

Ibrahim H. Osman
Institute of Mathematics and Statistics
University of Kent, Canterbury
Kent CT2 7NF, U.K
E-mail: I.H.Osman@ukc.ac.uk

&

James P. Kelly
School of Business
Campus Box 419
University of Colorado
Boulder, CO 80309, USA
E-mail: James.Kelly@colorado.edu

Abstract:

Meta-heuristics are the most recent development in approximate search methods for solving complex optimization problems, that arise in business, commerce, engineering, industry, and many other areas. A meta-heuristic guides a subordinate heuristic using concepts derived from artificial intelligence, biological, mathematical, natural and physical sciences to improve their performance. We shall present brief overviews for the most successful meta-heuristics. The paper concludes with future directions in this growing area of research.

Keywords: Adaptive search, Approximate algorithms, Combinatorial optimization, Genetic algorithms, Heuristics, Hybrids, Neural networks, Simulated annealing, Tabu search.

1 Introduction

Meta-heuristics have developed dramatically since their inception in the early 1980s. They have had widespread success in attacking a variety of practical and difficult combinatorial optimization problems. These families of approaches include but are not limited to greedy random adaptive search procedure, genetic algorithms, problem-space search, neural networks, simulated annealing, tabu search, threshold algorithms, and their hybrids. They incorporate concepts based on biological evolution, intelligent problem solving, mathematical and physical sciences,

nervous systems, and statistical mechanics. In the remaining part of the paper, we shall give briefs on: combinatorial optimization problems and meta-heuristic types. We conclude with trends and future directions.

2 Combinatorial Optimization

Over the years, a great deal of effort has been invested in the field of combinatorial optimization theory in which approximate algorithms, often called heuristic algorithms, have become an important area of research and applications. Most practical problems which have finite or countable infinite number of alternative solutions can be formulated as combinatorial optimization problems. Combinatorial optimization is defined in Lawler (1976) as follows:

"Combinatorial optimization is the mathematical study of finding an optimal arrangement, grouping, ordering, or selection of discrete objects usually finite in numbers."

A typical combinatorial minimization problem can be specified by a set of instances. Each instance is associated with a solution space Ω , a *feasible* space X ($X \subseteq \Omega$) that is defined by the problem constraints, an *infeasible* region $\Omega \setminus X$ and an *objective* function C that assigns a real cost value to each $S \in \Omega$, i.e. $C: \Omega \rightarrow \mathbb{R}$. Generally, a combinatorial optimization problem is represented as:

$$\begin{aligned} P: \quad & \text{Minimize (or Maximize)} \quad C(S) \\ & \text{subject to } S \in X \subseteq \Omega. \end{aligned}$$

The sets X and Ω are discrete and can be defined by a set of *decision variables*. These variables can have different integer values depending their roles in the formulation of the problem. Their values are not generally given explicitly but defined within certain ranges. The aim in the optimization problem is to find an *optimal* feasible solution $S^* \in X$ such $C(S^*) \leq C(S')$ for all $S' \in X$ where $X \subseteq \Omega$.

Typical examples of practical combinatorial optimization problems are the location problems, the travelling salesman problem, variants of the assignment and scheduling problems, the circuit and facility layout problems, the set partitioning/covering problems and the vehicle routing problems. We refer to the excellent books by Nemhauser and Wolsey (1988), Papadimitriou and Steiglitz (1982) and Williams (1990) for detailed theory and applications as well as for an introduction to model building in mathematical programming and combinatorial optimization.

Combinatorial optimization problems are normally **easy** to describe but **difficult** to solve. The theory of *computational complexity* was discovered by Cook (1971) and it attempts to categorize the computational requirement of algorithms and classify problems met in practice

as *easy* or *hard*. We call a problem is easy if we can develop an algorithm which solves to optimality every instance of the problem in a polynomial-time complexity bounded by the size of the problem instance. Such a polynomial-time algorithm is to said to be *efficient*. A problem is called *hard (intractable)* if efficient algorithms for solving it do not exist. The results of the computational complexity theory provided evidence that many combinatorial problems are intractable and belong to the class of *NP-complete* (non-deterministic polynomial-time complete) problems, Garey and Johnson (1979). An optimal algorithm for a problem in this class would require a number of computational steps that grows exponentially with the problem size. Hence, only small sized instances can be solved. The common belief is that no efficient algorithm could possibly be ever found to solve these *inherently hard* problems. Thus, heuristics (or approximate algorithms) are considered to be the only practical tools for solving hard combinatorial optimization problems. Excellent guide-lines for designing and reporting on computational experiments with heuristics are reported in Barr *et al.* (1995).

3 Meta-heuristics

Meta-heuristics are a class of approximate methods, that are designed to attack hard combinatorial optimization problems where classical heuristics have failed to be effective and efficient. Meta-heuristics provide general frameworks that allow for creating new hybrids by combining different concepts derived from: classical heuristics; artificial intelligence; biological evolution; neural systems and statistical mechanics. These families of approaches include genetic algorithms, greedy random adaptive search procedure, problem-space search, neural networks, simulated annealing, tabu search, threshold algorithms and their hybrids. For a good tutorial some of the meta-heuristics, we refer to Pirlot (1993) and the comprehensive review in Osman (1995a) where meta-heuristics were defined as follows:

Definition 1: A meta-heuristic is an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search spaces using learning strategies to structure information in order to find efficiently near-optimal solutions.

It should be noted that there are theoretical convergences for some of the meta-heuristics under some assumptions. However, these assumptions can not be satisfied or approximated in most practical applications. Hence the ability to prove optimality is lost and approximate solutions are obtained. Despite this shortcoming, meta-heuristics have been highly successful in finding optimal or near-optimal solutions to many practical applications of optimizations in diverse areas better than their subordinate heuristics. The diversity of these applications can be found in: the bibliographies, Osman and Laporte (1995), Laporte and Osman (1995b), Stewart, Liaw and White (1994), Nissen (1993), Collins, Eglese and Golden (1988); the edited volumes, Laporte and Osman

(1995a), Pesch and Voss (1995), Glover *et al.* (1993); the books, Aarts and Lenstra (1995), Osman and Kelly (1995), Rayward-Smith (1995), Michalewicz (1994), Holland (1994), Reeves (1993), Davis (1991), Goldberg (1989) and Aarts and Korst (1989), van Laarhoven and Aarts (1987); the proceedings, Osman and Kelly (1995), Albrecht, Reeves and Steel (1993), Forrest (1993), Belew and Booker (1991) and Shaffer (1989) and Grefenstette (1985). In the remainder of this section, we shall briefly describe the most popular meta-heuristics and subordinate methods starting with the classical neighbourhood (local) search.

3.1 Classical Neighbourhood Search

Classical neighbourhood (or local search) methods form a general class of approximate heuristics based on the concept of exploring the vicinity of the current solution. Neighbouring solutions are generated by a *move-generation mechanism*. These solutions are *selected* and *accepted* according to some pre-defined criteria. Given a combinatorial minimization problem defined by the triplet (X, Ω, C) representing the sets of feasible and infeasible solutions, the objective function to be minimized, let us then define the following:

Definition 2: A neighbourhood structure is a mapping $N: \Omega \rightarrow \Omega$, which defines for each $S \in \Omega$ a set $N(S) \subseteq \Omega$ of solutions that are in the vicinity of S . The set $N(S)$ is called the neighbourhood of S and each $S' \in N(S)$ is called a neighbour of S .

Definition 3: A move-generation mechanism generates the set of neighbours changing one attribute or a combination of attributes of an given instance S . A move-generation is a transition from a solution S to another solution $S' \in N(S)$ in one step (or iteration).

Definition 4: A candidate list of solutions $\overline{N(S)}$ is a set of solutions S' that satisfies the pre-defined acceptance and admission criteria, $\overline{N(S)} \subseteq N(S)$.

An attribute of a solution is often associated with a binary variable in P or the objective value, etc. In order to design a local search algorithm, one typically needs to specify the following choices:

- (i) **Initialization:** how an initial feasible solution S can be generated.
- (ii) **Generation mechanism:** how (feasible) neighbours S' from $N(S)$ can be obtained and what move-generation mechanism to use.
- (iii) **Acceptance and selection strategies:** if there are many neighbours that can be accepted, which S' from $\overline{N(S)}$ should be selected. There are two selection criteria:
 - (a) A *first-accept* (FA) strategy selects the first neighbour S' that

enters $\overline{N(S)}$.

(b) A *best-accept* (BA) strategy selects the "best" S' in $\overline{N(S)}$.

Once S' is identified, it replaces S as the current solution, otherwise S is retained and the search continues.

- (iv) **Stopping Test:** the stopping criterion depends on the selection strategies and the type of the local search algorithms.

In Table 1, we shall sketch the basic steps of a classical local search descent procedure:

Step 1: Get an initial solution S and compute its objective $C(S)$.
Step 2: While there is an untested neighbour $S' \in \overline{N(S)}$, execute the following:
(a) Generate sequentially a trial $S' \in \overline{N(S)}$ and compute $C(S')$.
(b) If $C(S') < C(S)$ then, S' replaces S as a current solution. Otherwise, retains S and Step 2 is repeated.
Step 3: Terminate the search and return S as the local optimal solution.

Table 1. A local search descent procedure.

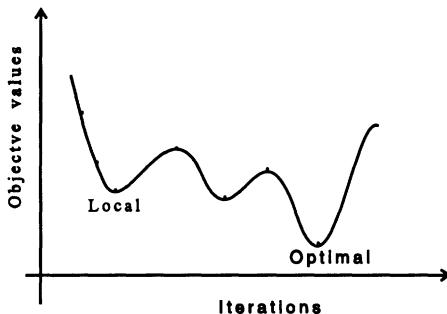


Figure 1. Global and local optimum.

The *local* optimum produced by any local search descent procedure can be very far from optimality (see Figure 1). Local search methods perform a *blind* search since they only accept sequentially solutions which produce reductions in the objective function value. They do not make use of any information gathered during the execution of the algorithm. They depends heavily on initial solutions and the neighbourhood generation mechanisms. In order to avoid some of the local search descent

disadvantages, while maintaining the simplicity and generality of the approach, some deterministic and probabilistic guidances strategies were developed. The details are left to the appropriate sections.

3.2 Genetic algorithms

Genetic algorithms (GA) are a class of adaptive search methods based on a abstract model of natural evolution. They were developed by Holland in the 1970s and only recently their potential for solving combinatorial optimization problems has been explored. The basic idea is to maintain a population of candidate solutions that evolves under a selective pressure that favours better solutions. Hence they can be viewed as a class of local search based on a *solution-generation* mechanism operating on attributes of a set of solutions rather than attributes of a single solution by the move-generation mechanism of local search methods.

Generally, GA is an iterative procedure that operates on a finite population of N chromosomes (solutions). The chromosomes are fixed strings with binary values (0 or 1 called alleles) at each position (or locus). Each chromosome of the population is evaluated according to a fitness function. Members of the population are selectively interbred in pairs to produce offspring. The fitter a member of the population the more likely it is to produce offspring. Genetic operators are used to facilitate the breeding process that results in offspring inheriting properties from their parents. The offspring are evaluated and placed in the population, possibly replacing the weaker members of the last generation. Thus, the search mechanism consists of three phases: *evaluation* of the fitness of each chromosome, *selection* of the parent chromosomes, and *application genetic operators* to the parent chromosomes. The new chromosomes resulting from these operations form the population for the next generation and the process is repeated until the system ceases to improve. A simple genetic algorithm is presented in Table 2.

- | |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> Step 1: Generate an initial population of strings (genotypes). Step 2: Assign a fitness value to each string in the population. Step 3: Pick a pair of strings (parents) for breeding. Step 4: Put offspring produced in a temporary population (mating pool). Step 5: If the temporary population is not full, then go to Step 3. Step 6: Replace the current population with the temporary population and a portion of the current population. Step 7: If the termination criterion is not met, go to Step 2. |
|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Table 2. A simple genetic algorithm

The *initial population* in GA is normally generated at random. However, it can be seeded by heuristic solutions since they may be of a good quality. Ways to avoid early convergences were developed and are reviewed in Jones (1993). The *set of genetic operators* in GA includes Holland's *Cross-over*, *Mutation*, and *Inversion*. In cross-over, one or two cut-points are selected at random and the information between the two chromosomes are exchanged. Mutation is a background operator to prevent GA from becoming trapped in a local optimum by selecting a random position and changing its value. In inversion, two-cut-points are randomly chosen and the order of the bits are reversed.

Genetic algorithms encounters a number of problems when solving combinatorial problems. They may fail to find satisfactory solutions for many reasons. The GA binary encoding/decoding has been found unsuitable and normal cross-over operations often lead to many infeasible solutions. The failure should be addressed for GA to be successful by revising the problem representation, designing special cross-over operations and using repair schemes for infeasibility or penalty functions. The most successful GA implementations use hybrid GA combinations with other techniques including: branch and bound, Nagar, Heragu, and Haddock (1995); local search descent, Glass and Potts (1995), Thangiah, Osman and Sun (1994), Kolen and Pesch (1994) and Thangiah *et al.* (1993); simulated annealing and tabu search, Glover, Kelly and Laguna (1995), Thangiah, Osman and Sun (1994) and Lin, Koa and Hsu (1993). For more details on GAs principles, we refer to Mulhenbein (1995) and Holland (1992).

3.3 Neural Networks

Neural networks have claimed intriguing success in problems whose structures can be exploited by processes linked to those of associated memory. They have been used to preform prediction and classification or recognize patterns. They have generally performed less impressively in combinatorial optimization problems. So far, neural network heuristics are not competitive with the best meta-heuristics for any class of problems.

The interest in using neural networks in combinatorial optimization problems was pioneered by the work of Hopfield and Tank (1985) on the traveling salesman problem. The Hopfield networks consist of a set of competing connected elements. The competing elements are logic units with binary states and are linked by symmetric connections. Each connection is associated with a weight representing the interconnection between units when both are "on". A consensus function (energy function) assigns to each configuration of the network a real value. The units may change their states in order to maximize the consensus. A state change of an individual unit is determined by a deterministic response function of the states of its adjacent units. If the response function is determined by a given probability function, this associated randomized version of the Hopfield network is called a Boltzmann machine. The

challenge of the model is to choose an appropriate network structure and corresponding connection strengths such that the problem of finding near optimal solutions of the optimization problem is equivalent to finding maximal configurations of a Hopfield network.

The *elastic net algorithms* (Durbin and Willshaw, 1987) and the *self-organizing feature maps* (Kohonen, 1988) are two other approaches to using neural network for combinatorial optimization. The results in the literature indicated that the elastic net is better than the self-organizing map on the traveling salesman problem and both are better than the Hopfield network, (Potvin, 1993). New results are provided on the traveling salesman problem using hierarchical elastic net (Vakutinsky and Golden, 1995) and on the vehicle routing problem using the self-organising map (Ghaziri, 1995) showed some promising results. For excellent reviews on neural networks in combinatorial optimization problems, we refer to Bruke and Ignizio (1992) and Looi (1992).

3.4 Simulated annealing

The simulated annealing (SA) meta-heuristic has its origins in statistical mechanics. The interest began with the work of Kirkpatrick, Gelatt and Vecchi (1983) and Cerny (1985). The SA algorithm is based on the analogy between the annealing process of solids and the problem of solving combinatorial optimization problems. In condensed matter physics, annealing denotes a process in which a solid (crystal) in a *heat bath* is melted by increasing the temperature of the heat bath to a high maximum value at which all molecules of the crystal randomly arrange themselves into a liquid phase. The temperature of the melted crystal is then reduced until the crystal structure is frozen (reaches a low *ground state*). If the cooling is done very quickly by dropping the external temperature immediately to zero and by not allowing the crystal to reach thermal equilibrium for the intermediate temperature values, widespread irregularities and defects can be locked into the crystal structure. This is known as rapid *quenching* and results in meta-stable structures.

In the annealing process, if the temperature of the solid reaches zero, no state transition can lead to a state of higher energy and the solid *freezes*. In practice, crystals are grown by a process of *careful annealing* in which the temperature (T) descends slowly through a series of levels. At each level, the crystal is kept long enough to reach equilibrium at that temperature. As long as the temperature is not zero, uphill (increased energy) moves remain possible. By keeping the temperature from getting too far from the current energy level, we can hope to avoid local optima, until we are relatively close to the ground state.

Metropolis *et al.* (1953) proposed a *Monte-Carlo* method to simulate the evolution to thermal equilibrium of a crystal for a fixed value of the temperature T . The method generates sequences of states of the crystal in the following way: Given the current state, S , of the crystal, characterised by the position of its molecules, a small perturbation is applied

by a small displacement of a randomly chosen molecule. If the difference in the energy level, Δ , between the current state and the newly generated state S' is negative - the new perturbed state is of a lower energy - S' is accepted and the process continues from the new state. When $\Delta \geq 0$, S' is accepted with probability $\theta \leq e^{-\Delta/T}$; otherwise S is retained as the current state. This acceptance rule is referred to as the Metropolis (or simulation) criterion. The name simulated annealing thus refers to the use of the *simulation* techniques in conjunction with an *annealing* (or cooling) *schedule* of the declining temperatures. A great variety of theoretical and practical cooling schedules have been suggested in the literature. These cooling schedules have been classified and reviewed in Osman and Christofides (1994). The SA algorithm steps are summarised in Table 3.

<p>Step 1: Generate an initial random or heuristic solution S. Set an initial temperature T, and other cooling schedule parameters.</p> <p>Step 2: Choose randomly $S' \in N(S)$ and compute $\Delta = C(S') - C(S)$</p> <p>Step 3: If:</p> <ul style="list-style-type: none"> (i) S' is better than S ($\Delta < 0$), or (ii) S' is worse than S but "accepted" by the randomization process at the present temperature T, i.e. $e^{\left(\frac{-\Delta}{T}\right)} > \theta, \text{ (where } 0 < \theta < 1 \text{ is a random number).}$ <p>Then replace S by S'.</p> <p>Else Retain the current solution S.</p> <p>Step 4: Update the temperature T depending on a set of rules, including:</p> <ul style="list-style-type: none"> (i) The cooling schedule used (ii) Whether an improvement was obtained in Step 3 above, (iii) Whether the neighbourhood $N(S)$ has been completely searched. <p>Step 5: If a "stopping test" is successful stop, else go to Step 2.</p>

Table 3. Simulated annealing procedure

Simulated annealing has been combined with other approaches like genetic algorithms, tabu search, see Osman and Laporte (1995), Thangiah, Osman and Sun (1994), Fox (1993) and Lin, Koa and Hsu (1993). A combination of simulated annealing and tabu search concepts resulted a successful hybrid algorithm in that the neighbourhood was searched sequentially and systematically using a first selection strategy rather than the random or aggressive selections to provide a search intensification while the simulated annealing acceptance controlled by a non-monotonic

cooling schedule is used to provide a search diversification. The hybrid approach does not explicitly use memory structures except for the temperature resettings. Successful implementations of such approach showed that it provides better results than the simple simulated annealing or tabu search, see Osman (1995b, 1993), Hasan and Osman (1995), Osman and Christofides (1994), Thangiah, Osman and Sun (1994). This approach is very similar to the probabilistic tabu search approach by Glover and Lokketangen (1994) except that the later uses instead a ranking probability function that favours the selection of best moves. Finally, SA applications are numerous and diverse and its applications in OR are reviewed, Dowsland (1995), Koulamas, Antony and Jaen (1994), Ingber (1993), Eglese (1990), van Laarhoven and Aarts (1987).

3.5 Tabu search

Tabu search is an iterative meta-heuristic search procedure introduced by Glover (1986) for solving optimization problems. It is based on the general tenets of intelligent problem solving. Tabu search shares the ability to guide a subordinate heuristic (such as the local search procedure) to continue the search beyond a local optimum where the embedded heuristic would normally become trapped. The process in which the TS method seeks to transcend local optimality is based on an *aggressive* evaluation that chooses the *best* available move at each iteration even when this move may result in a degradation of the objective value. TS begins in the same way as an ordinary local search, proceeding iteratively from one solution S to another S' until a chosen termination criterion is satisfied.

Tabu search goes beyond local search by employing a strategy of modifying $N(S)$ as the search progresses, i.e. replacing it by another neighbourhood $\overline{N(S)}$, known as a candidate list of solutions. Tabu search uses special short term and long term memory structures to determine $\overline{N(S)}$ and to organize the search. The solutions admitted to $\overline{N(S)}$ by these memory structures are determined by identifying solutions encountered over a specified horizon. Attributes that changed over the specified horizon are recorded in one or more lists and are given *tabu-active* status. The short-term memory structure manages what goes in and out of the lists and explicitly identifies the status of these attributes. Solutions that contain tabu-active elements or combinations of these attributes become *tabu*. The aim is to prevent certain solutions from the recent past from belonging to $\overline{N(S)}$ and from being revisited. Moreover, TS uses explicit as well as attributive memories. Explicit memory records complete solutions, typically elite (good) solutions visited during the search. The special solutions are introduced at strategic intervals to enlarge $\overline{N(S)}$.

In TS, the short-term memory keeps track of solution attributes that have changed in the recent past and is called *recency-based memory*. To exploit this memory, selected attributes that occur in solutions recently visited are designated tabu-active. The duration that an attribute remains tabu-active is called *tabu-tenure*. Tabu tenure can vary over different intervals of time. This variation would make it possible to create different kinds of trade off between short and longer term strategies. An important element of flexibility in tabu search is the introduction of tabu solutions to $\overline{N(S)}$ by means of an *aspiration* criterion. The tabu status can be overridden if certain conditions are met, e.g. a tabu solution which has a solution better than any previously seen solutions deserves to be considered *admissible* to $\overline{N(S)}$. Another *aggressive* aspect exists in TS. For situations where $\overline{N(S)}$ is large or its elements are expensive to evaluate a smaller set (candidate list) is sampled. The generic TS steps are depicted in Table 4.

- | |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>Step 1: Get an initial solution S, and initialize the short-term memory structures.</p> <p>Step 2: Select the <i>best admissible</i> solution, $S_{best} \in \overline{N(S)} \subseteq N(S)$; ($S_{best}$ is the best of all $S' \in \overline{N(S)}$: S' is not <i>tabu-active</i>, or passed an aspiration test).</p> <p>Step 3: Update the current solution $S = S_{best}$,
update the short-term memory structures.</p> <p>Step 4: Repeat Step 2 and Step 3 until a <i>stopping criterion</i> is satisfied.</p> |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Table 4. Tabu search procedure

The core of the above procedure is the short-term memory which manages the interplay between various TS strategies. In any implementation, the elements of the short-term memory structures must be defined: attributes of accepted solutions to be stored, types of tabu restrictions and static or dynamic rules for tabu-tenure, selection strategies to choose from the set of candidate solutions, aspiration criteria and stopping rules. Finally, other advanced strategies introduced to enhance the basic tabu search are: diversification of the long term memory (Soriano and Gendreau 1995, Husbscher and Glover 1994, and Kelly, Laguna and Glover 1994) versus the intensification of the short-term memory, probabilistic move selection strategies (Osman 1995b and 1993, Hasan and Osman 1995, Glover and Lokketangen 1994, Osman and Christofides 1994, Thangiah, Osman and Sun 1994, and Glover 1990) rather than the aggressive elitism strategy, compound moves and data structures (Rego and Roucairol 1994, Glover, Pesch and Osman 1994, Osman 1993, Osman and Salhi 1993, and Glover 1992), Hashing functions (Hasan and Osman 1995, and Woodruff and Zemel 1993) to identify tabu solutions, dynamic and random rules for tabu-tenure (Battiti and Tecchiolli 1995, Battiti and Tecchiolli 1994, Dammeyer and Voss 1993, Gendreau, Hertz

and Laporte 1994, Taillard 1991 and Glover 1990) and other hybrids including; Glover, Kelly and Laguna(1995), Fox (1993), Moscato (1993), and Battiti and Tecchiolli (1992). For more details on recent TS developments, we refer to Glover (1995, 1994, 1990, 1989).

3.6 Threshold algorithms

Threshold algorithms are variants of simulated annealing and tabu search meta-heuristics. They operate by reference to probability, implicit thresholds without much usage memory structures. There are two types of threshold algorithms: *threshold accepting* and *Tabu thresholding*.

Threshold accepting was introduced by Ducek and Scheurer (1990) as a deterministic version of simulated annealing. At iteration k , the Metropolis acceptance criterion of simulated annealing is replaced by a deterministic threshold acceptance in which a neighbour $S' \in N(S)$ is accepted if $\Delta = C(S') - C(S)$ is less than T_k a threshold value. A threshold acceptance algorithm uses a non-increasing sequence of positive thresholds ($T_k \geq 0, T_{k-1} \geq T_k$, and $T_k \rightarrow 0$). The aim is to allow non-improving neighbours to be accepted in a limited way while at the end of the search where $T_k = 0$, the algorithm performs a random descent local search algorithm. The difficulty with thresholding accepting approach is to find an appropriate sequence of the threshold values. For more details and variants, we refer to (Glass and Potts 1995, Sinclair 1993, Charon and Hurdy (1993), and Ducek and Scheuer 1990). The basic procedure is sketched in Table 5.

<p>Step 1: Get an initial solution S and compute its objective $C(S)$. Initialize the sequence of the threshold values $(T_k)_{k=1,\dots,K}$.</p>
<p>Step 2: At iteration k, perform a fixed number of N_k trial moves to generate N_k neighbours.</p> <ul style="list-style-type: none"> (a) Randomly generate a trial solution, $S' \in N(S)$ and compute $C(S')$. (b) If $C(S') - C(S) < T_k$ then, S' replaces S as a current solution, otherwise, retain S and repeat Step (a).
<p>Step 3: Update the value of T_k, if the termination criterion is met, stop with S is the local optimal solution, Otherwise go toe Step 2.</p>

Table 5. The threshold acceptance procedure

Tabu thresholding was proposed by Glover (1993) for implementing the basic ideas of tabu search without explicit use of memory structures. The tabu thresholding algorithm embodies the principle of aggressive exploration of the search space in a non-monotonic way with additional probabilistic elements. In an overview, the method consists of two alternating phases: an improving phase and a mixed phase. The improving phase allow only improving moves to be accepted. This phase is terminated with a local optimum. The mixing phase accepts both non-improving and improving moves. The choices of moves in the two phases are governed by employing candidate list strategies to isolate subsets of moves to be examined at each iteration and by a probabilistic best criterion. The number of iterations of the mixed phases is controlled by a tabu timing parameter t which may vary randomly between lower and upper limits. Successful applications of tabu thresholding are found in Valls, Marti, Lino (1995) and a number of papers in this book. The tabu threshold procedure is sketched in Table 6.

Improving Phase.

Step 1: Generate an initial solution S .

Step 2: Generate a subset $\overline{N(S)}$ of available neighbours, and let $S_{BEST} \subseteq \overline{N(S)}$ be the set of improving solutions. If $\overline{N(S)}$ is empty and S_{BEST} does not exist, expand $\overline{N(S)}$ so that S_{BEST} is not empty or $\overline{N(S)} \equiv N(S)$ at which this phase is terminated.

Step 3: If $\overline{N(S)}$ is not empty, choose a solution $S_{best} \in S_{BEST}$ probabilistically to be the current solution and go to Step 2.

Mixed Phase.

Step 3: Select a value for the tabu-timing parameter t .

Step 4: Generate a subset $\overline{N(S)}$ of current available neighbours, and choose a solution $S_{best} \in \overline{N(S)}$ probabilistically to be the current solution.

Step 5: Continue Step 4 for t iterations, or until an aspiration criterion is satisfied and return to Step 2.

Table 6. The tabu thresholding procedure

3.7 Problem-space methods

Problem-space methods are a class of heuristics superimposed on fast problem-specific constructive procedures. The aim is to generate many different starting solutions that can be improved by local search methods. This class contains three type of methods: greedy random adaptive search procedure (GRASP); Problem-space heuristic and incomplete construction/improvement procedure.

The GRASP method was developed by Feo and Resende (1989). It combines the power of greedy heuristics, randomization, and local search procedures. The randomization concept within GRASP was motivated by the work of Hart and Shogan (1987). GRASP is an iterative procedure and each iteration consists of two phases: a construction phase and a local search improvement phase. The construction phase is itself an iterative greedy and adaptive process. The method is adaptive because the greedy function takes into account previous decisions in the construction when considering the next choice. There is a probabilistic component in GRASP that is applied to the selection of elements during the construction phase. An initial solution is built by considering one element at a time. Each element is added using a greedy function that minimizes (maximizes) the objective function. At each stage of the construction, a *candidate list*, consisting of high-quality elements according to the adaptive greedy function, is formed. Then, the next element to enter the constructed solution is randomly selected from the current candidate list. The iterative process is continued until a constructive initial solution is obtained. The improvement phase typically includes a local search procedure to improve the constructed initial solution. Both the construction and improvement phases are repeated for a given number of iterations after which the procedure terminates with a local optimum. It should be noted that if the best element is selected at each stage of the construction phase, this constructed solution would be the same solution of the greedy heuristic without the added features of GRASP. The GRASP procedure has been implemented successfully to many problems (Kontorvadis and Bard 1995, Laguna, Feo and Erlod 1994, Feo, Resende and Smith 1994, and Feo and Resende 1994).

Problem-space and heuristic-space methods were introduced by Storer, Wu and Vaccari (1992). They are defined in *problem-space* and *heuristic-space* rather than in the solution space of local search methods. Both methods require the application of some fast, base heuristic H at each iteration to generate feasible solutions with automatically computed objective. In the problem-space procedure, the base heuristic is applied to perturbed data of the original problem. A problem-space neighbourhood can be defined as all perturbed problems within a certain distance (ϵ) from the original data. If P_0 is the vector of the original data, D is a random vector of perturbations, then the neighbourhood, N_p , of problems close to the original one is defined as: $N_p = \{P_0 + D, \forall D \text{ with } \|D\| < \epsilon\}$. Applying the base heuristic H to each instance of N_p would generate another neighbourhood of solutions N_s . Each solution in N_s is evaluated using the original data. On the other hand, the heuristic-space procedure relies on the ability to define parameterized versions of the base heuristic H . For each setting of the parameters, a slightly different version of the heuristic is created. The parameters of the heuristics define the search

space. In both approaches, generated solutions may be improved by any local search procedures. For more details, we refer to Storer, Flanders and Wu (1995).

Finally, an **incomplete construction/improvement** method were used by Russell (1995). In this approach, an improvement phase was periodically invoked during the construction process to the partial emerging solution rather than the standard implementation after the complete construction. This approach was successful in getting good solutions for the vehicle routing with time windows in Russell (1995) and Chiang and Russell (1995).

4 Conclusion

As mentioned at the beginning, this paper was to introduce the most successful approaches for solving combinatorial optimization problems. Meta-heuristics have proven their power in obtaining high quality solutions to many real world complex problems, and we expect this number to continue to grow in the future. We encourage and advocate their usages due to simplicity, robustness and easy of modification. However, the design of a good meta-heuristic remains an art. It depends on the skill and experience of the designer and the empirical computational experiments. In recent literature, there are calls for combining the best features and aspects of each meta-heuristic looking for such unified meta-heuristics. These calls invite more research and empirical analysis.

5 References

- E.H.L. Aarts and J.K. Lenstra (1995) *Local Search Algorithms*, John Wiley & Sons, Chichester.
- E.H.L. Aarts and J. Korst (1989) *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*, John Wiley and Sons Chichester.
- R.F. Albrecht, C.R. Reeves and N.C. Steele (1993) *Artificial Neural Nets and Genetic Algorithms*. Proceedings of the International Conference in Innsbruck, Austria, Springer-Verlag.
- B.S. Barr, B.L. Golden, J.P. Kelly, M.G.C. Resende and W.R. Stewart (1995) Designing and reporting on computational experiments with heuristic methods, *Journal of Heuristic*, Forthcoming.
- R. Battiti and G. Tecchiolli (1995), The continuous reactive tabu search: blending combinatorial optimization and stochastic search for global optimization, *Annals of Operational Research*, **60** Forthcoming.
- R. Battiti and G. Tecchiolli (1994) The reactive tabu search, *ORSA Journal on Computing* **6**, 126-140.
- R. Battiti and G. Techioilli (1992) Parallel biased search for combinatorial optimization: genetic algorithms and tabu, *Microprocessors and Microsystems* **16**, 351-367.

- R.K. Belew and L.B Booker (1991) *Proceedings of the Fourth International Conference on Genetic Algorithms* Morgan Kaufmann Publishers, San Mateo, CA.
- L.I. Burke and J.P. Ignizio (1992) Neural networks and operations research: an overview, *Computers & Operations Research*, **19**, 179-189.
- V. Cerny (1985) A thermodynamical approach to the travelling salesman problem: an efficient simulated annealing algorithm, *Journal of Optimization Theory and Applications* **45**, 41-51.
- I. Charon and O. Hurdy (1993) The noisy method: a new method for combinatorial optimization, *Operations Research Letters*, **14**, 133-137.
- W.-C Chiang and R.A. Russell (1995) simulated annealing metaheuristics for the vehicle routing problems with time windows, *Annals of Operations Research*, **60**, Forthcoming.
- N.E. Collins, R.W. Eglese, and B.L. Golden (1988) Simulated annealing - An annotated bibliography, *American Journal of Mathematical and Management Sciences*, **9**, 209-307.
- S.A. Cook (1971) The complexity of theorem-proving procedures, in: *Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing*, 151-158.
- F. Dammeyer and S. Voss (1993) Dynamic tabu list management using the reverse elimination method, *Annals of Operations Research*, **41**, 29-46.
- L. Davis (1991) *Handbooks of Genetic Algorithms*, Van Nostrand ReinHold, New York.
- K. Dowsland (1995) Variants of simulate annealing for practical problem solving, in: *Applications of Modern Heuristic Methods*, Ed. V. Rayward-Smith, Alfred Waller Ltd, in association with UNICOM, Henley-on-Thames.
- G. Ducek and T. Scheuer (1990) Threshold accepting: a general purpose optimization algorithm, *Journal of Computational Physics*, **90**, 161-175.
- R. Durbin and D. Willshaw (1987) An analogue approach to the TSP using elastic net method, *Nature*, 689-691.
- R.W. Eglese (1990) Simulated annealing: a tool for operational research, *European Journal of Operational Research*, **46**, 271-281.
- T.A Feo, M.G.C. Resende, and S.H. Smith (1994) A greedy randomized adaptive search procedure for maximum independent set, *Operations Research* **42**, 860.
- T.A. Feo and M.G.C. Resende (1994) Greedy randomized adaptive search procedures, Working paper, AT&T Bell Laboratories, Murray Hill, NJ 07974.

- T.A. Feo, and M.G.C. Resende, (1989) A probabilistic heuristic for a computationally difficult set covering problem, *Operations Research Letters*, **8**, 67-71.
- S. Forrest (1993) *Proceedings of an International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, San Mateo, CA.
- B. Fox (1993) Integrating and accelerating tabu search, simulated annealing and genetic algorithms, *Annals of Operations Research*, **41**, 47-67.
- M.R. Garey and D.S. Johnson (1979) *Computers and Intractability: a guide to the theory of NP-completeness*, W.H. Freeman and Company, New York.
- M. Gendreau, A. Hertz, and G. Laporte (1994) A tabu search heuristic for the vehicle routing problem, *Management Science* **40**, 1276-1290.
- H. Ghaziri (1995) Supervision in the self-organizing feature map: Application to the vehicle routing in: *Metaheuristics: the state of the art* 1995, Eds. I.H. Osman and J.P. Kelly, Kluwers Academic Publishers, Boston.
- C.A. Glass and C.N. Potts (1995) A comparison of local search methods for flow shop scheduling, *Annals of Operations Research*, **60** Forthcoming.
- F. Glover, E. Pesch and I.H. Osman (1994) Efficient facility layout Planning, Working paper, Graduate School of Business, University of Colorado, Boulder, CO 80309.
- F. Glover, J.P. Kelly, and M. Laguna (1995) Genetic algorithms and tabu search - hybrids for optimization, *Computers & Operations Research*, **22**, 111.
- F. Glover, M. Laguna, E. Taillard and D. de Werra (1993) *Tabu Search*, Annals of Operations Research, **43**, J.C. Baltzer Science Publishers, Basel, Switzerland.
- F. Glover (1995) Tabu search fundamentals and uses, Working paper, Graduate School of Business, University of Colorado, Boulder, CO 80309.
- F. Glover (1994) Tabu search: Improved solution alternatives, in: *Mathematical Programming, State of the Art 1994*, Eds. J.R. Brige and K.G. Murty, The University of Michigan, Michigan.
- F. Glover and A. Lokketangen (1994) Probabilistic tabu search for Zero-One mixed integer programming problems, Working paper, Graduate School of Business, University of Colorado, Boulder, CO 80309.
- F. Glover (1993) Tabu thresholding: improved search by non-monotonic trajectories, Working Paper, Graduate School of Business, University of Colorado, Boulder, Colorado 80309-0419 USA.
- F. Glover (1992) Ejection chains, reference structures and alternating path methods for the travelling salesman problems, Working paper, Graduate School of Business, University of Colorado, Boulder, CO 80309.

- F. Glover (1990) Tabu search: part II, *ORSA Journal on Computing*, **2**, 4-32.
- F. Glover (1989) Tabu search: part I, *ORSA Journal on Computing*, **1**, 190-206.
- F. Glover (1986) Future paths for integer programming and links to artificial intelligence, *Computers and Operations Research*, **1**, 533-549.
- D.E. Goldberg (1989) *Genetic algorithms in search, Optimization, and Machine Learning*, Addison-Wesley, New York.
- J.J. Grefenstette (1985) *Proceedings of an International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, San Mateo, CA.
- J. Hart and A. Shogan (1987) Semi-greedy heuristics: an empirical study, *Operations Research Letters*, **6**, 107-114.
- M. Hasan and I.H. Osman (1995) Local search algorithms for the maximal planar layout problem, *International Transactions in Operations Research*, **2**, 89-106.
- J.H. Holland (1992) *Adaptation in Natural and Artificial Systems*, 2nd edition, The University of Michigan Press, Ann Arbor.
- J.J. Hopfield and D.W. Tank (1985) Neural computation of decisions in optimization problems, *Biological Cybernetics*, **52**, 141-152.
- R. Hubscher and F. Glover (1994) Applying tabu search with influential diversification to multiprocessor scheduling, *Computers & Operations Research*, **21**, 867.
- L. Ingber (1993) Simulated annealing - practice versus theory, *Mathematical And Computer Modelling* **18**, 29.
- A.J. Jones (1993) Genetic algorithms and their applications to the design of neural networks, *Neural Computing and Applications*, **1**, 32-45.
- J.P. Kelly, M. Laguna and F. Glover (1994) A study of diversification strategies for the quadratic assignment problem, *Computers & Operations Research*, **21**, 885.
- P. Soriano and M. Gendreau (1995) Diversification strategies in tabu search algorithm for the maximum clique problem, *Annals of Operations Research*, **60** Forthcoming.
- S. Kirkpatrick, C.D. Gelatt, and P.M. Vecchi (1983) Optimization by simulated annealing, *Science*, **220**, 671-680.
- T. Kohonen (1988) *Self-organization and Associative Memory*, Springer-Verlag, Berlin.
- A. Kolen and E. Pesch (1994) Genetic local search in combinatorial optimization, *Discrete Applied Mathematics*, **48**, 273.
- G. Kontoravdis and J.F. Bard (1995) Improved heuristics for the vehicle routing problem with time windows, *ORSA Journal on Computing*, **7**, Forthcoming.
- C. Koulamas, S.R. Antony and R. Jaen, (1994) A survey of simulated annealing application to operations research problems, *OMEGA*, **22**, 41-56.

- P.J. van Laarhoven, and E.H.L. Aarts (1987) *Simulated Annealing: Theory and Applications*, Reidl, Dordrecht.
- M. Laguna, T.A. Feo and H.C. Elrod (1994) A greedy randomized adaptive search procedure for the two-partition problem, *Operations Research*, **42**, 677-687.
- G. Laporte and I.H. Osman (1995a) *Meta-heuristics in Combinatorial Optimization*, (J.C. Baltzer Science Publishers, Basel, Switzerland, 1995).
- G. Laporte and I.H. Osman (1995b) Routing Problems: A bibliography, *Annals of Operations Research*, Forthcoming.
- E.L. Lawler (1976) *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York.
- F.T. Lin, C.Y. Kao and C.C. Hsu (1993) Applying the genetic approach to simulated annealing in solving some NP-hard problems, *IEEE Transactions on Systems Man and Cybernetics* **23**, 1752-1567.
- C.K. Looi (1992) Neural network methods in combinatorial optimization, *Computers & Operations Research*, **19**, 191-208.
- W. Metropolis, A. Roenbluth, M. Rosenbluth, A. Teller, and E. Teller (1953) Equation of the state calculations by fast computing machines, *Journal of Chemical Physics*, **21**, 1087-1092.
- Z. Michalewicz (1994) *Genetic Algorithms+ Data Structures= Evolution Programs*, Springer-Verlag, End edition, New York.
- H. Mulhenbein (1995) Evolutionary algorithms: theory and application, in: *Local Search in Combinatorial Optimization*, Eds E.H.L. Aarts and J.K. Lenstra, John Wiley and Sons, Chichester.
- A. Nagar, S.S. Heragu and J. Haddock (1995) A combined branch-and-bound and genetic algorithm based approach for a flowshop scheduling problem, *Annals of Operations Research*, **60**, Forthcoming.
- G.L. Nemhauser and L.A. Wolsey (1988) *Integer and Combinatorial Optimization*, John Wiley & Sons, New York.
- V. Nissen (1993) Evolutionary algorithm in Management Science: an overview and list of references, Working Paper 9309, University of Goettingen, Gosslerstr. 12a, D-37073 Goettingen, Germany, can be obtained by anonymous ftp gwdu03.gwdg.de in pub/msdos/reports/wi.
- I.H. Osman (1995a) An introduction to Meta-heuristics, in: *Operational Research Tutorial Papers Series*, Annual Conference OR37 - Canterbury 1995, Eds. C. Wildson and M. Lawrence (Operational Research Society Press, 1995).
- I.H. Osman (1995b) Heuristics for the generalized assignment problem: simulated annealing and tabu search approaches, *OR Spektrum*, **17**, 2/3 Forthcoming.
- I.H. Osman (1993) Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problems, *Annals of Operations Research*, **41**, 421-451.

- I.H. Osman and J.P. Kelly (1995), *Proceedings of the Metaheuristics International Conference*, Breckenridge, Colorado, (Kluwers Academic Publishers, Boston, July 1995).
- I.H. Osman and G. Laporte (1995) Metaheuristics for combinatorial optimization problems: An annotated bibliography, *Annals of Operational Research*, 60, Forthcoming.
- I.H. Osman and N. Christofides (1994) Capacitated clustering problems by hybrid simulated annealing and tabu search, *International Transactions in Operational Research*, 1, 317-336.
- I.H. Osman and S. Salhi, Local search strategies for the mix fleet vehicle routing problem, Working paper UKC/IMS/OR93/8, Institute of Mathematics and Statistics, University of Kent, Canterbury, UK (1993).
- C.H. Papadimitriou and K. Steiglitz (1982) *Combinatorial optimization: algorithms and complexity*, Prentice Hall, New York.
- E. Pesch and S. Voss (1995) *Applied Local Search*, OR Spektrum, 17, Springer-Verlag, Germany.
- M. Pirlot (1992) General local search heuristics in combinatorial optimization: a tutorial, *Belgian Journal of Operations, Statistics, and Computer Science*, 32, 7-67.
- J.-Y. Potvin (1993) the travelling salesman problem: A neural network perspective, *ORSA Journal on Computing*, 5, 328-348.
- V.J. Rayward-Smith (1995) *Applications of Modern Heuristic Methods*, Alfred Waller Ltd, in association with UNICOM, Henley-on-Thames.
- C.R. Reeves (1993) *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Publications, Oxford.
- R. Rego and C. Roucairol (1994) An efficient implementation of ejection chain procedures for the vehicle routing problem, Working paper RR-94/44, Laboratoire PRISM, Universite de Versailles, France.
- R.A. Russell (1995) Hybrid heuristics for the vehicle routing problem with time windows, *Transportation Science*, Forthcoming.
- J.D. Schaffer (1989) *Proceedings of the Third International Conference on Genetic Algorithms* Morgan Kaufmann Publishers, San Mateo, CA.
- M. Sinclair (1993) Comparison of the performance of modern heuristics for combinatorial problem on real data, *Computers & Operations Research*, 20, 687-695.
- B.S. Stewart, C.-F Liaw and C.C. White (1994) A bibliography of heuristic search through 1992, *IEEE Transactions on Systems, Man, and Cybernetics*, 24, 268-293.
- R.H. Storer, S.D. Wu and R. Vaccari (1992) New search spaces for sequencing problems with application to job shop scheduling, *Management Science*, 38, 1495-1509.

- R.H. Storer, S.W. Flanders and S.D. Wu (1995) Problem space local search for number partitioning, *Annals of Operations Research*, 60, Forthcoming.
- E. Taillard (1991) Robust taboo search for the quadratic assignment problem, *Parallel Computing*, **17**, 443-455.
- S.R. Thangiah, I.H. Osman, R. Vinayagamoorthy and T. Sun (1993) Algorithms for vehicle routing problems with time deadlines, *American Journal of Mathematical and Management Sciences*, **13**, 323-354
- S.R. Thangiah, I.H. Osman and T. Sun (1994) Metaheuristics for vehicle routing problems with time windows, Working paper UKC/IMS/OR94/8, Institute of Mathematics and Statistics, University of Kent, Canterbury. Forthcoming in Annals of OR.
- A.I. Vakutinsky and B.L. Golden (1995) A hierarchical strategy for solving traveling salesman problem using elastic nets, *Journal of Heuristics*, Forthcoming.
- V. Valls, R. Marti and P. Lino (1995) A tabu thresholding algorithm for arc crossing minimization in bipartite graphs, *Annals of Operations Research*, 60, Forthcoming.
- H.P. Williams (1990) *Model Building in Mathematical Programming*, 3rd edition, John Wiley & Sons, New York.
- D.L. Woodruff and Z. Zemel (1993) Hashing vectors for tabu search, *Annals of Operations Research*, **41**, 123-138.

A Parallel Genetic Algorithm for the Set Partitioning Problem*

David Levine

*Argonne National Laboratory
Mathematics and Computer Science Division
9700 South Cass Avenue
Argonne, Illinois 60439, U.S.A.
E-mail: levine@mcs.anl.gov*

Abstract:

This paper describes a parallel genetic algorithm developed for the solution of the set partitioning problem—a difficult combinatorial optimization problem used by many airlines as a mathematical model for flight crew scheduling. Tests on forty real-world set partitioning problems were carried out on an IBM SP parallel computer. We found that performance, as measured by the quality of the solution found and the iteration on which it was found, improved as additional subpopulations were added to the computation. With larger numbers of subpopulations the genetic algorithm was regularly able to find the optimal solution to problems having up to a few thousand integer variables. A notable limitation was the difficulty solving problems with many constraints.

Key Words: airline crew scheduling, combinatorial optimization, parallel genetic algorithms

1. Introduction

This paper describes a parallel genetic algorithm developed for the solution of the set partitioning problem—a difficult combinatorial

*This work was supported by the Mathematical, Information, and Computational Sciences Division subprogram of the Office of Computational and Technology Research, U.S. Department of Energy, under Contract W-31-109-Eng-38.

optimization problem used by many airlines as a mathematical model for flight crew scheduling. The set partitioning problem (SPP) may be stated mathematically as

$$\text{Minimize } z = \sum_{j=1}^n c_j x_j \quad (1)$$

subject to

$$\sum_{j=1}^n a_{ij} x_j = 1 \quad \text{for } i = 1, \dots, m \quad (2)$$

$$x_j = 0 \text{ or } 1 \quad \text{for } j = 1, \dots, n, \quad (3)$$

where a_{ij} is binary for all i and j , and $c_j > 0$. The goal is to determine values for the binary variables x_j that minimize the objective function z .

The best-known application of the SPP is airline crew scheduling. In this formulation each row represents a flight leg (a takeoff and landing) and the columns represent legal round-trip rotations (pairings) an airline crew might fly. Associated with each assignment of a crew to a particular flight leg is a cost, c_j . The matrix elements a_{ij} are defined by

$$a_{ij} = \begin{cases} 1 & \text{if flight leg } i \text{ is on rotation } j \\ 0 & \text{otherwise.} \end{cases} \quad (4)$$

Airline crew scheduling is a very visible and economically significant problem, as indicated by the amount and variety of recent work [1, 2, 3, 4, 5, 6]. Our interest was in studying the effectiveness of a parallel genetic algorithm for solving the SPP.

2. The Genetic Algorithm

A solution to the SPP problem is given by specifying values for the binary decision variables x_j . The value of one (zero) indicates that column j is included (not included) in the solution. This solution may be represented by a binary vector \mathbf{x}^\dagger with the interpretation that x_j is one (zero) if bit j is one (zero) in the binary vector. Representing an SPP solution in a genetic algorithm (GA) is straightforward and natural. A bit in a GA string is associated with each

[†]We use \mathbf{x} interchangeably as the solution to the SPP problem or as a bitstring in the GA population.

column j . The bit is one if column j is included in the solution, and zero otherwise.

For our evaluation function we used

$$\sum_{j=1}^n c_j x_j + \sum_{i=1}^m \lambda_i \Phi_i(\mathbf{x}), \quad (5)$$

where

$$\Phi_i(\mathbf{x}) = \begin{cases} 1 & \text{if constraint } i \text{ is infeasible,} \\ 0 & \text{otherwise.} \end{cases}$$

The first term is the SPP objective function. The second term, the penalty function,[‡] penalizes constraint violations. The scalar λ_i is a weighting factor which we set to the largest c_j from the columns that intersected row i .

Our early experience [7, 8] was that the GA by itself had trouble finding optimal (often even feasible) solutions. Therefore, we developed a search heuristic (ROW) to assist the GA in finding feasible, or near-feasible, strings to apply the GA operators to. For each GA iteration, one of the m rows is randomly selected. If no column covers the selected row, one is randomly chosen from the set of columns that can cover this row, and it is set to one. If the row is feasible, we set to zero the column that covers this row, and to one the first column found (if any) that also covers this row, but *only* if the change further minimizes Eq. (5). If more than one column covers the row, we randomly set all but one to zero.

The sequential GA was the result of significant research and experimentation [7, 8]. The first step is to pick a random string and apply the ROW heuristic to it. Next, two parent strings are selected by holding two binary tournaments, and a random number, $r \in [0, 1]$, is generated. If r is less than the crossover probability of .6, we create two new offspring via uniform crossover and randomly select one of them to insert in the population. Otherwise, we randomly select one of the two parent strings, make a copy of it, and apply mutation to complement bits in the copy with probability $1/n$. In either case, the new string is tested to see whether it

[‡]We also tested a penalty term that depended on the magnitude of the constraint violation [8] without observing any significant difference

duplicates a string already in the population. If it does, it undergoes (possibly additional) mutation until it is unique. The least-fit string in the population is deleted, the new string is inserted, and the population is reevaluated.

The parallel genetic algorithm is based on an island model. In the island model GA a population is divided into several subpopulations, each of which is randomly initialized and independently runs a sequential GA on its own subpopulation. Occasionally, fit strings migrate between subpopulations. The intent is to share the information contained in the more fit strings between subpopulations, and to maintain subpopulation diversity.

The island model requires choices for several communication parameters. In this work the best string in a subpopulation was selected to migrate to a neighboring subpopulation every 1,000 iterations. The string to delete was selected by holding a probabilistic binary tournament (with parameter .4). The logical topology of the subpopulations was a two-dimensional toroidal mesh. Each processor exchanged strings with its four neighbors, alternating between them each migration generation.

3. Parallel Experiments

The parallel computer we used for our experiments was an IBM SP with 128 nodes, each of which consisted of an IBM RS/6000 Model 370 workstation processor, 128 MB of memory, and a 1 GB disk. The SP uses a high-performance switch for connecting the nodes. The SP supports the distributed-memory programming model. Our code was written in C and used IBM's EUI-H message-passing software.

Each test problem was run once using 1, 2, 4, 8, 16, 32, 64, and 128 subpopulations. Each subpopulation was of size 100. As additional subpopulations were added to the computation, the total number of strings in the *global* population increased. Our assumption was that even though we were doubling the computational effort required whenever we added subpopulations, by mapping each subpopulation to an SP processor, the total elapsed time would remain relatively constant. A run was terminated either when the optimal solution was found,[§] or when all subpopulations had per-

[§]For these tests, the value of the (known) optimal solution was stored in the

formed 100,000 iterations. Each time a parallel run was made, all subpopulations were randomly seeded by using the random number generator described in [9].

3.1 Test Problems

To test the parallel genetic algorithm, we selected a subset of forty problems (most of the small- and medium-sized problems, and a few of the larger problems) from the test set used by Hoffman and Padberg [6]. The test problems are given in Table 1, where they have been sorted according to increasing numbers of columns. The columns in this table are the test problem name, the number of rows and columns in the problem, the number of nonzeros in the A matrix, the optimal objective function value for the LP relaxation, and the objective function value of the optimal integer solution.

Many of these problems are “long and skinny”; that is, they have few rows relative to the number of columns (it is common in the airline industry to generate subproblems of the complete problem that contain only a subset of the flight legs the airlines are interested in, solve the subproblems, and try to create a solution to the complete problem by piecing together the subproblems). Of these test problems, all but two of the first thirty have fewer than 3,000 columns (**nw33** and **nw09** have 3,068 and 3,103 columns, respectively.) The last ten problems are significantly larger, not just because there are more columns, but also because there are more constraints. One reason we did not test all of the larger problems is that we did not have access to a matrix reduction capability such as typically is used (e.g., as in [6]) to preprocess large problems and can significantly reduce the size.

3.2 Experimental Results

Table 2 shows the percentage from optimality of the best solution found in any of the subpopulations as a function of the number of subpopulations. An entry of “O” in the table indicates the optimal solution was found. An entry of “X” in the table means no integer feasible solution was found by any of the subpopulations. A numerical entry is the percentage from the optimal solution of the best *feasible* solution found by any subpopulation after the iteration

program, which tested the best feasible solution found each iteration against the optimal solution and stopped if they were the same.

Table 1: Test Problems

Problem Name	No. Rows	No. Cols	No. Nonzeros	LP Optimal	IP Optimal
nw41	17	197	740	10972.5	11307
nw32	19	294	1357	14570.0	14877
nw40	19	404	2069	10658.3	10809
nw08	24	434	2332	35894.0	35894
nw15	31	467	2830	67743.0	67743
nw21	25	577	3591	7380.0	7408
nw22	23	619	3399	6942.0	6984
nw12	27	626	3380	14118.0	14118
nw39	25	677	4494	9868.5	10080
nw20	22	685	3722	16626.0	16812
nw23	19	711	3350	12317.0	12534
nw37	19	770	3778	9961.5	10068
nw26	23	771	4215	6743.0	6796
nw10	24	853	4336	68271.0	68271
nw34	20	899	5045	10453.5	10488
nw43	18	1072	4859	8897.0	8904
nw42	23	1079	6533	7485.0	7656
nw28	18	1210	8553	8169.0	8298
nw25	20	1217	7341	5852.0	5960
nw38	23	1220	9071	5552.0	5558
nw27	22	1355	9395	9877.0	9933
nw24	19	1366	8617	5843.0	6314
nw35	23	1709	10494	7206.0	7216
nw36	20	1783	13160	7260.0	7314
nw29	18	2540	14193	4185.3	4274
nw30	26	2653	20436	3726.8	3942
nw31	26	2662	19977	7980.0	8038
nw19	40	2879	25193	10898.0	10898
nw33	23	3068	21704	6484.0	6678
nw09	40	3103	20111	67760.0	67760
nw07	36	5172	41187	5476.0	5476
nw06	50	6774	61555	7640.0	7810
aa04	426	7195	52121	25877.6	26402
k101	55	7479	56242	1084.0	1086
aa05	801	8308	65953	53735.9	53839
nw11	39	8820	57250	116254.5	116256
aa01	823	8904	72965	55535.4	56138
nw18	124	10757	91028	338864.3	340160
k102	71	36699	212536	215.3	219
nw03	59	43749	363939	24447.0	24492

limit was reached. A blank entry means that the test was not made (usually because of a resource limit or an abort). Entries marked with a superscript ^a did not complete and are discussed further in [8].

Table 2 shows the percentage from optimality of the best solution found in any of the subpopulations as a function of the number of subpopulations. An entry of “O” in the table indicates the optimal solution was found. An entry of “X” in the table means no integer feasible solution was found by any of the subpopulations. A numerical entry is the percentage from the optimal solution of the best *feasible* solution found by any subpopulation after the iteration limit was reached. A blank entry means that the test was not made (usually because of a resource limit or an abort). Entries marked with a superscript ^a did not complete and are discussed further in [8].

Optimal solutions were found for all but one of the first 32 problems. For approximately two-thirds of these problems, only four subpopulations were necessary before the optimal solution was found. For the other one-third of the problems, additional subpopulations were necessary in order to find the optimal solution. In general, the best solution improved as additional subpopulations participated, even if the optimal solution was not reached. Problem **nw06**, with 6,774 columns, was the largest problem for which an optimal solution was found. For three larger problems, **k101**, **k102**, and **nw03**, although optimal solutions were not found, increasingly better integer solutions were found as subpopulations were added.

For six problems we were unable to find *any* integer feasible solutions. For three of these problems (**nw10**, **nw11**, and **nw18**) the GA was able to find infeasible strings with higher fitness than feasible ones and had concentrated its search on those strings. In other words, the GA was simply exploiting the fact that for these problems the penalty term used in the evaluation function is not strong enough. For the other three problems (**aa01**, **aa04**, and **aa05**) this was not the case. On average, near the end of a run an (infeasible) solution had an evaluation function value approximately twice that of the optimal integer solution.

Table 3 contains the first iteration on which some subpopulation

Table 2: Percent from Optimality vs. Number of Subpopulations

Problem Name	Number of Subpopulations							
	1	2	4	8	16	32	64	128
nw41	O	O	O	O	O	O	O	O
nw32	0.0006	O	0.0006	O	O	O	O	O
nw40	O	O	0.0036	O	O	O	O	O
nw08	X	0.0219	O	O	O	O	O	O
nw15	O	O	O	0.0001	4.4285	O	O	O
nw21	0.0037	0.0037	O	O	O	O	O	O
nw22	0.0735	0.0455	0.0252	O	O	O	O	O
nw12	0.1375	0.0912	0.0332	0.0218	0.0094	O	O	0.0246 ^a
nw39	0.0425	O	O	O	O	O	O	O
nw20	0.0091	O	O	O	O	O	O	O
nw23	O	O	O	O	0.0006	O	O	O
nw37	O	0.0163	O	O	O	O	O	O
nw26	0.0011	O	O	O	O	O	O	O
nw10	X	X	X	X	X	X	X	X ^a
nw34	0.0203	0.0214	O	O	O	O	O	O
nw43	0.0831	0.0626	0.0350	O	O	O	O	O
nw42	0.2727	0.0229	O	O	O	O	O	O
nw28	0.0469	O	O	O	O	O	O	O
nw25	0.1040	0.1137	O	O	O	O	O	O
nw38	0.0323	O	O	O	O	O	O	O
nw27	0.0818	0.0567	O	0.0039	O	O	O	O
nw24	0.0826	0.0215	O	0.0015	0.0038	O	O	O
nw35	0.0770	O	0.0171	O	O	O	O	O
nw36	0.0038	0.0010	0.0194	0.0010	0.0019	O	O	O
nw29	0.0580	O	O	0.0116	O	O	O	O
nw30	0.1116	O	O	O	O	O	O	O
nw31	0.0069	0.0069	O	O	O	O	O	O
nw19	0.1559	0.1332	0.0715	0.0880	0.0148	O	O	O
nw33	0.0128	O	O	O	O	O	O	O
nw09	0.0398	X	0.0363	0.0231	0.0155	0.0151	0.154 ^a	O
nw07	0.3089	O	O	O	O	O	O	O
nw06	2.0755	0.2532	O	0.1779	0.0448	0.0291	O	O
aa04	X	X	X	X	X			
k101	0.0524	0.0359	0.0368	0.0303	0.0239	0.0184	0.0082	0.0092 ^a
aa05	X	X	X		X			
nw11	X	X	X	X	X	X	X	X
aa01	X	X	X	X	X	X	X	
nw18	X	X	X	X	X	X	X	X
k102	0.1004 ^a	0.1004	0.0502	0.0593	0.0593 ^a		0.0410	0.0045
nw03	0.2732	0.1125 ^a	0.1371 ^a					0.0481

Table 3: First Optimal Iteration vs. Number of Subpopulations

Problem Name	Number of Subpopulations							
	1	2	4	8	16	32	64	128
nw41	3845	1451	551	623	758	402	398	362
nw32	F	1450	F	3910	2740	2697	2054	1006
nw40	540	1597	F	1658	2268	958	979	696
nw08	X	F	34564	8955	14760	10676	8992	10631
nw15	4593	17157	5560	F	F	929	692	1321
nw21	F	F	7875	3929	4251	1818	1868	2514
nw22	F	F	F	29230	3370	3037	2229	1820
nw12	F	F	F	F	F	62976	34464	F ^a
nw39	F	2345	3738	1079	1396	900	1232	913
nw20	F	2420	3018	5279	27568	2295	2282	1654
nw23	2591	6566	3437	3452	F	1723	2125	1477
nw37	75737	F	1410	1386	1443	1370	835	779
nw26	F	84765	52415	24497	13491	1660	1512	2820
nw10	X	X	X	X	X	X	X	X ^a
nw34	F	F	2443	1142	1422	1110	1417	843
nw43	F	F	F	11004	3237	21069	4696	3296
nw42	F	F	2702	3348	1070	1223	1187	724
nw28	F	903	1897	1232	776	718	371	191
nw25	F	F	2634	70642	4351	5331	1024	1896
nw38	F	68564	27383	1431	1177	1093	603	514
nw27	F	F	610	F	2569	1669	3233	2135
nw24	F	F	908	F	F	11912	2873	4798
nw35	F	3659	F	3182	1876	1224	1158	634
nw36	F	F	F	F	F	3367	2739	4200
nw29	F	17212	5085	F	17146	1368	2243	795
nw30	F	3058	1777	1154	1650	846	866	949
nw31	F	F	1646	3085	1287	1890	1682	732
nw19	F	F	F	F	F	79125	27882	37768
nw33	F	1670	1659	7946	1994	2210	829	873
nw09	F	X	F	F	F	F	F ^a	71198
nw07	F	29033	7459	4020	4831	1874	2543	1935
nw06	F	F	51502	F	F	F	48215	19165
aa04	X	X	X	X	X			
k101	F	F	F	F	F	F	F	F ^a
aa05	X	X	X		X			
nw11	X	X	X	X	X	X	X	X
aa01	X	X	X	X	X	X		
nw18	X	X	X	X	X	X	X	X
k102	F ^a	F	F	F	F ^a		F	F
nw03	F	F ^a	F ^a					F

found an optimal solution. An entry of “F” means a nonoptimal integer feasible solution was found. As a general trend, the first optimal iteration occurs earlier as we increase the number of subpopulations. If we recall that the migration frequency is set to 1,000, it is notable that the number of problems for which an optimal solution is found before any migration takes place grows with the increase in the number of subpopulations. With one subpopulation an optimal solution was found for only one problem before migration occurred. With 128 subpopulations the optimal solution was found for 13 problems before migration occurred.

Table 4 compares the solutions found by `lp_solve`, a public-domain branch-and-bound program, the work of Hoffman and Padberg [6] (the column *HP*), and our work (the column *SSGAROW*). The entry is a “O” if the optimal solution was found, a numerical entry that is the percentage from optimality of the best suboptimal integer feasible solution found, or an “X” if no feasible solution was found. The results for SSGAROW are the best solution found in Table 2.

For the first thirty-two problems all three algorithms found optimal solutions (except SSGAROW on `nw10`). On the larger problems we observe that branch-and-cut solved all problems to optimality. Both `lp_solve` and SSGAROW had trouble with the `aa` problems; neither found a feasible solution to any of the three problems. For the two `k1` problems, SSGAROW was able to find good integer feasible solutions, while `lp_solve` did not find any feasible solutions. For the larger `nw` problems, `lp_solve` did much better than SSGAROW, proving two optimal (`nw11`, `nw03`) and finding a good integer feasible solution to the other. SSGAROW has “penalty troubles” with two of these and on `nw03` computes an integer feasible, but suboptimal solution.

Since the results presented in Table 4 were calculated on different model IBM workstation processors, a direct comparison of CPU timings is not appropriate. However, taking into account the performance of each machine’s processor on standard benchmarks [8], it is possible to make a qualitative performance comparison. In general, we found the branch-and-cut solution times approximately an order of magnitude faster than the branch-and-bound times. On the smaller problems SSGAROW was competitive with branch-and-

Table 4: Comparison of Solution Time

Problem	lp_solve	HP	SSGAROW
nw41	O	O	O
nw32	O	O	O
nw40	O	O	O
nw08	O	O	O
nw15	O	O	O
nw21	O	O	O
nw22	O	O	O
nw12	O	O	O
nw39	O	O	O
nw20	O	O	O
nw23	O	O	O
nw37	O	O	O
nw26	O	O	O
nw10	O	O	X
nw34	O	O	O
nw43	O	O	O
nw42	O	O	O
nw28	O	O	O
nw25	O	O	O
nw38	O	O	O
nw27	O	O	O
nw24	O	O	O
nw35	O	O	O
nw36	O	O	O
nw29	O	O	O
nw30	O	O	O
nw31	O	O	O
nw19	O	O	O
nw33	O	O	O
nw09	O	O	O
nw07	O	O	O
nw06	O	O	O
aa04	X	O	X
k101	X	O	.0092
aa05	X	O	X
nw11	O	O	X
aa01	X	O	X
nw18	.0110	O	X
k102	X	O	.0045
nw03	O	O	.0481

bound, but on some of the larger problems the branch-and-bound times were often an order of magnitude faster than SSGAROW. We do note, however, that the SSGAROW timings were done using a heavily instrumented, unoptimized version of our program, and a number of possible areas for performance improvement exist.

4. Conclusions

Using the hybrid SSGAROW algorithm in an island model was an effective approach for solving real-world SPP problems of up to a few thousand integer variables. For all but one of the thirty-two small and medium-sized test problems the optimal solution was found. For several larger problems, good integer feasible solutions were found. We found two limitations, however. First, for several problems the penalty term was not strong enough. The GA exploited this feature by concentrating its search on infeasible strings that had better evaluations than a feasible string would have had. A second limitation was the fact that three problems had many constraints. For these problems, even though the penalty term seemed adequate, SSGAROW was unable to find a feasible solution.

Adding additional subpopulations (which increase the global population size) was beneficial. When an optimal solution was found, it was usually found on an earlier iteration. In cases where the optimal solution was not found, but a feasible one was (i.e., on the largest test problems), the quality of the feasible solution improved as additional subpopulations were added to the computation. Also notable was the fact that, as additional subpopulations were added, the number of problems for which the optimal solution was found before the first migration occurred continued to increase.

We compared SSGAROW with implementations of branch-and-cut and branch-and-bound algorithms. Branch-and-cut was clearly superior to both SSGAROW and branch-and-bound, finding optimal solutions to all test problems, and in less time. Both SSGAROW and branch-and-bound found optimal solutions to the small and medium-sized test problems; on larger problems the results were mixed.

Acknowledgment

I thank the anonymous referees for their helpful comments. This paper is based on my Ph.D. thesis [8].

References

- [1] D. Abramson, H. Dang, and M. Krishnamoorthy. *A comparison of two methods for solving 0/1 integer programs using a general purpose simulated annealing algorithm*. Technical Research Report 81, Griffith University School of Computing and Information Technology, Australia, 1993.
- [2] R. Anbil, R. Tanga, and E. Johnson. A Global Approach to Crew Pairing Optimization. *IBM Systems Journal*, 31(1):71–78, 1992.
- [3] P. Chu and J. Beasley. *A Genetic Algorithm for the Set Partitioning Problem*. Technical Report, Imperial College, 1995.
- [4] J. Desrosiers, Y. Dumas, M. Solomon, and F. Soumis. *The Airline Crew Pairing Problem*. Technical Report G-93-39, Université de Montréal, 1993.
- [5] M. Fischer and P. Kedia. Optimal Solution of Set Covering/Partitioning Problems Using Dual Heuristics. *Management Science*, 36(6):674–688, 1990.
- [6] K. Hoffman and M. Padberg. Solving Airline Crew-Scheduling Problems by Branch-and-Cut. *Management Science*, 39(6):657–682, 1993.
- [7] D. Levine. A genetic algorithm for the set partitioning problem. In S. Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 481–487, San Mateo, 1993. Morgan Kaufmann.
- [8] D. Levine. *A Parallel Genetic Algorithm for the Set Partitioning Problem*. Ph.D. thesis, Illinois Institute of Technology, Chicago, 1994. Department of Computer Science.
- [9] G. Marsaglia, A. Zaman, and W. Tseng. *Stat. Prob. Letter*, 9(35), 1990.

Evolutionary Computation and Heuristics

Zbigniew Michalewicz

*Department of Computer Science
University of North Carolina
Charlotte, NC 28223, USA
E-mail: zbysek@uncc.edu*

and

*Institute of Computer Science
Polish Academy of Sciences
ul. Ordona 21
01-237 Warsaw, Poland
E-mail: zbysek@ipipan.waw.pl*

Abstract:

Evolutionary computation techniques constitute an important category of heuristic search. Any evolutionary algorithm applied to a particular problem must address the issue of genetic representation of solutions to the problem and genetic operators that would alter the genetic composition of offspring during the reproduction process. However, additional heuristics should be incorporated in the algorithm as well; these heuristic rules provide guidelines for evaluating unfeasible and feasible individuals. This paper surveys such heuristics for discrete and continuous domains and discusses their merits and drawbacks.

Key Words: Constrained optimization, evolutionary computation, genetic algorithms, unfeasible individuals.

1. Introduction

It is generally accepted that any evolutionary algorithm to solve a problem must have five basic components: (1) a genetic representation of

solutions to the problem, (2) a way to create an initial population of solutions, (3) an evaluation function (i.e., the environment), rating solutions in terms of their ‘fitness’, (4) genetic operators that alter the genetic composition of children during reproduction, and (5) values for the parameters (population size, probabilities of applying genetic operators, etc.). The data structure used for representation of solutions to the problem and a set of genetic operators constitute its most essential components. For example, the original genetic algorithms, devised to model *adaptation processes*, mainly operated on binary strings and used recombination operator with mutation as a background operator (Holland 1995). Evolution strategies (Schwefel 1981) used real variables¹ and aimed at *numerical optimization*. Because of that, the individuals incorporated also a set of strategic parameters. Evolution strategies relied mainly on mutation operator (Gaussian noise with zero mean); only recently a discrete recombination was introduced for object variables and intermediate recombination—for strategy parameters. On the other hand, evolutionary programming techniques (Fogel et al. 1966) aimed at building a system to solve *prediction tasks*. Thus they used a finite state machine as a chromosome and five mutation operators (change the output symbol, change a state transition, add a state, delete a state, or change the initial state).² In genetic programming techniques (Koza 1992) the structure which undergoes evolution is a hierarchically structured computer program.³ The search space is a hyperspace of valid programs, which can be viewed as a space of rooted trees.

There are few heuristics to guide a user in selection of appropriate data structures and operators for a particular problem. One popular heuristic in applying evolutionary algorithms to real-world problems is based on modifying the algorithm by the problem-specific knowledge; this problem-specific knowledge is incorporated in chromosomal data structures and specialized genetic operators (Michalewicz 1994). Another possibility is based on hybridization; in (Davis 1991) the author wrote:

“When I talk to the user, I explain that my plan is to hybridize the genetic algorithm technique and the current algorithm by employing the following three principles:

- *Use the Current Encoding.* Use the current algorithm’s encoding technique in the hybrid algorithm.

¹However, they started with integer variables as an experimental optimum-seeking method.

²‘Change’, ‘add’, and ‘delete’ are specialized versions of mutation.

³Actually, Koza has chosen LISP’s S-expressions for all his experiments.

- *Hybridize Where Possible.* Incorporate the positive features of the current algorithm in the hybrid algorithm.
- *Adapt the Genetic Operators.* Create crossover and mutation operators for the new type of encoding by analogy with bit string crossover and mutation operators. Incorporate domain-based heuristics as operators as well.”

There are a few heuristics available for creating an initial population: one can start from a randomly created population, or use an output from some deterministic algorithm to initialize it (with many other possibilities in between these extremes). There are also some general heuristic rules for determining values for the various parameters; for many genetic algorithms applications, population size stays between 50 and 100, probability of crossover—between 0.65 and 1.00, probability of mutation—between 0.001 and 0.01. Additional heuristic rules were used to vary the population size or probabilities of operators during the evolution process.

It seems that one item only from the list of five basic components of the evolutionary algorithm—the evaluation function—usually is taken “for granted”. Indeed, in many cases the process of selection of an evaluation function is straightforward; most research concentrated rather on scaling the values returned by the evaluation function and selection methods (proportional, ranking, tournament selections). Consequently, during the last two decades, many difficult functions have been examined; often they served as test-beds for different selection methods, various operators, different representations, and so forth. However, the process of selection of evaluation function might be quite complex by itself, especially, when we deal with feasible and unfeasible solutions to the problem; several heuristics usually are incorporated in this process. In the paper we examine some of these heuristics and discuss their merits and drawbacks.

The paper is organized as follows. The next section defines a search space and poses a few questions, which serve as the basis for the main discussion (Section 3). Section 4 concludes the paper.

2. Feasible and unfeasible solutions

In evolutionary computation methods the evaluation function serves as the only link between the problem and the algorithm. The evaluation function rates individuals in the population: better individuals have better chances for survival and reproduction. Hence it is essential to define

an evaluation function which characterize the problem in a ‘perfect way’. In particular, the issue of handling feasible and unfeasible individuals should be addressed very carefully: very often a population contains unfeasible individuals but we search for a feasible optimal one. Finding a proper evaluation measures for feasible versus unfeasible individuals often determines the success or failure of the algorithm.

In general, a search space \mathcal{S} consists of two disjoint subsets of feasible and unfeasible subspaces, \mathcal{F} and \mathcal{U} , respectively (see Figure 1).

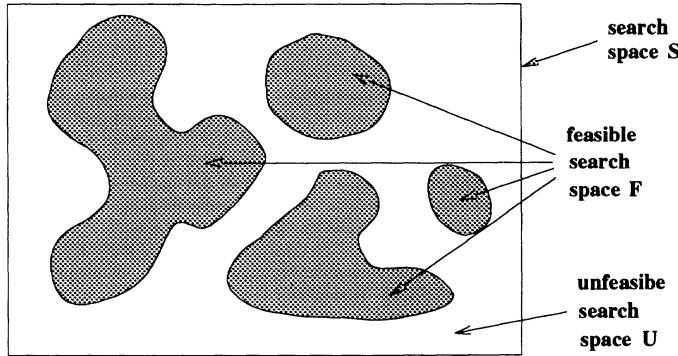


Figure 1: A search space and its feasible and unfeasible parts

We do not make any assumptions about these subspaces; in particular, they need not be convex and they need not be connected (e.g., as it is the case in the example in Figure 1 where feasible part \mathcal{F} of the search space consists of four disjoined subsets). In solving optimization problems we search for a *feasible* optimum. During the search process we have to deal with various feasible and unfeasible individuals; for example (see Figure 2), at some stage of the evolution process, a population may contain some feasible (b, c, d, e, i, j, k, p) and unfeasible individuals (a, f, g, h, l, m, n, o), while the optimum solution is marked by ‘X’.

The presence of feasible and unfeasible individuals in the population influences other parts of the evolutionary algorithm; for example, the elitist selection method should consider a possibility of preserving best *feasible* individual, or just the best individual overall. Further, some operators might be applicable to feasible individuals only. However, the major aspect of such scenario is the need for evaluation of feasible and unfeasible individuals. The problem of how to evaluate individuals in the population is far from trivial. In general, we have to design two evaluations functions, $eval_f$ and $eval_u$, for feasible and unfeasible domains,

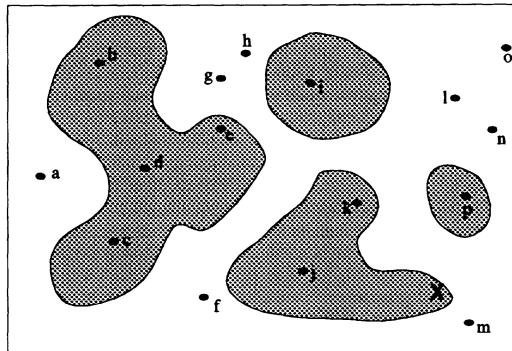


Figure 2: A population of 16 individuals, a – o

respectively. There are many important questions to be addressed (we discuss them in detail in the next section):

- A.** How should two feasible individuals be compared, e.g., ‘c’ and ‘j’ from Figure 2? In other words, how should the function $eval_f$ be designed?
- B.** How should two unfeasible individuals be compared, e.g., ‘a’ and ‘n’? In other words, how should the function $eval_u$ be designed?
- C.** How are the functions $eval_f$ and $eval_u$ related to each other? Should we assume, for example, that $eval_f(s) \succ eval_u(r)$ for any $s \in \mathcal{F}$ and any $r \in \mathcal{U}$ (the symbol \succ is interpreted as ‘is better than’, i.e., ‘greater than’ for maximization and ‘smaller than’ for minimization problems)?
- D.** Should we consider unfeasible individuals harmful and eliminate them from the population?
- E.** Should we ‘repair’ unfeasible solutions by moving them into the closest point of the feasible space (e.g., the repaired version of ‘m’ might be the optimum ‘X’, Figure 2)?
- F.** If we repair unfeasible individuals, should we replace unfeasible individual by its repaired version in the population or rather should we use a repair procedure for evaluation purpose only?
- G.** Since our aim is to find a feasible optimum solution, should we choose to penalize unfeasible individuals?
- H.** Should we start with initial population of feasible individuals and maintain the feasibility of offspring by using specialized operators?

I. Should we change the topology of the search space by using decoders?

J. Should we extract a set of constraints which define feasible search space and process individuals and constraints separately?

Several trends for handling unfeasible solutions have emerged in the area of evolutionary computation. We discuss them in the following section using examples from discrete and continuous domains.

3. Heuristics for evaluating individuals

This section discusses several methods for handling unfeasible solutions for problems from discrete and continuous domains; most of these methods emerged quite recently. Only a few years ago Richardson et al. (1989) claimed: "Attempts to apply GA's with constrained optimization problems follow two different paradigms (1) modification of the genetic operators; and (2) penalizing strings which fail to satisfy all the constraints." This is not longer the case as a variety of methods have been proposed. Even the category of penalty functions consists of several methods which differ in many important details on how the penalty function is designed and applied to unfeasible solutions. Other methods maintain the feasibility of the individuals in the population by means of specialized operators or decoders. impose a restriction that any feasible solution is 'better' than any unfeasible solution, consider constraints one at the time in a particular linear order, repair unfeasible solutions, use multiobjective optimization techniques, are based on cultural algorithms, or rate solutions using a particular co-evolutionary model. We discuss these techniques in turn by addressing questions A – J from the previous section.

A. HOW SHOULD TWO FEASIBLE INDIVIDUALS BE COMPARED, E.G., 'C' AND 'J' FROM FIGURE 2? IN OTHER WORDS, HOW SHOULD THE FUNCTION $eval_f$ BE DESIGNED?

This is usually the easiest question: for most optimization problems, the evaluation function f for feasible solutions is given. This is the case for numerical optimization problems and for most operation research problems (knapsack problem, traveling salesman problem, etc.) However, for some problems the selection of evaluation function might be far from trivial. For example, in building an evolutionary system to control a mobile robot (Michalewicz and Xiao 1995) there is a need to evaluate robot's paths. It is unclear, whether path #1 or path #2 (Figure 3) should have better evaluation (taking into account their total distance,

clearance from obstacles, and smoothness).⁴ For such problems there is a need for some heuristic measures to be incorporated into the evaluation function.

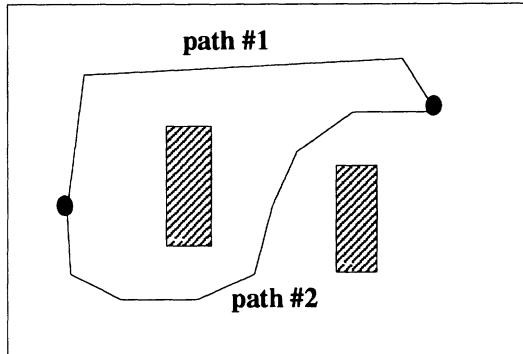


Figure 3: Paths in an environment

B. HOW SHOULD TWO UNFEASIBLE INDIVIDUALS BE COMPARED, E.G., ‘A’ AND ‘N’? IN OTHER WORDS, HOW SHOULD THE FUNCTION eval_u BE DESIGNED?

This is a quite hard question. We can avoid the problem altogether by rejecting unfeasible individuals (see part D). solution. Sometimes it is possible to extend the domain of function eval_f to handle unfeasible individuals, i.e., $\text{eval}_u(x) = \text{eval}_f(x) \pm Q(x)$, where $Q(x)$ represents either a penalty for unfeasible individual x , or a cost for repairing such individual (see part G). Another option is to design a separate evaluation function eval_u , independent of eval_f , however, in a such case we have to establish some relationship between these functions (see part C).

C. HOW ARE THE FUNCTIONS eval_f AND eval_u RELATED TO EACH OTHER? SHOULD WE ASSUME, FOR EXAMPLE, THAT $\text{eval}_f(s) \succ \text{eval}_u(r)$ FOR ANY $s \in \mathcal{F}$ AND ANY $r \in \mathcal{U}$?

Some researchers (Powell and Skolnick 1993, Michalewicz and Xiao 1995) report good results of their evolutionary algorithms, which work under the assumption that any feasible individual is better than any unfeasible one. However, this need not be the case. For example, it is unclear, whether feasible path #2 should be better than unfeasible path #1 (Figure 4).

⁴Note, that comparison between feasible and unfeasible paths (paths #1 and #2, Figure 4) is much harder.

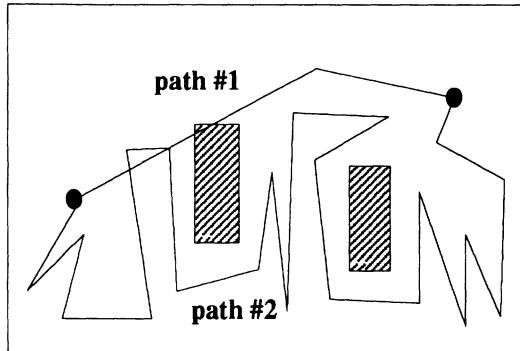


Figure 4: Paths in an environment

Some researchers prefer to define function $eval_u$ on the basis of $eval_f$:

$$eval_u(x) = eval_f(x) \pm Q(x),$$

where $Q(x)$ represents either a penalty for unfeasible individual x , or a cost for repairing such individual (see part G). However, these methods are problem dependent; often the selection of $Q(x)$ is as difficult as solving the original problem. Also, it may happen that for two individuals: feasible y and unfeasible x :

$$eval_u(x) = eval_f(x) \pm Q(x) > eval_f(y),$$

which may lead the algorithm to converge to unfeasible solution. This is the main reason for applying dynamic penalties Q (see part G) which increase pressure on unfeasible individuals with respect to the current state of the search.

D. SHOULD WE CONSIDER UNFEASIBLE INDIVIDUALS HARMFUL AND ELIMINATE THEM FROM THE POPULATION?

This “death penalty” heuristic is a popular option in many evolutionary techniques like evolution strategies or evolutionary programming. The method of eliminating unfeasible solutions from a population may work reasonably well when the feasible search space is convex and it constitutes a reasonable part of the whole search space (e.g., evolution strategies do not allow equality constraints since with such constraints the ratio between the sizes of feasible and unfeasible search spaces is zero). Otherwise such an approach has serious limitations. For example, for many problems where an initial population consists of unfeasible

individuals only, it might be essential to improve them (as oppose to ‘reject’ them). Moreover, quite often the system can reach the optimum solution easier if it is possible to “cross” an unfeasible region (especially in non-convex feasible search spaces).

E. SHOULD WE ‘REPAIR’ UNFEASIBLE SOLUTIONS BY MOVING THEM INTO THE CLOSEST POINT OF THE FEASIBLE SPACE (E.G., THE REPAIRED VERSION OF ‘M’ MIGHT BE THE OPTIMUM ‘X’, FIGURE 2)?

Repair algorithms enjoy a particular popularity in the evolutionary computation community: for many combinatorial optimization problems (e.g., traveling salesman problem, knapsack problem, set covering problem, etc.) it is relatively easy to ‘repair’ an unfeasible individual. Such repaired version can be used either for evaluation only, i.e.,

$$\text{eval}_u(x) = \text{eval}_f(y),$$

where y is a repaired (i.e., feasible) version of x , or it can also replace (with some probability) the original individual in the population (see part F).

The process of repairing unfeasible individual is related to combination of learning and evolution (so-called Baldwin effect, Whitley et al. 1994). Learning (local search in general, and local search for the closest feasible solution, in particular) and evolution interact with each other: the fitness value of the improvement is transferred to the individual. In that way a local search is analogous to learning that occurs during one generation of particular string.

The weakness of these methods is in their problem dependence. For each particular problem a specific repair algorithm should be design. Moreover, there are no standard heuristics on design of such algorithms: usually it is possible to use a greedy repair, random repair, or any other heuristic which would guide the repair process.

F. IF WE REPAIR UNFEASIBLE INDIVIDUALS, SHOULD WE REPLACE UNFEASIBLE INDIVIDUAL BY ITS REPAIRED VERSION IN THE POPULATION OR RATHER SHOULD WE USE A REPAIR PROCEDURE FOR EVALUATION PURPOSE ONLY?

The question of replacing repaired individuals is related to so-called Lamarckian evolution (Whitley et al. 1994), which assumes that an individual improves during its lifetime and that the resulting improvements are coded back into the chromosome. As stated in Whitley et al. 1994:

“Our analytical and empirical results indicate that Lamarckian strategies are often an extremely fast form of search. However, functions exist where both the simple genetic algorithm without learning and the Lamarckian strategy used [...] converge to local optima while the simple genetic algorithm exploiting the Baldwin effect converges to a global optimum.”

It is why it is necessary to use the replacement strategy very carefully. Recently (see Orvosh and Davis 1993) a so-called 5%-rule was reported: this heuristic rule states that in many combinatorial optimization problems, an evolutionary computation technique with a repair algorithm provides the best results when 5% of repaired individuals replace their unfeasible originals. However, many recent experiments (e.g., Michalewicz 1994) indicated that for many combinatorial optimization problems this rule did not apply. Currently a GA-based system is being developed for constrained numerical optimization, which is based on repairing unfeasible individuals. Once the system is completed, a replacement strategy will be carefully examined.

G. SINCE OUR AIM IS TO FIND A FEASIBLE OPTIMUM SOLUTION, SHOULD WE CHOOSE TO PENALIZE UNFEASIBLE INDIVIDUALS?

This is the most common approach in genetic algorithm community. The domain of function $eval_f$ is extended; the approach assumes that

$$eval_u(p) = eval_f(p) + penalty(p)?$$

The major question is, how should such a penalty function $penalty(p)$ be designed? The intuition is simple: the penalty should be kept as low as possible, just above the limit below which infeasible solutions are optimal (so-called *minimal penalty rule*). However, it is difficult to implement this rule effectively.

There were many approaches which are based on penalty functions. It is possible to measure a violation of a constraint and assign a penalty accordingly (larger violations result in larger penalties, see Homaifar et al. 1994). It is possible to adjust penalties in a dynamic way, taking into account the current state of the search or the generation number (Smith and Tate 1993, Michalewicz and Attia 1994). Bean and Hadj-Alouane (1992) experimented with adaptive penalties, where if constraints do not pose a problem, the search would continue with decreased penalties, otherwise the penalties would be increased. The presence of both feasible and unfeasible individuals in the set of best individuals in the last few generations means that the current value of penalty component is set

right. Le Riche et al. 1995 designed a (segregated) genetic algorithm which uses two values of penalty parameters instead of one; these two values aim at achieving a balance between heavy and moderate penalties by maintaining two subpopulations of individuals. Some additional heuristics on design of penalty functions were studied (Richardson et al. 1989).

Still the experimental evidence is not sufficient to make any conclusions on usefulness of penalty methods in evolutionary techniques, in particular for highly constrained problems.

H. SHOULD WE START WITH INITIAL POPULATION OF FEASIBLE INDIVIDUALS AND MAINTAIN THE FEASIBILITY OF OFFSPRING BY USING SPECIALIZED OPERATORS?

During the last decade several specialized systems were developed for particular optimization problems; these systems use a unique chromosomal data structures and specialized ‘genetic’ operators which alter their composition. Some of such systems were described in Davis (1991). Known examples of such systems include Genocop (Michalewicz and Janikow 1991) for optimizing numerical functions with linear constraints and Genetic-2N (Michalewicz et al. 1991) for nonlinear transportation problem. These systems are much more reliable than any other evolutionary techniques based on penalty approach (Michalewicz 1994).

I. SHOULD WE CHANGE THE TOPOLOGY OF THE SEARCH SPACE BY USING DECODERS?

Decoders offer an interesting option for all practitioners of evolutionary techniques. In these techniques a chromosome “gives instructions” on how to build a feasible solution. For example, a sequence of items for the knapsack problem can be interpreted as: “take an item if possible”—such interpretation would lead always to feasible solutions.

However, it is important to point out that several factors should be taken into account while using decoders. Each decoder imposes a relationship T between a feasible solution and decoded solution (see Figure 5).

It is important that several conditions are satisfied: (1) for each solution $s \in \mathcal{F}$ there is a decoded solution d , (2) each decoded solution d corresponds to a feasible solution s , and (3) all solutions in \mathcal{F} should be represented by the same number of decodings d . Additionally, it is reasonable to request that (4) the transformation T is computationally

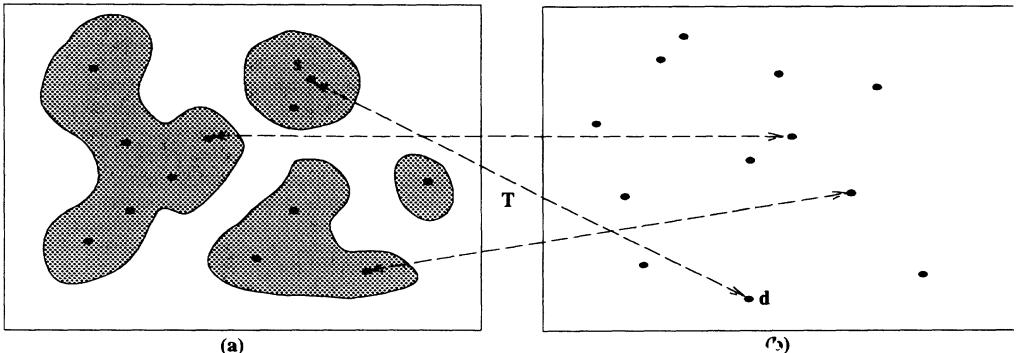


Figure 5: Transformation T between solutions in original (a) and decoder's (b) space

fast and (5) it has locality feature in the sense that small changes in the decoded solution result in small changes in the solution itself. An interesting study on coding trees in genetic algorithm was reported by Palmer and Kershenbaum (1994), where the above conditions were formulated.

J. SHOULD WE TAKE A SET OF CONSTRAINTS (WHICH DEFINE FEASIBLE SEARCH SPACE) AND PROCESS INDIVIDUALS AND CONSTRAINTS SEPARATELY?

This is an interesting heuristic. First possibility would include utilization of multi-objective optimization methods, where the objective function f and constraint violation measures f_j (for m constraints) constitute a $(m + 1)$ -dimensional vector \vec{v} :

$$\vec{v} = (f, f_1, \dots, f_m).$$

Using some multi-objective optimization method, we can attempt to minimize its components: an ideal solution x would have $f_j(x) = 0$ for $1 \leq i \leq m$ and $f(x) \leq f(y)$ for all feasible y (minimization problems).

Another approach, recently reported by Paredis 1994, is based on a co-evolutionary model, where a population of potential solutions co-evolves with a population of constraints: fitter solutions satisfy more constraints, whereas fitter constraints are violated by more solutions. Yet another heuristic is based on the idea of handling constraints in a particular order; Schoenauer and Xanthakis (1993) called this method a “behavioural memory” approach. This method requires some number of iterations; in each iteration the algorithm minimizes violation of

one constraints, rejecting individuals which violate constraints previously considered.

4. Conclusions

The paper surveys many heuristics which support the most important step of any evolutionary technique: evaluation of the population. It is clear that some further study in this area are necessary: different problems require different “treatment”. The author is not aware of any results which provide heuristics on relationships between categories of optimization problems and evaluation techniques in the presence of unfeasible individuals; this is an important area of future research.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant IRI-9322400.

5. References

- T. Bäck, F. Hoffmeister and H.-P. Schwefel, A Survey of Evolution Strategies. *Proceedings of the Fourth International Conference on Genetic Algorithms*, (Morgan Kaufmann Publishers, Los Altos, CA, 1991) 2–9.
- J.C. Bean and A.B. Hadj-Alouane, A Dual Genetic Algorithm for Bounded Integer Programs. Department of Industrial and Operations Engineering, The University of Michigan, TR 92-53 (1992).
- L. Davis, *Genetic Algorithms and Simulated Annealing*, (Morgan Kaufmann Publishers, Los Altos, CA, 1987).
- L. Davis, *Handbook of Genetic Algorithms*, (Van Nostrand Reinhold, New York, 1991).
- K.A. De Jong, An Analysis of the Behavior of a Class of Genetic Adaptive Systems. Doctoral dissertation, University of Michigan, *Dissertation Abstract International*, 36(10), 5140B.
- L.J. Eshelman and J.D. Schaffer, Real-Coded Genetic Algorithms and Interval Schemata. In *Foundations of Genetic Algorithms – 2*, ed. D. Whitley, (Morgan Kaufmann, Los Altos, CA, 1993) 187–202.
- D.B. Fogel and L.C. Stayton, On the Effectiveness of Crossover in Simulated Evolutionary Optimization. *BioSystems* 32 (1994) 171–182.
- L.J. Fogel, A.J. Owens and M.J. Walsh, *Artificial Intelligence through Simulated Evolution*, (Wiley, New York, 1966).

- D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, (Addison Wesley, Reading, MA, 1989).
- A.B. Hadj-Alouane and J.C. and Bean. A Genetic Algorithm for the Multiple-Choice Integer Program. Department of Industrial and Operations Engineering, The University of Michigan, TR 92-50 (1992).
- W. Hock and K. Schittkowski, *Test Examples for Nonlinear Programming Codes*, Lecture Notes in Economics and Mathematical Systems, Vol.187, (Springer-Verlag, New York, 1987).
- J.H. Holland, *Adaptation in Natural and Artificial Systems*, (University of Michigan Press, Ann Arbor, 1975).
- A. Homaifar, S. H.-Y. Lai and X. Qi, Constrained Optimization via Genetic Algorithms. *Simulation* 62 (1994) 242–254.
- J.A. Jones and C.R. Houck, On the Use of Non-Stationary Penalty Functions to Solve Nonlinear Constrained Optimization Problems With GAs. In *Proceedings of the Evolutionary Computation Conference—Poster Sessions*, part of the IEEE World Congress on Computational Intelligence, Orlando, 26–29 June 1994, 579–584.
- J.R. Koza, *Genetic Programming*, (MIT Press, Cambridge, MA, 1992).
- R. Le Riche, C. Vayssade, and R.T. Haftka, A Segragated Genetic Algorithm for Constrained Optimization in Structural Mechanics. Technical Report, Universite de Technologie de Compiegne, France (1995).
- Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, (Springer-Verlag, 2nd edition, New York, 1994).
- Z. Michalewicz and N. Attia, In Evolutionary Optimization of Constrained Problems. *Proceedings of the 3rd Annual Conference on Evolutionary Programming*, eds. A.V. Sebald and L.J. Fogel, (World Scientific Publishing, River Edge, NJ, 1994) 98–108.
- Z. Michalewicz and C. Janikow, Handling Constraints in Genetic Algorithms. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, (Morgan Kaufmann Publishers, Los Altos, CA, 1991) 151–157.
- Z. Michalewicz, T.D. Logan and S. Swaminathan, Evolutionary Operators for Continuous Convex Parameter Spaces. In *Proceedings of the 3rd Annual Conference on Evolutionary Programming*, eds. A.V. Sebald and L.J. Fogel, (World Scientific Publishing, River Edge, NJ, 1994) 84–97.
- Z. Michalewicz and J. Xiao, Evaluation of Paths in Evolutionary Planner/Navigator, In *Proceedings of the 1995 International Workshop*

- on Biologically Inspired Evolutionary Systems*, Tokyo, Japan, May 30–31, 1995, 45–52.
- D. Orvosh and L. Davis, Shall We Repair? Genetic Algorithms, Combinatorial Optimization, and Feasibility Constraints. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, (Morgan Kaufmann Publishers, Los Altos, CA, 1993) 650.
- J. Paredis, Co-evolutionary Constraint Satisfaction. In *Proceedings of the 3rd Conference on Parallel Problem Solving from Nature*, (Springer-Verlag, New York, 1994) 46–55,
- D. Powell and M.M. Skolnick, Using Genetic Algorithms in Engineering Design Optimization with Non-linear Constraints. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, Morgan Kaufmann Publishers, Los Altos, CA, 1993) 424–430.
- I. Rechenberg, *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*, (Frommann-Holzboog Verlag, Stuttgart, 1973).
- R.G. Reynolds, An Introduction to Cultural Algorithms. In *Proceedings of the Third Annual Conference on Evolutionary Programming*, (World Scientific, River Edge, NJ, 1994) 131–139.
- R.G. Reynolds, Z. Michalewicz and M. Cavaretta, Using Cultural Algorithms for Constraint Handling in Genocop. *Proceedings of the 4th Annual Conference on Evolutionary Programming*, San Diego, CA, March 1–3, 1995.
- J.T. Richardson, M.R. Palmer, G. Liepins and M. Hilliard, Some Guidelines for Genetic Algorithms with Penalty Functions. In *Proceedings of the Third International Conference on Genetic Algorithms*, (Morgan Kaufmann Publishers, Los Altos, CA, 1989) 191–197.
- J.D. Schaffer, Some Experiments in Machine Learning Using Vector Evaluated Genetic Algorithms. Doctoral dissertation, Vanderbilt University (1984).
- M. Schoenauer and S. Xanthakis, Constrained GA Optimization. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, (Morgan Kaufmann Publishers, Los Altos, CA, 1993) 573–580.
- H.-P. Schwefel, *Numerical Optimization for Computer Models*. (Wiley, Chichester, UK, 1981).
- W. Siedlecki and J. Sklanski, Constrained Genetic Optimization via Dynamic Reward–Penalty Balancing and Its Use in Pattern Recognition. In *Proceedings of the Third International Conference on Genetic*

- Algorithms*, (Morgan Kaufmann Publishers, Los Altos, CA, 1989) 141–150.
- A. Smith and D.M. Tate, Genetic Optimization Using a Penalty Function. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, (Morgan Kaufmann Publishers, Los Altos, CA, 1989) 499–505.
- N. Srinivas and K. Deb, Multiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. Department of Mechanical Engineering, Indian Institute of Technology, Kanput, India (1993).
- D. Whitley, V.S. Gordon, and K. Mathias, Lamarckian Evolution, the Baldwin Effect and function Optimization. In *Proceedings of the Parallel Problem Solving from Nature, 3*, (Springer-Verlag, New York, 1994), 6–15.
- A.H. Wright, Genetic Algorithms for Real Parameter Optimization. In *Foundations of Genetic Algorithms*, ed. G. Rawlins, First Workshop on the Foundations of Genetic Algorithms and Classifier Systems, Morgan Kaufmann Publishers, Los Altos, CA, 1991) 205–218.

Gene Pool Recombination in Genetic Algorithms

Heinz Mühlenbein
*GMD 53754 St. Augustin
 Germany
 muehlenbein@gmd.de*

Hans-Michael Voigt
*T.U. Berlin 13355 Berlin
 Germany
 voigt@fb10.tu-berlin.de*

Abstract:

A new recombination operator, called Gene Pool Recombination (*GPR*) is introduced. In GPR, the genes are randomly picked from the gene pool defined by the selected parents. The mathematical analysis of GPR is easier than for two-parent recombination (*TPR*) normally used in genetic algorithms. There are n difference equations for the marginal gene frequencies that describe the evolution of a population for a fitness function of size n . For simple fitness functions TPR and GPR perform similarly, with a slight advantage for GPR. Furthermore the mathematical analysis shows that a genetic algorithm with only selection and recombination is not a global optimization method, in contrast to popular belief.

Keywords: Difference equations, genetic algorithms, Hardy-Weinberg equilibrium, recombination.

1. Introduction

Genetic algorithms (GAs) use at least three different components for guiding the search to an optimum — selection, mutation and recombination. Understanding the evolution of genetic populations is still an important problem for biology and for scientific breeding. Mühlenbein and Schlierkamp-Voosen (1993, 1994) have introduced classical approaches from population genetics, the science of breeding, and statistics to analyze genetic algorithms. They describes the evolution of genetic populations as a dynamical system by difference or differential equations. Analyzing GAs that use both recombination and selection turns out to be especially difficult. The problem is that the mating of two genotypes creates a complex linkage between genes at different loci. This linkage is very hard to model and represents the major problem in population genetics (Naglyaki 1992).

For simple linear fitness functions we have found approximate solutions to the equations that describe the evolution of a genetic population through selection and recombination. Looking carefully at the assumptions leading to the approximation, we found that the equations obtained would be exact if a different recombination scheme were used. This recombination scheme we call *gene pool recombination (GPR)*. In GPR, for each locus the two alleles to be recombined are chosen *independently* from the gene pool defined by the selected parent population. The biologically inspired idea of restricting the recombination to the alleles of two parents for each offspring is abandoned. The latter recombination we will call *two-parent recombination (TPR)*.

The idea of using more than two parents for recombination is not new. Already Mühlenbein (1989) used eight parents; the offspring allele was obtained by a majority vote. Multi-parent recombination has also been investigated recently by Eiben, Raue and Ruttkay (1994) though their results are somewhat inconclusive. For binary functions the bit-based simulated crossover (BSC) of Syswerda (1993) is similar to GPR. However, his implementation merged selection and recombination. An implementation of BSC which separates selection and recombination was empirically investigated by Eshelman and Schaffer (1993). GPR is an extension of BSC, it can be used for any representation — discrete or continuous.

In this paper we will investigate TPR and GPR for discrete binary functions. It will be shown that GPR is easier to analyze than TPR. Furthermore, it converges faster. Nevertheless, in many cases TPR can be considered as an approximation to GPR.

2. Response to selection

In this section we summarize the theory presented in Mühlenbein and Schlierkamp-Voosen (1993, 1994). Let $\bar{f}(t)$ be the average fitness of the population at generation t . The response to selection is defined as

$$R(t) = \bar{f}(t+1) - \bar{f}(t). \quad (1)$$

The amount of selection is measured by the selection differential

$$S(t) = \bar{f}_s(t) - \bar{f}(t), \quad (2)$$

where $\bar{f}_s(t)$ is the average fitness of the selected parents. The equation for the response to selection relates R and S :

$$R(t) = b(t) \cdot S(t). \quad (3)$$

The value $b(t)$ is called the *realized heritability*. For many fitness functions and selection schemes, the selection differential can be expressed as a function of the standard deviation σ_p of the fitness of the population. For *truncation selection* (selecting the $T \cdot N$ best individuals) and for normally distributed fitness, the selection differential is proportional to the standard deviation (Falconer 1981):

$$\frac{S(t)}{\sigma_p(t)} = I.$$

The value I is called the *selection intensity*. For arbitrary distributions one can show the following estimate (Nagaraja 1982):

$$\frac{S(t)}{\sigma_p(t)} \leq \sqrt{\frac{1-T}{T}}.$$

For normally distributed fitness the famous equation for the response to selection is obtained (Falconer 1981):

$$R(t) = I \cdot b(t) \cdot \sigma_p(t). \quad (4)$$

The above equation is valid for a large range of distributions, not just for a normal distribution. The response depends on the selection intensity, the realized heritability, and the standard deviation of the fitness distribution. In order to use the above equation for prediction, one has to estimate $b(t)$ and $\sigma_p(t)$. The equation also gives a design goal

for genetic operators — to *maximize the product of heritability and standard deviation*. In other words, if two recombination operators have the same heritability, the operator creating an offspring population with larger standard deviation is to be preferred.

The equation defines also a design goal for a selection method — to *maximize the product of selection intensity and standard deviation*. In simpler terms, if two selection methods have the same selection intensity, the method giving the higher standard deviation of the selected parents is to be preferred. For proportionate selection as used by the simple genetic algorithm (Goldberg 1989) it was shown by Mühlenbein and Schlierkamp-Voosen (1993) that

$$\frac{S(t)}{\sigma_p(t)} = \frac{\sigma_p(t)}{\bar{f}(t)}.$$

The equation shows that the selection intensity of proportionate selection goes to zero for $t \rightarrow \inf$, whereas truncation selection has a constant selection intensity. Proportionate selection selects too weak in the final stages of the search.

Before showing that the evolution of a genetic population depends indeed on the selection intensity, the heritability and the standard deviation, we will derive exact difference equations for two and three loci directly.

3. Exact analysis of TPR for two loci

For simplicity we restrict the discussion to two loci and proportionate selection. In this case there are four possible genotypes: $(0, 0)$, $(0, 1)$, $(1, 0)$, and $(1, 1)$ which we index by $i = (0, 1, 2, 3)$. We denote their fitness values m_0 , m_1 , m_2 , and m_3 respectively. Let $q_i(t)$ be the frequency of genotype i at generation t . We assume an infinite population and *uniform crossover*. For proportionate selection the exact equations describing the evolution of the frequencies q_i can be derived easily. These equations — known for diploid chromosomes in population genetics (Crow and Kimura 1970) — assume an infinite population.

$$q_i(t+1) = \frac{m_i}{\bar{f}(t)} q_i(t) + \epsilon_i \frac{D(t)}{2\bar{f}(t)^2} \quad i = 0, 1, 2, 3 \quad (5)$$

with $\epsilon = (-1, 1, 1, -1)$. $\bar{f}(t) = \sum_{i=0}^3 m_i q_i(t)$ is the average fitness of the population. $D(t)$ defines the deviation from linkage equilibrium

$$D(t) = m_0 m_3 q_0(t) q_3(t) - m_1 m_2 q_1(t) q_2(t). \quad (6)$$

Note that $D(t) = 0$ if $q_0(t)q_3(t) = q_1(t)q_2(t)$ and $m_0 m_3 = m_1 m_2$. The first condition is fulfilled if the genotypes are binomially distributed. This assumption is called the *Hardy-Weinberg equilibrium* in population genetics. The general nonlinear difference equations have not yet been solved analytically (see the discussion by Naglyaki (1992)), but it is possible to derive an exact expression for the realized heritability. By summation we obtain

$$R(t) = \bar{f}(t+1) - \bar{f}(t) = \frac{V(t)}{\bar{f}(t)} - (m_0 + m_3 - m_1 - m_2) \frac{D(t)}{2\bar{f}(t)^2}, \quad (7)$$

where $V(t) = \sigma^2(t) = \sum q_i(t)(m_i - \bar{f}(t))^2$ denotes the variance of the population. Using $S(t) = V(t)/\bar{f}(t)$ we obtain the exact equation for the heritability,

$$b(t) = 1 - (m_0 - m_1 - m_2 + m_3) \frac{D(t)}{2\bar{f}(t)V(t)}. \quad (8)$$

In general, $b(t)$ depends on the genotype frequencies. Note that $b(t) = 1$ if $D(t) = 0$ or $m_0 + m_3 = m_1 + m_2$. The second assumption is fulfilled for the function ONEMAX(2) which has the fitness values $m_0 = 0, m_1 = m_2 = 1, m_3 = 2$. From (5) we obtain

$$\begin{aligned}\bar{f}(t+1) &= q_1(t+1) + q_2(t+1) + 2q_3(t+1) \\ &= \frac{q_1(t) + q_2(t) + 4q_3(t)}{\bar{f}(t)} = 1 + \frac{2q_3(t)}{\bar{f}(t)}.\end{aligned}$$

Let $p(t)$ denote the frequency of allele 1. Then by definition $\bar{f}(t) = 2p(t)$. Therefore we obtain

$$R(t) = 1 - p(t) + \frac{B_3(t)}{p(t)}, \quad (9)$$

where $B_3(t)$ denotes how far $q_3(t)$ deviates from the frequency given by the binomial distribution: $B_3(t) = q_3(t) - p^2(t)$.

The exact difference equation for $p(t)$ can be written as

$$p(t+1) = p(t) + \frac{1}{2} (1 - p(t)) + \frac{B_3(t)}{2p(t)}. \quad (10)$$

This equation has two unknown variables, $p(t)$ and $q_3(t)$. Therefore $p(t)$ cannot be directly computed. Selection leads the population away from the binomial distribution, and TPR is not able to recreate a binomial distribution for the offspring population.

We now discuss a function where $D(t) = 0$ if the population starts in a Hardy-Weinberg equilibrium. An example is MULT(2) with fitness values $m_0 = 1, m_1 = m_2 = 2, m_3 = 4$. In this case the difference equation for $p(t)$ is given by

$$p(t+1) = p(t) \frac{2}{1 + p(t)}, \quad (11)$$

which is solved easily.

In summary, even linear fitness functions lead to difficult systems of difference equations. The genetic population moves away from Hardy-Weinberg equilibrium. A class of multiplicative fitness functions with $m_0 m_3 = m_1 m_2$ leads to simpler equations, because the population stays in Hardy-Weinberg equilibrium.

4. Gene Pool Recombination

The exact analysis of recombination together with selection leads to difficult nonlinear differential equations. Recombination of two genotypes creates a linkage between the genes at different loci. This linkage is very hard to describe mathematically. Therefore we decided to look for a recombination operator that leads to simpler equations, like those we used as an approximation. This operator must maintain the population in a Hardy-Weinberg equilibrium. More general for n loci, it must create a multinomial distribution of the genotypes. Fortunately, there is a simple recombination scheme that fulfills this condition; we call it *gene pool recombination (GPR)*.

Definition: *In gene pool recombination the two “parent” alleles of an offspring are randomly chosen for each locus with replacement from the gene pool given by the parent population selected before. Then the offspring allele is computed using any of the standard recombination schemes for TPR.*

For binary functions GPR is obviously a Bernoulli process. Let $p_i^s(t)$ be the frequency of allele 1 at locus i in the **selected** parent population. Then GPR creates offspring with allele frequency $p_i(t+1) = p_i^s(t)$ and variance $p_i(t+1)(1-p_i(t+1))$ at locus i .

In order to analyze GPR we will derive difference equations for the gene frequencies, valid for arbitrary fitness functions and infinite populations. As before, we restrict the analysis to the case of two loci and proportionate selection.

Let $q_i(t)$ be the frequency of genotype i at generation t . For $n = 2$ loci, the marginal gene frequencies $p_1(t)$ and $p_2(t)$ can be obtained from

$$p_1(t) = q_2(t) + q_3(t) \quad p_2(t) = q_1(t) + q_3(t).$$

We assume that the initial population has a binomial distribution. This means that

$$\begin{aligned} q_0(0) &= (1 - p_1(0))(1 - p_2(0)) \\ q_1(0) &= (1 - p_1(0))p_2(0) \\ q_2(0) &= p_1(0)(1 - p_2(0)) \\ q_3(0) &= p_1(0)p_2(0). \end{aligned}$$

Then the following theorem holds:

Theorem 1 *Let the initial population have a binomial distribution. For an infinite population with GPR and proportionate selection, the marginal frequencies $p_1(t)$ and $p_2(t)$ can be obtained from*

$$p_1(t+1) = p_1(t) \frac{m_2(1 - p_2(t)) + m_3p_2(t)}{\bar{f}(t)} \quad (12)$$

$$p_2(t+1) = p_2(t) \frac{m_1(1 - p_1(t)) + m_3p_1(t)}{\bar{f}(t)}. \quad (13)$$

The realized heritability, $b(t)$, is given by

$$b(t) = 1 - (m_0m_3 - m_1m_2)(m_0 - m_1 - m_2 + m_3) \frac{p_1p_2(1 - p_1)(1 - p_2)}{V\bar{f}} \quad (14)$$

where p_1 , p_2 , V , and \bar{f} depend on t .

Proof: Proportionate selection selects the genotypes for the parents of population $t+1$ according to

$$q_i^s(t) = \frac{m_i}{\bar{f}(t)}q_i(t).$$

From q_i^s the marginal frequencies p_1^s and p_2^s can be obtained from the two equations $p_1^s(t) = q_2^s(t) + q_3^s(t)$ and $p_2^s(t) = q_1^s(t) + q_3^s(t)$. For each locus, GPR is a Bernoulli process; therefore, the marginal gene frequencies of parents and offspring remain constant

$$p_1(t+1) = p_1^s(t), \quad p_2(t+1) = p_2^s(t).$$

Combining these equations gives equations (12) and (13). The expression for the realized heritability can be obtained after some manipulations.

Remark: Theorem 1 can be extended to arbitrary functions of size n , or genetically speaking to n loci. This means that the evolution of an infinite genetic population with GPR and proportionate selection is fully described by n equations for the marginal gene frequencies. In contrast, for TPR one needs 2^n equations for the genotypic frequencies. For GPR one can in principle solve the difference equations for the marginal gene frequencies instead of running a genetic algorithm.

Note that $b(t) = 1$ if $m_0 m_3 = m_1 m_2$ or if $m_1 + m_2 = m_0 + m_3$. Let us first consider ONEMAX(2). The average fitness is given by $\bar{f}(t) = p_1(t) + p_2(t)$. If $p_1(0) = p_2(0)$, we have $p_1(t) = p_2(t) = p(t)$ for all t . From $\bar{f}(t) = 2p(t)$ we obtain

$$R(t) = 1 - p(t) \quad (15)$$

and

$$p(t+1) = p(t) + \frac{1}{2}(1 - p(t)). \quad (16)$$

This equation is similar to the equation obtained for TPR. It can be solved easily. Both equations become equal if $B_3(t) = 0$. This shows that for linear fitness functions, GPR and TPR give similar results — with a slight advantage for GPR, which converges faster.

Let us now turn to the function MULT(2). Combining $\bar{f}(t) = (1+p(t))^2$ with equation (12), we obtain equation (11). For MULT(2) TPR and GPR lead to the same difference equation. One can show that in general for multiplicative functions ($m_0 m_3 = m_1 m_2$) TPR and GPR are equal.

For many loci the above analysis can easily be extended to fitness functions which are called “unitation” functions. For these functions the fitness values depend only on the number of 1’s in the genotype. Again for simplicity we consider three loci only. Let u_i denote the fitness of a genotype with i 1’s. Under the assumption that $p_1(0) = p_2(0) = p_3(0)$, all marginal frequencies have the same value, which we denote as $p(t)$. Then we obtain for the marginal frequency $p(t+1) = c \cdot p(t)$

$$c = \frac{u_1(1-p)^2 + 2u_2p(1-p) + u_3p^2}{u_0(1-p)^3 + 3u_1p(1-p)^2 + 3u_2p^2(1-p) + u_3p^3}, \quad (17)$$

where $p = p(t)$. If $c > 1$ the marginal frequency $p(t)$ increases, if $c < 1$ it decreases. As a specific example we analyze a “deceptive” function of 3 loci (see Goldberg 1989). Let the fitness values of this function be $u_0 = 28, u_1 = 26, u_3 = 30$. The global optimum is at 111, the local optimum at 000. The fitness value for u_2 will be varied. Depending on u_2 and the initial population, the genetic population will converge to 000 or 111. In figure 1 c is shown for $u_2 = 25$ and $u_2 = 0$. For $u_2 = 0$, the area where the marginal frequency p is decreasing is larger than the area where the marginal frequency is increasing. This means that the population will converge to the second optimum if $p(0) = 0.5$. For $u_2 = 25$, the population will converge to 111 if $p(0) = 0.5$.

Remark: The analysis of unitation functions of three or more loci shows that a genetic algorithm using selection and recombination only is **not a global optimization** method. Depending on the frequency distribution of the genotypes and the fitness values, a genetic algorithm with infinite population size will deterministically converge to one of the local optima. The equations derived in this chapter can be used to determine the optima to which the genetic algorithm will converge.

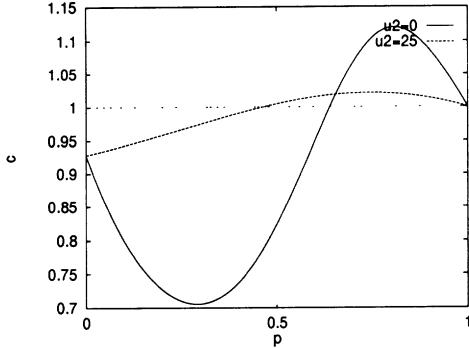


Figure 1: The line $c = 1$ divides the attractor regions 000 and 111.

As a last example we will analyze a fitness function where the results of TPR and GPR are very different. For simplicity we take two loci. The fitness values are defined as $m_0 = 0.99$, $m_1 = 0$, $m_2 = 0$, and $m_3 = 1$.

Table 1: Results for TPR and GPR for a bimodal function

t	REC	q_0	q_1	q_3	\bar{f}	Var
0	TPR	0.250	0.250	0.250	0.4975	0.2475
1	TPR	0.372	0.125	0.378	0.7463	0.1857
2	TPR	0.369	0.125	0.381	0.7463	0.1857
3	TPR	0.365	0.125	0.385	0.7464	0.1856
9	TPR	0.287	0.121	0.471	0.7556	0.1819
0	GPR	0.250	0.250	0.250	0.4975	0.2475
1	GPR	0.247	0.250	0.252	0.4975	0.2475
2	GPR	0.242	0.250	0.257	0.4977	0.2476
3	GPR	0.233	0.250	0.268	0.4983	0.2477
6	GPR	0.120	0.226	0.427	0.5457	0.2467
9	GPR	0.000	0.006	0.988	0.9883	0.0115

Table 1 shows that GPR very slowly changes the gene frequencies at the beginning. In fact, if $m_0 = 1$ the population would stay in equilibrium. After three generations GPR changes the gene frequencies very quickly. In contrast, TPR dramatically changes the frequencies in the first generation. The population immediately goes to the equilibrium points for the symmetric fitness function $m_0 = m_3 = 1$, $m_1 = m_2 = 0$. It takes TPR a long time to leave this equilibrium point and march to the optimum.

Proportionate selection should not be used for a genetic algorithm, its selection intensity is far too low (Mühlenbein and Schlierkamp-Voosen 1994). Unfortunately, for other selection methods the equations for the marginal gene frequencies are difficult to obtain. For truncation selection an approximate analysis can be done by using the equation for the response to selection. The following theorem was proven by Mühlenbein and Schlierkamp-Voosen (1993) in a different context.

Theorem 2 Let $p(t)$ be the frequency of allele 1 in the population at generation t . Let the fitness have a binomial distribution, i.e. $\sigma_p(t) = \sqrt{n \cdot p(t) \cdot (1 - p(t))}$.

If the population is large enough to converge to the optimum, and if the selection

intensity I is greater than 0, then the number of generations needed to reach equilibrium is approximately

$$GEN_e = \left(\frac{\pi}{2} - \arcsin(2p_0 - 1) \right) \cdot \frac{\sqrt{n}}{I}, \quad (18)$$

where $p_0 = p(0)$ denotes the probability of allele 1 in the initial population. The value $p(t)$ can be approximated as

$$p(t) \approx 0.5 \left(1 + \sin\left(\frac{I}{\sqrt{n}}t + \arcsin(2p_0 - 1)\right) \right). \quad (19)$$

Remark: The assumptions of the theorem are fulfilled for the ONEMAX(n) function. In this case the theorem is also approximately valid for TPR (Mühlenbein and Schlierkamp-Voosen 1994). For ONEMAX(n) GPR converges about 25% faster than TPR.

5. Genetic drift

The analysis of the previous sections is valid for very large populations. This leads to deterministic equations for averages. In finite populations the chance introduced by finite sampling has to be modelled. The mathematical analysis can be done in principle with Markov chains, but, unfortunately, the number of transitions scales exponentially with the number of loci and the size of the population. The analysis gets simpler if we assume no selection. This case is called *genetic drift*. The finiteness of the population causes convergence to a single genotype, even without selection. It is a result of sampling with replacement. Genetic drift is an important factor for genetic algorithms, because if the selection is too weak, then genetic drift is causing the “convergence” of the population.

Asoh and Mühlenbein (1994) have analyzed genetic drift for TPR. They showed that a genetic algorithm with TPR but without selection converges surprisingly quickly. This means that just by sampling with replacement, the variance of the population is continuously reduced. At the equilibrium the population consists of a single genotype only. For TPR the following proposition was obtained by fitting numerically obtained values:

Proposition 1 Let the number of loci be n . Let each gene have two alleles. Using TPR the mean convergence time $\tau_n(N)$ of a population of size N is approximately

$$\tau_n(N) \approx 1.4N (0.5 \ln(n) + 1.0) \quad \text{for } p(0) = 1/2. \quad (20)$$

Genetic drift for GPR is similar to a standard statistical process. Given n independent sampling processes with replacement, each process having a population N , what is the expected time for all n processes to converge, i.e. to consist of copies of one member only? This is a classical statistical question which can be answered for certain distributions. We state the following theorem without proof.

Theorem 3 Let the expected mean convergence time of the process be exponentially distributed with mean $\tau = 1/\lambda$. Then the expected mean time for all n processes to converge is given by

$$\tau_n = \frac{1}{\lambda} \sum_{\nu=1}^n \frac{1}{\nu}. \quad (21)$$

Table 2: Genetic drift for GPR (n loci, N popsize).

n/N	5	11	51	101	401
1	5.7	13.9	68.9	137.9	553.1
2	8.0	19.2	94.8	189.5	759.0
4	10.6	25.4	124.4	248.4	993.5
8	13.4	32.0	156.3	311.9	1247.0
16	16.4	38.9	189.5	378.0	1510.0

The sum can be approximated by $\ln(n) + \gamma$ with $\gamma = 0.577\dots$. By setting $1/\lambda = 1.4N$ equations 20 and 21 have the same asymptotic order. We have not been able to verify that the expected mean convergence time of GPR is exponentially distributed. Therefore we present numerical simulations obtained by Markov chains in table 2.

For $n = 1$ GPR and TPR are equal. For large N the mean time to convergence is approximately $\tau_1 = 1.39N$, as predicted by the proposition. The increase of τ with n is smaller than given by equation 21. A comparison of the numerical values for TPR presented by Asoh and Mühlenbein (1994) and table 2 shows that τ_n is only slightly larger for GPR than for TPR. This demonstrates that TPR and GPR give on the average very similar results, despite the fact that the underlying statistical processes are different.

Genetic drift is an important factor for genetic algorithms, especially in small populations. Whereas large populations converge deterministically to an equilibrium, can small populations converge to any of the many optima.

6. Conclusion

Gene pool recombination more closely resembles standard statistical process than does two-parent mating. For genetic algorithms GPR clearly separates the two stages of a search (which can be identified in any mathematical search algorithm): selecting promising areas to be searched and searching these areas in a reasonable manner. GPR searches within the selected area more reasonably than does TPR, which is more biased to the specific genotypes of the population.

GPR is mathematically more tractable than TPR. Nevertheless, the behavior is often surprisingly similar. In terms of population genetics: GPR keeps the population in a Hardy-Weinberg equilibrium, i.e., the genotypes always have a binomial distribution.

The equations obtained for GPR with infinite population clearly show that a genetic algorithm using only recombination is *not a global optimizer*. The genetic algorithm will not converge to a small isolated peak, even if it is the global optimum. Depending on the initial distribution of the genotypes, the population will deterministically converge to an area that has a large attractor region with many good fitness values. For a small population the genetic algorithm may converge to any good fitness value, just by chance.

The GPR idea can be easily extended to other problem domains. For continuous functions it directly leads to multi-parent search as described by Glover (1994). Variants of GPR have been successfully used for the traveling salesman problem. GPR liberates genetic algorithms from two-parent mating and all the difficult problems connected with it. Our theoretical analysis confirms the observation of Eshelman and Schaffer (1993): "Our hunch that the unique niche for pair-wise mating is much smaller than most GA researchers believe, and that the unique niche for two-point crossover is even smaller."

Acknowledgment: The theorem concerning genetic drift was proven by J. Bendisch. The simulations for the genetic drift were done by G. Paaß. This work is part of the SIFOGA project supported by the Real World Computing Partnership.

7. References

- Ash H. and Mühlenbein H., On the mean convergence time of evolutionary algorithms without selection and mutation, in: *Parallel Problem Solving from Nature*, eds. Davidor Y., Schwefel H.-P. and Männer R., Lecture Notes in Computer Science 866, (Springer-Verlag, 1994) pp. 88–97.
- Crow J. F. and Kimura M., *An Introduction to Population Genetics Theory*, (Harper and Row, New York, 1970).
- Eiben A., Raue P.-E. and Ruttkay Z., Genetic algorithms with multi-parent recombination, in: *Parallel Problem Solving from Nature*, eds. Davidor Y., Schwefel H.-P. and Männer R., Lecture Notes in Computer Science 866, (Springer-Verlag, 1994) pp. 78–87.
- Eshelman L. J. and Schaffer J. D., Crossover's niche, in: *Fifth Int. Conf. on Genetic Algorithms*, ed. Forrest S., (Morgan Kaufmann, San Mateo, 1993) pp. 9–14.
- Falconer D. S., *Introduction to Quantitative Genetics*, (Longman, London, 1981).
- Glover F., Genetic algorithms and scatter search: unsuspected potentials, *Statistics and Computing* 4(1994) 131–140.
- Goldberg D., *Genetic Algorithms in Search, Optimization and Machine Learning*, (Addison-Wesley, Reading, 1989).
- Mühlenbein H., Parallel genetic algorithm, population dynamics and combinatorial optimization, in: *3rd Int. Conf. on Genetic Algorithms*, ed. Schaffer H., (Morgan Kaufmann, San Mateo, 1989) pp. 416–421.
- Mühlenbein H. and Schlierkamp-Voosen D., Predictive Models for the Breeder Genetic Algorithm I. Continuous Parameter Optimization, *Evolutionary Computation* 1(1993) 25–49.
- Mühlenbein H. and Schlierkamp-Voosen D., The science of breeding and its application to the breeder genetic algorithm, *Evolutionary Computation* 1(1994) 335–360.
- Nagaraja H., Selection differentials, in: *Encyclopedia of Statistical Science*, eds. Kotz E., Johnson N. and Read C., (Wiley, New York, 1982) pp. 334–336.
- Nagylaki T., *Introduction to Theoretical Population Genetics*, (Springer, Berlin, 1992).
- Syswerda G., Simulated crossover in genetic algorithms, in: *Foundations of Genetic Algorithms 2*, ed. Whitley L. D., (Morgan Kaufmann, San Mateo, 1993) pp. 239–255.

Genetic and Local Search Algorithms as Robust and Simple Optimization Tools

Mutsunori Yagiura and Toshihide Ibaraki

Department of Applied Mathematics and Physics

Graduate School of Engineering

Kyoto University

Kyoto-city, 606-01 Japan

Email: *yagiura, ibaraki@kuamp.kyoto-u.ac.jp*

Abstract:

One of the attractive features of recent metaheuristics is in its robustness and simplicity. To investigate this direction, the single machine scheduling problem is solved by various genetic algorithms (GA) and random multi-start local search algorithms (MLS), using rather simple definitions of neighbors, mutations and crossovers. The results indicate that: (1) the performance of GA is not sensitive about crossovers if implemented with mutations, (2) simple implementation of MLS is usually competitive with (or even better than) GA, (3) GRASP type modification of MLS improves its performance to some extent, and (4) GA combined with local search is quite effective if longer computational time is allowed.

Key Words: combinatorial optimization, metaheuristics, genetic algorithm, local search, GRASP, single machine scheduling.

1. Introduction

There are numerous combinatorial optimization problems, for which computing exact optimal solutions is computationally intractable, e.g., those known as *NP-hard* [6]. However, in practice, we are usually satisfied with good solutions, and in this sense, *approximate*

mate (or *heuristic*) *algorithms* are very important. In addition to common heuristic methods such as *greedy method* and *local search* (abbreviated as LS), there is a recent trend of *metaheuristics* which combine such heuristic tools into more sophisticated frameworks. Those metaheuristics include *random multi-start local search* (abbreviated as MLS) [4][15], *simulated annealing*, *tabu search*, *genetic algorithm* (abbreviated as GA) [3][8][10] and so on.

One of the attractive features of these metaheuristics is in its simplicity and robustness. They can be developed even if deep mathematical properties of the problem domain are not at hand, and still can provide reasonably good solutions, much better than those obtainable by simple heuristics. We pursue this direction more carefully in this paper, by implementing various metaheuristics such as MLS and GA, and comparing their performance. The objective of this paper is not to propose the most powerful algorithm but to compare general tendencies of various algorithms. The emphasis is placed not to make each ingredient of such metaheuristics too sophisticated, and to avoid detailed tuning of the program parameters involved therein, so that practitioners can easily test the proposed framework to solve their problems of applications.

The LS starts from an initial solution x and repeats replacing x with a better solution in its *neighborhood* $N(x)$ until no better solution is found in $N(x)$. The MLS applies the LS to a number of randomly generated initial solutions and outputs the best solution found during the entire search. A variant of MLS called GRASP (greedy randomized adaptive search procedure) differs from MLS in that it generates initial solutions by randomized greedy methods. Some good computational results are reported in [4][5][14]. Simulated annealing and tabu search try to enhance the local search by allowing the replacement of the current solution x with a worse solution in $N(x)$ thereby avoiding to be trapped into bad local optima. The GA is a probabilistic algorithm that simulates the evolution process, by improving a set of candidate solutions as a result of repeating the operations such as crossover, mutation and selection. Among various modifications [3][20][23], it is reported in [1][13][16][22] that introducing local search technique in GA is quite effective. This is called *genetic local search* (abbreviated as GLS).

As a concrete problem to test, we solve in this paper the single machine scheduling problem (abbreviated as SMP), which is known to be NP-hard. It asks to determine an optimal sequence of n jobs that minimizes a cost function defined for jobs, e.g., the total weighted sum of earliness and tardiness. The efficiency of algorithms are measured on the basis of the number of solution samples evaluated, rather than the computational time, since the computational time depends on the computers used and other factors such as programming skill.

Our computational results indicate that GA performs rather poorly if mutation is not used, and is very sensitive to the crossover operations employed. However, if mutation is introduced, much better solutions are obtained almost independently of the crossover operators in use. Therefore, GA, using both crossover and mutation operators, is not a bad choice for practical purposes. However, a simple implementation of MLS appears better since it usually gives solutions of similar (or even better) quality with the same number of samples. Finally, it is also observed that GLS and GRASP tend to outperform GA and MLS in the sense that they produce solutions of better quality if sufficiently large number of samples are tested.

Our experiment does not cover other metaheuristics such as simulated annealing and tabu search, and it is a target of our future research to assess and compare such metaheuristics.

2. Single Machine Scheduling Problem

The single machine scheduling problem (SMP) asks to determine an optimal sequence of n jobs in $V = \{1, \dots, n\}$, which are processed on a single machine without idle time. A sequence $\sigma: \{1, \dots, n\} \rightarrow V$ is a one-to-one mapping such that $\sigma(k) = j$ means that j is the k -th job processed on the machine. Each job i becomes available at time 0, requires integer processing time p_i and incurs cost $g_i(c_i)$ if completed at time c_i , where $c_i = \sum_{j=1}^{\sigma^{-1}(i)} p_{\sigma(j)}$. A sequence σ is optimal if it minimizes

$$\text{cost}(\sigma) = \sum_{i \in V} g_i(c_i). \quad (1)$$

The SMP is known to be NP-hard for most of the interesting forms of $g_i(\cdot)$. We consider in particular

$$g_i(c_i) = h_i \max\{d_i - c_i, 0\} + w_i \max\{0, c_i - d_i\}, \quad (2)$$

where $d_i \in Z_+$ (set of nonnegative integers) is the due date of job i , and $h_i, w_i \in Z_+$ are respectively the weights given to earliness and tardiness of job i .

A number of exact algorithms, which are based on branch-and-bound, have been studied so far [19]. Another type of algorithm *SSDP* (*successive sublimation dynamic programming*) was proposed in [12] and it is observed that problem instances of up to $n = 35$ can be practically solved to optimality.

3. Metaheuristic Algorithms

3.1 Local Search and Multi-Start Local Search

The local search (LS) proceeds as follows, where $N(\sigma)$ denotes the neighborhood of σ .

LOCAL SEARCH (LS)

- 1 (initialize): Generate an initial solution σ by using some heuristics.
- 2 (improve): If there exists a solution $\sigma' \in N(\sigma)$ satisfying $\text{cost}(\sigma') < \text{cost}(\sigma)$, then set $\sigma := \sigma'$ and repeat step 2; otherwise output σ and stop.

A solution σ satisfying $\text{cost}(\sigma) \leq \text{cost}(\sigma')$ for all $\sigma' \in N(\sigma)$ is called a *local optimal solution*.

The LS may be enhanced by repeating it from a number of randomly generated initial solutions and outputs the best solution found during the entire search. This is called the *random multi-start local search* (MLS).

The performance of LS and MLS critically depends on: (1) the way of generating initial solutions, (2) the definition of $N(\sigma)$, and (3) the search strategy (i.e., how to search the solutions in $N(\sigma)$). In our experiment, we examine only the following strategies from the view point of simplicity.

- (1) Initial solutions: σ are randomly selected with equal probability.

(2) Neighborhoods: $N_{ins}(\sigma) = \{\sigma_{i \leftarrow j} \mid i \neq j\}$ and $N_{swap}(\sigma) = \{\sigma_{i \leftrightarrow j} \mid i \neq j\}$. Here $\sigma_{i \leftarrow j}$ is the sequence obtained from σ by moving the j -th job to the location before the i -th job, while $\sigma_{i \leftrightarrow j}$ is obtained by interchanging the i -th job and j -th job of σ .

(3) Search strategies: FIRST scans $N(\sigma)$ according to a prespecified random order and selects the first improved solution σ' satisfying $\text{cost}(\sigma') < \text{cost}(\sigma)$, and BEST selects the solution σ' having the best cost in the entire area of $N(\sigma)$.

Many other neighborhood structures are possible (e.g., [15] for the traveling salesman problem). Other metaheuristics such as simulated annealing and tabu search can be considered as modifications of MLS in which sophisticated search strategies for $N(\sigma)$ are used.

3.2 Greedy Randomized Adaptive Search Procedure

Greedy randomized adaptive search procedure (GRASP) is a variation of MLS [4][5][14], in which the initial solutions are generated by randomized greedy methods. In our experiment, the initial solutions are generated as follows. At each step, let

$$M = \{i \in V \mid i \text{ is not scheduled yet}\}.$$

Then a job $i \in M$ is chosen as the next job according to a criterion based on a local gain function (e.g., the i with the smallest d_i). There are two types of algorithms, depending upon whether a schedule is constructed forward or backward. A forward schedule starts with the job to be processed at time 0, and continues adding the next job to be processed until M becomes empty. A backward schedule is symmetrically defined from the last job to the first job. We will describe below the forward schedule only.

1. Set $M := V$ and $t := 0$.
2. Choose a candidate set $CA \subseteq M$ of a fixed number of jobs ($|CA|$ is a prespecified positive integer) in the decreasing order of the local gain $e(i, t)$, and then randomly choose $i \in CA$ as the next job. Let $M := M - \{i\}$ and $t := t + p_i$.
3. Repeat step 2 until $M = \emptyset$.

Here the local gain function $e(i, t)$ represents the heuristic used. We employed the following six functions (and hence 12 algorithms corresponding to forward and backward constructions).

The first three local gain functions are given by

$$e_1(i, t) = g_i(t + p_i + \bar{p})\delta_i(t) - g_i(t + p_i)(1 - \delta_i(t)) \quad (3)$$

$$e_2(i, t) = w_i\delta_i(t) - h_i(1 - \delta_i(t)) \quad (4)$$

$$e_3(i, t) = \frac{w_i}{p_i}\delta_i(t) - \frac{h_i}{p_i}(1 - \delta_i(t)), \quad (5)$$

where \bar{p} is the average processing time $\bar{p} = \sum_{i \in V} p_i/n$, and $\delta_i(t)$ indicates whether the due date d_i is urgent or not, i.e.,

$$\delta_i(t) = \begin{cases} 1, & t + \bar{p} + p_i \geq d_i, \\ 0, & \text{otherwise.} \end{cases} \quad (6)$$

To define $e_4(i, t)$, suppose that all jobs j in $M - \{i\}$ are sorted in nondecreasing order of d_j , i.e., $d_{j_1} \leq \dots \leq d_{j_k}$. Then

$$e_4(i, t) = -g_i(t + p_i) - \sum_l g_{j_l}(c_{j_l}), \quad (7)$$

where $c_{j_l} = t + p_i + \sum_{h=1}^l p_{j_h}$. This gives the sum of $g_j(c_j)$ of all jobs in M if i is scheduled next and the rest is then scheduled in nondecreasing order of d_j . Functions e_5 and e_6 are similar to e_4 but use nonincreasing order of w_j and nondecreasing order of h_j , respectively, for set $M - \{i\}$.

3.3 Genetic Algorithm

The genetic algorithm (GA) is a probabilistic algorithm that repeats the operations such as crossover, mutation and selection to the set of candidate solutions P . This algorithm can be viewed as a generalization of LS, in which the neighborhood is defined to be the set of solutions obtainable from P by crossover and mutation operators. We denote such neighborhood by $N_{CROSS-MUT}$ if crossover operator CROSS and mutation operator MUT are employed.

GENETIC ALGORITHM (GA)

- 1 (initialize): Construct a set of initial candidate solutions P .

- 2 (crossover and mutate): Generate a set of solutions $Q \subseteq N_{CROSS-MUT}(P)$.
- 3 (select): Select a set of $|P|$ solutions P' from $P \cup Q$, and set $P := P'$.
- 4 (iterate): Repeat steps 2 to 3 until some stopping criterion is satisfied.

Among various types of crossover, mutation and selection operators known so far [3][8][17][20][24], we consider representative operators, as described in the following Subsections.

3.3.1 Crossover

A crossover operator generates one or more solutions (*children*) by combining two or more candidate solutions (*parents*). Here, we assume that one child σ_C is produced from two parents σ_A and σ_B .

PMX (partially mapped crossover)[7]: An n -bit mask $m \in \{0, 1\}^n$ is generated randomly. Set $\sigma_C(i) := \sigma_A(i)$ for all i with $m(i) = 0$, and for each i with $m(i) = 0$ exchange $\sigma_B(i)$ with $\sigma_B(j)$ such that $\sigma_B(j) = \sigma_A(i)$. Then set $\sigma_C(i) := \sigma_B(i)$ for all i with $m(i) = 1$.

Crossover operators based on arbitrary masks are called uniform, and those based on restricted masks having at most k adjacent 0-1 pairs are called k -point; e.g., masks 11000 and 10011 are 1-point and 2-point respectively. Here we consider 1-point, 2-point and uniform PMX, which are respectively denoted PMX1, PMX2 and PMXU.

CX (cycle crossover)[17]: From sequences σ_A and σ_B , form a permutation $\binom{\sigma_A}{\sigma_B}$, and decompose it into cycles R_1, R_2, \dots, R_{k_0} , where R_k denotes the set of positions i in the k -th cycle. By definition, $\{\sigma_A(i) \mid i \in R_k\} = \{\sigma_B(i) \mid i \in R_k\}$ holds for all k . Then partition set $K = \{1, 2, \dots, k_0\}$ into K_A and K_B , and define σ_C by setting $\sigma_C(i) := \sigma_A(i)$ for all $i \in R_k$, if $k \in K_A$, and $\sigma_C(i) := \sigma_B(i)$ for all $i \in R_k$, if $k \in K_B$. Various strategies for partitioning K are possible. Here we consider (1) CXU: K is randomly partitioned into K_A and K_B , (2) CX1: $k \in K$ is randomly chosen, and $K_A := \{k\}$, $K_B := K - K_A$, and (3) CXA: $K_A := \{1, 3, 5, \dots\}$ and $K_B := \{2, 4, \dots\}$.

PsRND (position-based random crossover)[24]: Represent σ_A by a set of pairs $T_A = \{(i, \sigma_A(i)) \mid i = 1, \dots, n\}$; similarly for T_B and T_C . A child σ_C is constructed by starting from $T_C := \emptyset$ and by repeating the following as far as $T_A \cup T_B \neq \emptyset$ holds.

Randomly select an element $e \in T_A \cup T_B$ and set $T_C := T_C \cup \{e\}$, and remove the elements in $T_A \cup T_B$ which contradict with T_C .

If this halts before completing T_C by $T_A \cup T_B = \emptyset$, then complete T_C by adding the noncontradicting elements to T_C .

OX (order crossover)[2][16]: Generate randomly an n -bit mask $m \in \{0, 1\}^n$. Set $\sigma_C(i) := \sigma_A(i)$ for all i satisfying $m(i) = 0$. Define $D'_B = D_B - \{(i, j) \mid i \in S_m \text{ or } j \in S_m\}$, where $D_B = \{(i, j) \mid \sigma_B^{-1}(i) \leq \sigma_B^{-1}(j)\}$ and $S_m = \{\sigma_A(i) \mid m(i) = 0\}$. Then D'_B is the total order of σ_B restricted to $V - S_m$. Complete σ_C by assigning jobs to all the positions i satisfying $m(i) = 1$ according to D'_B . We call 1-point, 2-point and uniform crossover operators of this type as OX1, OX2 and OXU respectively.

Others: Many other crossover operators for sequencing problems have been proposed, e.g., free list crossover [9], partial order preserving crossover [24], alternating edges crossover [9], edge recombination crossover [20][23], matrix crossover [11], simulated crossover [1][21], and so on. Several of them were examined in [24] under a simple framework of GA without mutations.

3.3.2 Mutation

Mutation employed in this paper perturbs a candidate solution σ by repeating i times the random selection $\sigma' \in N(\sigma)$ and $\sigma := \sigma'$, where $i \in [1, k]$ is a positive integer randomly chosen, and k is a program parameter. Two types of neighborhood $N_{ins}(\sigma)$ and $N_{swap}(\sigma)$ are used as $N(\sigma)$; the resulting mutations are denoted as INS k and SWAP k respectively.

3.3.3 Generation of Candidate Solutions

Even after the crossover and mutation operators are fixed (i.e., $N_{CROSS-MUT}$ is defined), the set of generated candidate solutions $Q \subseteq N_{CROSS-MUT}(P)$ in step 2 of GA depends on other details. In

our experiment, we use $|Q| = 1$ and the child $\sigma_C \in Q$ is generated as follows.

1. Randomly select two parents $\sigma_A, \sigma_B \in P$.
2. After mutating either σ_A or σ_B by operator MUT, crossover them to produce σ_C by operator CROSS.

3.3.4 Selection

Selection in step 3 of GA chooses good candidate solutions from set $P \cup Q$ for the next generation. Though various types of selection strategies have been proposed [3][8], here we employ the following two simple strategies. The selection is executed only if the child $\sigma_C \in Q$ is not in P .

(1) BESTP: With a solution σ_{worst} satisfying $\text{cost}(\sigma_{worst}) \geq \text{cost}(\sigma)$ for all $\sigma \in P \cup Q$, let $P' := P \cup \{\sigma_C\} - \{\sigma_{worst}\}$.

(2) DELWP: Suppose $\text{cost}(\sigma_A) \leq \text{cost}(\sigma_B)$ without loss of generality, and let $P' := P \cup \{\sigma_C\} - \{\sigma_B\}$ with probability

$$p := \begin{cases} 1, & \text{if } \text{cost}(\sigma_C) \leq \text{cost}(\sigma_B) \\ \frac{\Delta_B}{\Delta_B + \Delta_C}, & \text{otherwise,} \end{cases}$$

where $\Delta_B := \text{cost}(\sigma_B) - \text{cost}(\sigma_A)$, $\Delta_C := \text{cost}(\sigma_C) - \text{cost}(\sigma_A)$; $P' := P$ with probability $1 - p$.

3.4 Genetic Local Search

Genetic local search (GLS) is a variation of GA [1][13][16][22], in which the new candidate solutions in Q are improved by LS.

GENETIC LOCAL SEARCH (GLS) (Steps 1 to 4 are the same as GA. Step 2' is added between steps 2 and 3 of GA.)

2' (improve): Improve each solution $\sigma \in Q$ by applying local search (LS).

GLS is different from MLS in that GLS generates the initial solutions for LS from the current P by crossover and/or mutation, while MLS generates them randomly from scratch.

While the improvement of a solution σ can be continued until a local optimal solution is obtained, cutting off the search at certain point is also possible to save computation time, since most of the

improvements usually occur in the early stage of LS. We examine the following simple termination rule: the number of samples for each LS is bounded above by $b|N(\sigma)|$ (b is a given positive integer, where $b = \infty$ means the original LS).

4. Computational Results

Computational experiments were performed on SUN SPARC station IPX using C language. The quality of the obtained solutions is evaluated by the average error from the best cost for the same problem instance found during the entire experiment.

4.1 Generation of Problem Instances

The tested problem instances are generated as follows. For $n = 35$ and 100, coefficients p_i, h_i, w_i for $i \in V (= \{1, \dots, n\})$ are generated by randomly selecting integers from interval [1, 10]. It has been observed in the literature (e.g., [18]) that problem hardness is related to two parameters RDD and LF , called the relative range of due dates and the average lateness factor, respectively. In our experiment,

$$\begin{aligned} RDD &= 0.2, 0.4, 0.6, 0.8, 1.0, \\ LF &= 0.2, 0.4, \end{aligned}$$

are used. Corresponding to each of these $5 \times 2 = 10$ cases, one problem instance is generated by selecting integer due dates d_i , $i \in V$, from interval

$$[(1 - LF - RDD/2)MS, (1 - LF + RDD/2)MS],$$

where $MS = \sum_{i \in V} p_i$. Sizes $n = 35$ and 100 are chosen, since $n = 35$ is the largest size that were solved to optimality by the SSDP method [12], and larger sizes should also be tested.

4.2 Random Multi-Start Local Search

Two types of neighborhoods N_{ins} and N_{swap} , and two types of search strategies FIRST and BEST of Section 3.1 are all examined. The average error (%) of the best solutions obtained by these four combinations are shown in Table 1, where 3×10^5 and 3×10^6 samples are generated for $n = 35$ and 100, respectively. Table 1 also shows

Table 1: Average error in % of the best solutions (average number of initial solutions) with MLS.

	$n = 35; 3 \times 10^5$ samples		$n = 100; 3 \times 10^6$ samples	
	N_{ins}	N_{swap}	N_{ins}	N_{swap}
FIRST	0.000 (97.6)	0.000 (103.6)	0.660 (103.6)	0.194 (104.6)
BEST	0.289 (8.6)	0.047 (13.9)	4.694 (3.2)	0.622 (5.0)

Table 2: Average error in % (standard deviation) of 100 local optima.

	$n = 35$		$n = 100$	
	N_{ins}	N_{swap}	N_{ins}	N_{swap}
FIRST	4.246 (4.670)	2.653 (4.078)	2.484 (2.332)	1.303 (1.491)
BEST	6.548 (7.473)	2.263 (3.582)	6.204 (5.475)	1.551 (1.699)

the average number of trials (i.e., the number of initial solutions) in parentheses. Note that even if a trial of LS is not completed at the time of termination, such a trial is also counted as one.

These results indicate that: (1) Search strategy FIRST obtains good solutions earlier than search strategy BEST. (Based on this, the search strategy is fixed to FIRST in the following experiments.) (2) The quality of solutions obtained by neighborhood N_{swap} is slightly better than that obtained by N_{ins} .

We also examined the average quality of local optimal solutions obtained by these local search algorithms. Table 2 shows the average error in % (standard deviation) of 100 (10 trials for each of 10 instances) local optima generated by each of the four combinations.

We observe that: (1) The differences of the solution quality among two search strategies are not critical. (2) The solution quality of N_{swap} local optima is better than that of N_{ins} on the average. (3) The combination of N_{ins} and BEST produces solutions of rather poor quality.

These results indicate that FIRST allows much more number of local optima being attained, where solution quality of each local optimum is comparable with that obtained by using BEST, within the same number of solution samples. This is the reason why FIRST

Table 3: Average error in % of the best solutions with GRASP using N_{swap} . The last column, init, indicates the average error of the initial greedy solutions generated by conventional greedy methods.

$ CA $	1	2	4	7	10	20	init
e_1 (f)	0.181	0.154	0.226	0.168	0.167	0.149	50.5
e_1 (b)	0.203	0.306	0.191	0.221	0.228	0.152	53.8
e_2 (f)	1.522	1.290	1.031	0.651	0.443	0.212	28.6
e_2 (b)	1.337	1.115	0.648	0.421	0.268	0.148	33.6
e_3 (f)	0.381	0.333	0.200	0.134	0.075	0.161	8.8
e_3 (b)	0.566	0.396	0.191	0.218	0.135	0.180	16.0
e_4 (f)	0.168	0.097	0.118	0.095	0.151	0.186	18.2
e_4 (b)	0.464	0.390	0.212	0.306	0.202	0.239	72.7
e_5 (f)	0.118	0.169	0.115	0.145	0.067	0.143	73.2
e_5 (b)	0.136	0.107	0.120	0.117	0.093	0.129	90.6
e_6 (f)	0.170	0.133	0.053	0.104	0.107	0.105	116.0
e_6 (b)	0.156	0.120	0.126	0.133	0.086	0.136	64.3
MLS				0.194			314.6

shows better performance in the computational results of Table 1.

4.3 Greedy Randomized Adaptive Search Procedure

A total of 12 local gain functions and parameter values $|CA| = 1, 2, 4, 7, 10, 20$ are tested to generate initial solutions of GRASP, where $|CA| = 1$ means the conventional greedy methods. In this Subsection, only the results with the neighborhood N_{swap} are shown; however, similar tendencies were observed for N_{ins} . Table 3 shows the average error (%) of the best solutions obtained by various GRASP algorithms, where e_i (f) (resp., e_i (b)) means the forward (resp., backward) construction with local gain function e_i . For the comparison purpose, the last row, MLS, indicates the average error when initial solutions are generated randomly. The last column, init, indicates the average error of the solutions generated by greedy methods, which means the quality of initial solutions used by GRASP when $|CA| = 1$.

The results indicate that: (1) Performance of GRASP critically depends on the local gain functions used for generating initial so-

lutions. (2) If the local gain function is properly chosen, GRASP improves the performance of MLS to some extent. However the performance is hardly affected by the parameter $|CA|$. (3) A local gain function which produces solutions of better quality does not always lead to better performance of GRASP.

In other words, GRASP is simple and can be powerful than MLS, but not robust with the local gain function used.

Table 4: Average error (%) of the best solutions with various versions of GA with $|P| = 100$.

		$n = 35; 3 \times 10^4$ samples			$n = 100; 3 \times 10^5$ samples		
		no MUT	INS1	SWAP1	no MUT	INS1	SWAP1
B	CXA	46.5	1.1	1.2	97.7	2.8	1.1
	CX1	46.4	1.8	2.4	96.2	2.3	0.8
	CXU	61.9	0.6	1.5	100.0	3.7	1.1
	PMX1	54.4	2.2	2.2	136.3	2.4	1.1
	PMX2	16.3	0.1	0.6	47.2	1.9	1.1
	PMXU	8.0	1.8	0.8	27.7	1.6	1.2
	PsRND	9.4	0.06	0.4	25.9	1.4	1.3
	OX1	110.1	5.1	3.5	190.9	3.1	2.1
	OX2	10.5	0.2	0.8	35.9	2.6	0.7
P	OXU	1.4	0.3	1.1	3.8	0.8	0.8
	MUT only	—	2.3 (0.7 [†])	4.0 (1.1 [†])	—	1.4 (1.2 [‡])	1.5 (0.7 [‡])
	CXA	48.2	1.3	0.8	83.4	3.3	0.8
	CX1	43.2	1.1	4.3	90.8	2.5	2.0
	CXU	40.8	1.6	2.1	91.5	4.0	0.7
	PMX1	57.5	1.3	1.3	136.1	2.1	1.5
	PMX2	21.2	1.0	0.7	57.1	1.4	1.6
	PMXU	4.8	0.6	1.0	22.5	1.8	0.8
	PsRND	4.9	0.3	1.4	20.3	1.2	1.1
W	OX1	107.9	4.2	2.6	184.3	3.1	1.8
	OX2	10.3	0.8	0.2	36.2	2.1	0.3
	OXU	0.9	0.07	0.1	2.4	1.1	0.5
	MUT only	—	3.3 (1.1 [†])	4.8 (0.4 [†])	—	1.4 (0.9 [‡])	1.9 (0.7 [‡])
	MLS	—	0.1	0.06	—	1.0	0.5

[†] results after 10^5 samples.

[‡] results after 10^6 samples.

4.4 Genetic Algorithm

Various neighborhoods $N_{CROSS-MUT}$ are compared under the framework of GA, in which $|P|$ is set to 100 and two selection strategies BESTP and DELWP are examined. Note that exactly one cost

evaluation occurs during one generation, since $|Q| = 1$ is used. Table 4 shows the average error (%) of the best solutions obtained by the tested algorithms, where 3×10^4 (3×10^5) samples are allowed for $n = 35$ ($n = 100$). Here, ‘MUT only’ means that crossover is not used and ‘no MUT’ means that mutation is not used. The results of MLS are also included for comparison, where the same neighborhood as mutation is used.

These results indicate that: (1) Using mutations is essential to get good solutions within the framework of GA. (2) Performance of GA is not sensitive against the types of crossover operators if combined with mutation, though it critically depends on the types of crossover operators if mutation is not used. (3) Selection strategy has little effect on the performance of GA. (4) Crossover is also effective to improve GA, since GA with MUT only needs slightly more samples than GA with crossover to obtain solutions of similar quality. (5) MLS performs better than GA.

We also tested GA when mutation rate k of Section 3.3.2 is set to greater than one. The results are omitted because of the limited space. It is observed that the performance of GA is hardly affected by parameter k if combined with crossover operators. However, the convergence of GA with MUT only becomes slower as k increases. It is also observed that $k = 1$ is sufficient for obtaining solutions of good quality.

Other population sizes $|P| = 10$ and 1000 are also tested so that we can compare the effect of increasing $|P|$ and that of incorporating mutation. The results are shown in Table 5. The search is terminated when no more improvement is considered to occur; to be concrete, 300 samples are tested for $|P| = 10$, and 2×10^5 (resp., 9×10^5) samples for $n = 35$ (resp., $n = 100$) are tested for $|P| = 1000$. Selection strategy BESTP is used, but mutation is not incorporated. Table 5 also include the results for $|P| = 100$ with mutation, where the same number of samples as Table 4 are tested. The solution quality is improved on the average if a larger $|P|$ is used. However, it critically depends on the crossover operator. In general, much more number of samples appears to be required to obtain good solutions by only increasing the population $|P|$, and mutation appears necessary to add.

Table 5: Average error (%) of the best solutions with GA using $|P| = 10, 100, 1000$.

$ P $	$n = 35$				$n = 100$			
	10		100		1000		10	
	noMUT	noMUT	SWAP1	noMUT	noMUT	noMUT	SWAP1	noMUT
CXA	237.8	46.5	1.2	2.5	262.9	97.7	1.1	10.6
CX1	248.7	46.4	2.4	2.8	271.7	96.2	0.8	13.8
CXU	221.4	61.9	1.5	3.5	250.3	100.0	1.1	12.5
PMX1	221.4	54.4	2.2	4.0	260.6	136.3	1.1	31.2
PMX2	228.0	16.3	0.6	0.4	242.6	47.2	1.1	9.1
PMXU	137.1	8.0	0.8	0.5	233.3	27.7	1.2	31.0
PsRND	165.1	9.4	0.4	0.2	190.1	25.9	1.3	31.4
OX1	251.1	110.1	3.5	16.7	292.8	190.9	2.1	94.3
OX2	222.3	10.5	0.8	0.3	274.7	35.9	0.7	4.6
OXU	173.6	1.4	1.1	0.03	208.3	3.8	0.8	0.5

Table 6: Average error in % of the best solutions with GA after 3×10^6 samples.

$ P $	10^1	10^2	10^3	10^4	MLS
INS1	4.874	0.639	0.593	0.522	0.660
SWAP1	0.802	0.800	0.536	0.325	0.194

GA with more tested samples is also tested. Table 6 shows the average error in % of the best solutions after 3×10^6 samples are generated, where $n = 100$, the selection strategy is BESTP, crossover type is OXU. For comparison purpose, the results of MLS are also included in Table 6, where the same neighborhood as the mutation is used.

The results indicate that: (1) Better solutions are obtained on the average as $|P|$ increases at the cost of testing more number of samples. (2) The quality of solutions obtained by MLS is still slightly better than those results of GA.

Note that much more computational time is needed to sample a solution with GA compared to other algorithms, such as MLS. We may conclude that the effectiveness of simple GA is in question.

Table 7: Average error (%) of the best solutions with GLS in which $|P| = 20$ and BESTP is used.

bound	$b = \infty$						$b = 1$	
neighbor	N_{ins}			N_{swap}			N_{ins}	N_{swap}
mutation	noMUT	INS1	SWAP1	noMUT	INS1	SWAP1	noMUT	noMUT
CXA	0.256	0.193	0.266	0.077	0.029	0.059	0.332	0.081
CX1	0.709	0.621	0.595	0.318	0.122	0.163	0.596	0.277
CXU	0.243	0.233	0.274	0.082	0.048	0.060	0.260	0.074
PMX1	0.286	0.315	0.335	0.092	0.062	0.081	0.327	0.157
PMX2	0.312	0.249	0.297	0.041	0.040	0.032	0.190	0.071
PMXU	0.264	0.271	0.341	0.028	0.023	0.046	0.205	0.028
PsRND	0.265	0.403	0.391	0.018	0.067	0.061	0.107	0.249
OX1	0.279	0.359	0.301	0.088	0.062	0.067	0.384	0.272
OX2	0.218	0.325	0.270	0.036	0.049	0.045	0.120	0.053
OXU	0.201	0.162	0.264	0.017	0.067	0.036	0.095	0.027
MUT only	—	0.596	0.508	—	0.100	0.054	—	—
MLS		0.660			0.194			

4.5 Genetic Local Search

It is observed by preliminary experiments that, within 3×10^5 (3×10^6) samples for $n = 35$ ($n = 100$), roughly 100 independent trials of LS starting from random initial solutions are possible using either N_{ins} or N_{swap} . $|P| = 20$ and selection strategy BESTP are used, and $b = \infty$ is assumed unless otherwise stated. By using these parameters, almost all the tested algorithms could obtain exact optimal solutions for all the instances of $n = 35$, and hence the results for $n = 35$ are omitted. Table 7 shows the average error (%) of the best solutions for $n = 100$. The results of GLS with $b = 1$, where no mutation is added, and the results of MLS are also included for comparison purposes.

In all cases, CX1 exhibited poor performance. One of the conceivable reasons of this phenomena is that CX1 can generate only children with limited variety. On the whole, we can summarize these results as follows. (1) GLS can obtain solutions of higher quality than MLS, provided that rather long computational time is allowed. (2) GLS is rather insensitive to the type of perturbations to generate new solutions, crossover and/or mutation. If only one of them is required, crossover appears slightly more effective than mutation. (3) The solution quality critically depends on the type

of neighborhood. (4) The convergence of GLS becomes faster by bounding the search length of LS (e.g., $b = 1$); however, the solution quality becomes slightly more sensitive to crossover than GLS with $b = \infty$.

5. Comparison and Conclusion

We conclude this paper by comparing four metaheuristic algorithms tested so far. Figures 1 and 2 show how the average error (%) of the best solutions improves as the number of samples increases. Both neighborhoods N_{ins} (Fig. 1) and N_{swap} (Fig. 2) are examined. In the case of GA, mutation operators INS1 and SWAP1 are used corresponding to N_{ins} and N_{swap} , respectively. For GRASP with neighborhood N_{ins} (resp., N_{swap}), local gain function e_6 (b) (resp., e_6 (f)) is used and parameter $|CA|$ is set to 7 (resp., 4). Selection strategy BESTP and crossover operator OXU are used for GA and GLS. The parameter $|P|$ is set to 1000 for GA and 20 for GLS, and mutation is not incorporated for GLS.

From these results, we can conclude that: (1) Performance of GA is robust about mutation; however, its performance is rather poor. (2) GRASP and GLS further improve the performance of MLS. GLS is more powerful than GRASP, if the same neighborhood is used. (Recall that the performance of GLS is quite robust about crossover, while the performance of GRASP is very sensitive to the local gain functions used to generate initial solutions.) (3) Performance of MLS, GRASP and GLS is affected by the type of neighborhood.

In view of these, we can summarize our recommendation as follows.

1. If the simplicity is our first concern, use MLS. In this case, the component to be defined in correspondence with the given problem is only the neighborhood.
2. If obtaining solutions of higher quality is important, use GLS. The simplest form of GLS is to introduce only mutations, in order to enhance MLS. As mutation can be realized by using the neighborhood of LS, the additional efforts required is very little, as far as the problem oriented efforts are concerned. It would also be worthwhile to introduce crossovers as well, since it may improve the performance further.

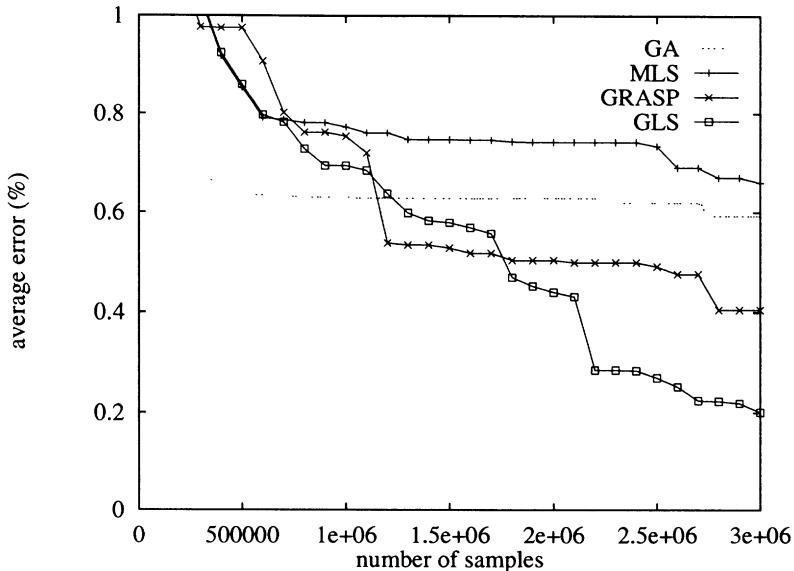


Figure 1: Average error (%) of the best solutions against the number of samples (neighborhood N_{ins} and mutation INS1 are used).

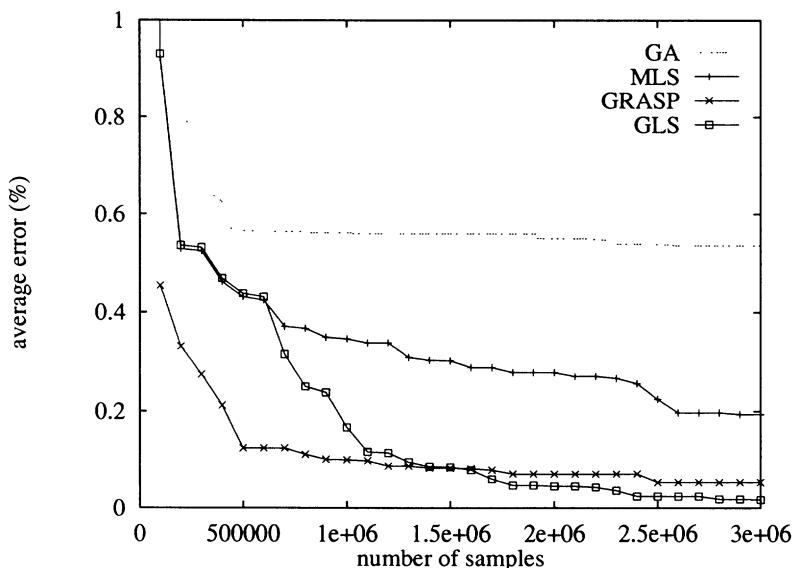


Figure 2: Average error (%) of the best solutions against the number of samples (neighborhood N_{swap} and mutation SWAP1 are used).

6. References

- [1] K.D. Boese, A.B. Kahng and S. Muddu, A New Adaptive Multi-Start Technique for Combinatorial Global Optimizations, *Operations Research Letters*, 16 (1994) 101.
- [2] L. Davis, Applying Adaptive Algorithms to Epistatic Domains, in: *Proceedings of the 9th IJCAI*, ed. A. Joshi (Morgan Kaufmann, California, 1985) p. 162.
- [3] L. Davis, *Handbook of Genetic Algorithms*, (Van Nostrand Reinhold, New York, 1991).
- [4] T.A. Feo, K. Venkatraman and J.F. Bard, A GRASP for a Difficult Single Machine Scheduling Problem, *Computers and Operations Research*, 18 (1991) 635.
- [5] T.A. Feo, M.G.C. Resende and S.H. Smith, A Greedy Randomized Adaptive Search Procedure for Maximum Independent Set, *Operations Research*, 42 (1994) 860.
- [6] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, (Freeman, New York, 1979).
- [7] D.E. Goldberg and R. Lingle, Alleles, Loci, and the Traveling Salesman Problem, in: *Proceedings of the 1st ICGA*, ed. J.J. Grefenstette (Lawrence Erlbaum Associates, New Jersey, 1985) p. 154.
- [8] D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, (Addison-Wesley, Massachusetts, 1989).
- [9] J. Grefenstette, R. Gopal, B. Rosmaita and D. Van Gucht, Genetic Algorithms for the Traveling Salesman Problem, in: *Proceedings of the 1st ICGA*, ed. J.J. Grefenstette (Lawrence Erlbaum Associates, New Jersey, 1985) p. 160.
- [10] J.H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, (MIT Press, Massachusetts, 1992).
- [11] A. Homaifar, S. Guan and G.E. Liepins, A New Approach on the Traveling Salesman Problem by Genetic Algorithms, in: *Proceedings of the 5th ICGA*, ed. S. Forrest (Morgan Kaufmann, California, 1993) p. 460.
- [12] T. Ibaraki and Y. Nakamura, A Dynamic Programming Method for Single Machine Scheduling, *European Journal of Operational Research*, 76 (1994) 72.
- [13] A. Kolen and E. Pesch, Genetic Local Search in Combinatorial Optimization, *Discrete Applied Mathematics*, 48 (1994) 273.
- [14] M. Laguna, T.A. Feo and H.C. Elrod, A Greedy Randomized Adap-

- tive Search Procedure for the Two-Partition Problem, *Operations Research*, 42 (1994) 677.
- [15] S. Lin and B.W. Kernighan, An Effective Heuristic Algorithm for the Traveling Salesman Problem, *Operations Research*, 21 (1973) 498.
 - [16] H. Mühlenbein, M. Gorges-Schleuter and O. Krämer, Evolution Algorithms in Combinatorial Optimization, *Parallel Computing*, 7 (1988) 65.
 - [17] I.M. Oliver, D.J. Smith and J.R.C. Holland, A Study of Permutation Crossover Operators on the Traveling Salesman Problem, in: *Proceedings of the 2nd ICGA*, ed. J.J. Grefenstette (Lawrence Erlbaum Associates, New Jersey, 1987) p. 224.
 - [18] C.N. Potts and L.N. Van Wassenhove, A Decomposition Algorithm for the Single Machine Total Tardiness Problem, *Operations Research Letters*, 1 (1982) 177.
 - [19] C.N. Potts and L.N. Van Wassenhove, A Branch and Bound Algorithm for the Total Weighted Tardiness Problem, *Operations Research*, 33 (1985) 363.
 - [20] T. Starkweather, S. McDaniel, K. Mathias, D. Whitley and C. Whitley, A Comparison of Genetic Sequencing Operators, in: *Proceedings of the 4th ICGA*, eds. R.K. Belew and L.B. Booker (Morgan Kaufmann, California, 1991) p. 69.
 - [21] G. Syswerda, Simulated Crossover in Genetic Algorithms, in: *Proceedings of the 2nd FOGA*, ed. L.D. Whitley (Morgan Kaufmann, California, 1993) p. 239.
 - [22] N.L.J. Ulder, E.H.L. Aarts, H.-J. Bandelt, P.J.M. Van Laarhouwen and E. Pesch, Genetic Local Search Algorithms for the Traveling Salesman Problem, in: *Proceedings of the 1st PPSN*, eds. H.-P. Schwefel and R. Männer (Springer-Verlag, Berlin, 1990) p. 109.
 - [23] D. Whitley, T. Starkweather and D. Fuquay, Scheduling Problems and Traveling Salesmen: The Genetic Edge Recombination Operator, in: *Proceedings of the 3rd ICGA*, ed. J.D. Schaffer (Morgan Kaufmann, California, 1989) p. 133.
 - [24] M. Yagiura and T. Ibaraki, On Genetic Crossover Operators for Sequencing Problems (in Japanese), *T. IEE Japan*, 114-C (1994) 713.

Comparison of Heuristic Algorithms for the Degree Constrained Minimum Spanning Tree

Geoff Craig[†], Mohan Krishnamoorthy[‡], M. Palaniswami[†]

*† Centre for Sensor Signal and Information Processing,
Department of Electrical and Electronic Engineering,
The University of Melbourne,
Parkville, VIC 3052, Australia.*

*‡ CSIRO Division of Mathematics and Statistics,
Private Bag 10, Rosebank MDC,
Clayton, VIC 3169, Australia.*

Contact Information: Mohan.Krishnamoorthy@dms.csiro.au

Abstract:

The degree constrained minimum spanning tree on a graph is the problem of generating a minimum spanning tree with constraints on the number of arcs that can be incident to vertices of the graph. In this paper we develop several heuristics, including simulated annealing, neural networks, greedy and greedy random algorithms for constructing such trees. We compare the computational performance of all of these approaches against the performance of an exact solution approach, using standard problems taken from the literature.

Key Words: degree constraints, greedy heuristics, minimum spanning tree, neural networks, simulated annealing

1 Introduction

The degree constrained minimum spanning tree (DCMST) is a minimum spanning tree (MST) on a graph, subject to constraints on the number of arcs incident at each vertex. A degree constrained

spanning tree (DCST) on a graph satisfies the degree constraints at each vertex, but need not be a minimal spanning of the vertices. If the degree constraint on each vertex is two, the DCMST almost reduces to the travelling salesman problem (TSP). If there are no degree constraints, the DCMST reduces to the MST.

Yamamoto(1978) finds the DCMST by finding the minimum common basis of two matroids. Narula and Ho (1980) formulate two greedy heuristics, and a branch and bound procedure based on Held and Karp's(1970, 1971) method for the TSP. Gavish(1982) uses subgradient optimisation for deriving Lagrangean based lower bounds. Savelsbergh and Volgenant(1985) use a branch and bound method for the DCMST based on an edge elimination approach that was applied to the TSP – see Volgenant and Jonker(1983). Volgenant(1989) describes a branch and bound procedure for the DCMST, based on Lagrangean relaxation and edge exchanges. A related problem to the DCMST, the Capacitated Minimum Spanning Tree (CMST), has received extensive treatment in the literature – see for example, Gavish(1982, 1983, 1985), Gavish and Hantler(1983) and Gouveia(1993).

Given an undirected complete graph $G = (V, A)$, $N = |V|$, with costs c_{ij} associated with the each arc $(i, j) \in A$, $c_{ii} = \infty$, a DCMST is a spanning tree of minimum total cost, such that the degree, d_i at each vertex $i \in V$ is at most a given value b_i . We define variables x_{ij} , where $x_{ij} = 1$ implies that an arc $(i, j) \in A$ is in the solution. We define an arc that is in the solution as a *branch*. All other arcs are denoted as *chords*. The DCMST can be formulated as a 0–1 integer programming problem – see Narula and Ho(1980).

By reducing it to an equivalent symmetric TSP, Garey and Johnson(1979) show that the DCMST is NP-hard. Hence, it is unlikely that there exists a polynomially bounded algorithm for the DCMST; so any exact solution approach will be inefficient as problem size increases. Hence the need for heuristic approaches to the DCMST. Heuristic methods like simulated annealing, tabu search, genetic algorithms, local search and greedy random search have been used extensively to solve complex optimisation problems – see

Reeves(1993). Recently, neural networks have also been successfully applied to solve a number of optimisation problems – see, for example, Hopfield and Tank(1985), Joppe, Cardon and Bioch(1990), Looi(1992). We provide results for practical-sized DCMST problems using traditional heuristics and a novel neural network method. We compare and evaluate the different approaches with the exact solution approach of Volgenant(1989).

2 Heuristics

We develop greedy and greedy random heuristics, a vertex clearing algorithm, a sequential unconstrained minimisation heuristic, simulated annealing, and a neural network approach for the DCMST.

2.1 Greedy Algorithms

The use of greedy algorithms for the DCMST is motivated by their speed and simplicity.

Kruskal Style Algorithm (GKRUS)

Kruskal's(1956) algorithm for the MST considers chords in increasing order of their cost. If the chord under consideration does not create a cycle in the the partial subgraph of the MST under construction, it is added to the subgraph. The algorithm stops when $(N - 1)$ branches have been added. To generate a DCMST, the procedure is performed as though generating an MST. However, if the addition of a chord causes a degree constraint to be violated, we ignore it and attempt to add the next chord.

Dijkstra Style Algorithm (GDIJK)

Dijkstra(1959) and Prim(1957) developed an efficient algorithm for finding the MST. Dijkstra's algorithm starts at any arbitrary vertex $i \in V$ and selects the least cost chord that connects i , thus generating a partial tree. At any iteration the algorithm adds to the current partial tree, the least cost chord that connects a new vertex to the partial tree. To generate a DCMST, we use the procedure suggested by Narula and Ho(1980). The procedure is performed as though generating an MST. However, if the addition of a chord to the partial tree, causes a degree constraint to be violated, we ignore

it and attempt to add the next chord. Unlike GKRUS, we note that different DCSTs can be obtained from GDIJK, depending on the starting vertex i . We therefore perform GDIJK N times, to get a wider sample of DCST solutions and use the least cost solution as the best DCST found.

2.2 Greedy Random Algorithms

GKRUS and GDIJK can be extended by introducing randomness at the point of chord selection. Our implementation is similar to that of Hart and Shogan(1987), who employ a cardinality-based semi-greedy random heuristic for the vehicle routing problem. We develop two algorithms, GRKRUS and GRDIJK, based on adding random elements to GKRUS and GDIJK respectively.

Greedy Random Kruskal and Dijkstra algorithms

Instead of adding the best chord at any stage of GKRUS or GDIJK, the chord that is added is chosen randomly from the best C available chords, see Hart and Shogan(1987). We tried three different forms for defining the parameter C . These were, $C = 2$, $C = \sqrt{N - n}$ and $C = \frac{N-n}{4}$, where n denotes the number of branches that have already been added. We only present results for the case $C = 2$, as the other approaches, and the values $C = 3$ and $C = 4$ did not produce as good solutions.

Vertex Clearing Algorithm (VCA)

The vertex clearing algorithm begins with the MST. If the MST violates any of the degree constraints, we choose randomly, one of the violated vertices. We remove all branches attached to this vertex and use the Greedy Kruskal style algorithm (as described above) to reconstruct a tree that satisfies degree constraints for that vertex. We repeat this process until all degree constraints are satisfied.

2.3 Simulated Annealing (SA)

Simulated annealing (SA) is recognised for its general applicability, needing only an initial solution, a definition of a neighbourhood and a cost function to be applied to a problem. This simplicity has led to the acceptance of SA as a solution method for combinatorial op-

timisation problems (COPs). See Collins, Eglese and Golden(1988) for a comprehensive bibliography of SA references and applications.

We considered two SA implementations: One which only considers feasible neighbourhood moves; another, which uses a penalty cost function to move through infeasible regions. The latter approach did not produce acceptable results and has been omitted. The former approach starts with an initial random tree which satisfies the degree constraints. The SA moves consist of *random branch-chord exchanges*. Such exchanges are performed by removing a random branch from the tree to form two *cutsets*. This branch is exchanged with a random chord that connects the two cutsets, such that the degree constraints are not violated.

The computation of our initial temperature θ_0 , was based on White(1984). The Markov chain length was fixed at $2N$. We use a reheating scheme for the cooling rate. Initially, the cooling rate, ρ is set at $\rho = 0.6$. We reheat when there is no change in the solution for $\lfloor \sqrt{N} \rfloor$ Markov chains. At each reheat, we set $\rho = \rho + \frac{(1-\rho)}{3}$; the corresponding temperature at this reheating point, θ_r , is reset according to the scheme used by Osman(1993): $\theta_r = \max(\theta_r/2, \theta_b)$, where θ_b is the temperature at which the best solution is found. We stop when six temperature reheats are completed.

2.4 Sequential Unconstrained Minimisation technique

The sequential unconstrained minimisation technique (SUMT) minimises a cost function which contains the actual cost together with a penalty cost, incorporating a penalty multiplier, for the constraints. Our implementation of the SUMT is adapted from the techniques described by Fiacco and McCormick(1968) and McCormick(1983).

Initially the problem is solved in its relaxed form; the penalty multiplier is zero. The constraints are slowly introduced to the problem, by incrementing the value of the multiplier. After each increment the problem is solved again, using the previous solution as the new starting solution. The SUMT is, therefore, analogous to a subgradient optimization approach to solving a lagrangean relaxation of constrained optimization problems.

For the DCMST, the SUMT cost function is the sum of the cost of the tree plus α times the sum of the squares of the degree constraint violations. Given a tree T , this function, denoted as $S(T)$, can be represented as:

$$S(T) = \sum_{i,j \in V} c_{ij}x_{ij} + \alpha \sum_{i \in V} (\max[0, d_i - b_i])^2 \quad (1)$$

where d_i is the degree of vertex i . At any given value of α , $S(T)$ is minimised via a *random descent search*. Initially α is set at a very small level, obtained by taking the minimum absolute change in cost in $2N$ random branch–chord exchanges, performed on a random spanning tree. We start with the MST and the initial α value and attempt to transform the MST into a DCST. At any iteration, we accept random branch–chord exchanges that result in an improvement of the SUMT cost function. After every $2N$ iterations, we increase the value of α to guide the search in the direction of feasible solutions. The process continues until a feasible tree is found, and no exchanges have been accepted for $6N$ consecutive iterations.

2.5 Neural Network

Artificial Neural Networks (ANNs) are models based on the functioning of the human brain – see Zurada(1992). They have been successfully used to solve a variety of practical problems in areas such as pattern recognition and optimisation – see Wilson and Sharda(1992). Motivations for the use of these methods include robustness, generalisation capabilities, and speed of operation through hardware implementability of inherent parallel structures. In the case of COPs, ANNs have been adapted to solve a range of problems. See Hopfield and Tank(1985), Nygard, Juell and Kaddaba(1990), Xu and Tsai(1991), Looi(1992), Lillo *et al*(1993), Potvin(1993), and Smith, Krishnamoorthy and Palaniswami(1994, 1995), for recent applications of ANNs to optimisation problems. The network we implemented contains two layers: a Hopfield style network – see Hopfield and Tank(1985), and a reachability network. For a more detailed description of our ANN implementation, refer to Craig, Krishnamoorthy and Palaniswami(1995a, 1995b).

2.5.1 Hopfield Style Network

The Hopfield style network contains $N(N - 1)/2$ neurons $x_{ij}(i < j)$, each representing an arc between vertices i and j . This network is used to minimise the total cost of the branches in the graph, subject to the constraints that each vertex is within its degree constraints and there are $N - 1$ vertices in the graph. These factors provide us with the following energy function to be minimised:

$$\begin{aligned} E(\bar{x}) = & \sum_{i=1}^{N-1} \sum_{j=i+1}^N x_{ij} c_{ij} + \lambda \sum_{i=1}^N (\zeta(i) - \sum_{j=i+1}^N x_{ij} - \sum_{j=1}^{i-1} x_{ji})^2 \quad (2) \\ & + \gamma \left(\sum_{i=1}^{N-1} \sum_{j=i+1}^N x_{ij} + 1 - N \right)^2 + \eta \sum_{i=1}^{N-1} \sum_{j=i+1}^N x_{ij} (1 - x_{ij}) \\ \text{where } \zeta(i) = & \max(1, \min(\sum_{j=i+1}^N x_{ij} + \sum_{j=1}^{i-1} x_{ji}, b_i)) \end{aligned}$$

By comparing the energy function for the problem and that of the standard Hopfield network, we obtain the connection strengths:

$$W_{kl,rs} = \eta \delta_{kr} \delta_{ls} - 2\gamma - 2\lambda(\delta_{kr} + \delta_{ls} + \delta_{lr} + \delta_{ks}) \quad (3)$$

$$\text{where } \delta_{ab} = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{otherwise} \end{cases}$$

and threshold values:

$$I_{kl} = 2\gamma(N - 1) + 2\lambda(\zeta(k) + \zeta(l)) - \frac{\eta}{2} - c_{kl} \quad (4)$$

2.5.2 Reachability Network

The second layer is a different type of recurrent network to the Hopfield Network, and is used to determine whether the graph is a spanning tree or not. It does this by determining the *reachability* of all the vertices from a random vertex. If every vertex is reachable, and the graph contains $N - 1$ branches, then we have a spanning tree. To determine reachability, we say that any neuron with a value greater than ϵ is in the graph. Given this and a random vertex, we check to see which vertices are in the same cutset as this vertex, this then enables us to determine whether we have a spanning tree. If we don't, the cutset is used to momentarily alter the threshold

values of all the neurons, by adding ΔI_{ij} , in such a way that the formation of branches between reachable and non reachable vertices is encouraged, $\Delta I_{ij} = \beta(c_{max} - c_{ij})$, while the formation of branches between unreachable vertices is discouraged, $\Delta I_{ij} = -\beta c_{ij}$. These encouragements and discouragements are made proportional to the costs of the arcs, so as to maintain some degree of optimality. This network is used at each iteration, after an initial settling period, to ensure that we end up with a spanning tree. For our experiments, we use the following values for the NN parameters: $\lambda = 2$, $\gamma = 2$, $\nu = 0.2$, $\epsilon = 0.1$ and $\beta = 180$. Also we use Euler's first order approximation with $\Delta t = 2.5E - 4$. Our sigmoid function is $g(x) = \frac{1}{2}(1 + \tanh(\frac{x}{\mu_0}))$ with gain width $\mu_0 = 0.009$.

3 Results and Discussion

We test our heuristics on a range of problems with $N=30$, 50, 70 and 100, taken from Volgenant(1989).

Table 1: Results for GKRUS, GDIJK, VCA, GRKRUS and GRDIJK on Symmetric problems.

Problem	GKRUS	GDIJK		VCA		GRKRUS		GRDIJK	
		min	avg	min	avg	min	avg	min	avg
s30-1	2.724	2.532	0.467	0.778	0.778	2.817	0.467	9.294	0.467
s30-2	4.729	4.739	1.754	2.365	2.365	6.972	3.738	10.618	1.754
s30-3	0.000	0.243	0.000	0.000	0.000	1.271	0.000	8.341	0.267
s30-4	21.022	19.901	0.000	19.071	17.770	24.216	17.770	21.984	0.000
s30-5	1.775	5.564	1.775	11.378	3.027	16.931	1.253	23.111	3.236
s30-6	3.672	3.952	0.000	3.371	2.919	8.098	3.578	15.665	4.896
s30-7	0.000	0.414	0.000	1.875	1.875	1.883	0.000	8.021	0.000
s30-8	0.000	0.679	0.000	0.000	0.000	8.299	0.000	11.136	0.000
s30-9	4.313	4.003	2.412	3.363	3.363	3.253	2.047	11.335	4.020
s30-10	1.087	10.945	1.087	0.988	0.988	5.138	0.000	17.220	6.126
s50-1	5.515	5.879	4.611	8.318	8.318	7.333	5.515	12.420	3.436
s50-2	1.028	1.209	0.000	1.028	1.028	1.823	1.028	3.767	0.891
s50-3	8.176	7.975	4.612	6.611	6.499	9.182	7.477	12.630	6.499
s50-4	7.469	7.167	2.201	0.314	0.314	9.002	1.887	10.866	2.358
s50-5	0.000	5.385	0.000	0.330	0.000	2.298	0.000	10.163	1.762
s50-6	0.000	2.891	0.226	0.000	0.000	2.393	0.000	12.573	0.376
s50-7	7.223	10.760	8.083	2.408	2.064	9.862	7.223	12.855	5.417
s50-8	0.390	0.359	0.000	0.390	0.390	5.269	0.390	8.198	0.390
s50-9	0.756	1.578	0.420	0.756	0.756	1.798	0.420	6.180	0.000
s50-10	5.969	5.907	1.990	2.102	1.557	8.010	2.076	14.443	2.422
s70-1	13.228	12.331	9.685	10.535	9.449	14.638	13.228	16.303	9.370
s70-2	1.929	5.218	2.160	0.849	0.849	3.580	1.929	9.592	3.164
s70-3	1.065	1.857	0.290	0.194	0.194	2.720	0.581	5.890	1.065
s70-4	8.003	7.801	4.533	3.399	3.399	10.496	5.170	10.790	1.841
s70-5	2.970	3.579	1.856	2.056	0.520	3.898	0.520	9.999	1.336
s70-6	2.634	3.348	2.634	1.013	1.013	3.222	2.634	7.332	0.000
s70-7	10.683	11.844	4.579	5.275	3.251	11.692	10.020	14.262	5.773
s70-8	2.883	2.368	0.076	0.455	0.455	3.672	2.731	5.724	0.303
s70-9	4.430	5.859	4.430	4.177	3.526	8.662	4.430	12.670	3.526
s70-10	19.764	20.766	14.803	15.685	14.252	21.756	20.079	24.821	16.220

The test data are divided into *euclidean* and *symmetric distance* problems. In the tables that follow, each problem is identified individually. For example, e50-9 is the ninth problem in the series

of ten euclidean problems of size $N = 50$. The neural network was run on a Cray Y-MP4E/464, all the other heuristics were tested on a SUN SPARC 10. For all our test problems, we consider a degree constraint of 3 ($b_i = 3, \forall i \in V$). We present the average and best solutions over a number of runs (ten in most cases, apart from GKRUS). The results are expressed as percentage relative deviations (GAP) from the exact solutions produced by Volgenant(1989).

Table 2: Results for GKRUS, GDIJK, VCA, GRKRUS and GRDIJK on Euclidean problems.

Problem	GKRUS	GDIJK		VCA		GRKRUS		GRDIJK	
		avg	min	avg	min	avg	min	avg	min
e30-1	0.000	0.000	0.000	0.000	0.000	1.753	0.000	11.963	4.486
e30-2	0.000	0.000	0.000	0.000	0.000	0.779	0.000	4.199	0.200
e30-3	0.000	0.000	0.000	0.000	0.000	1.625	0.311	13.654	1.554
e30-4	0.000	0.000	0.000	0.000	0.000	1.324	0.000	8.219	3.332
e30-5	0.449	0.520	0.449	0.449	0.449	1.155	0.000	9.553	2.404
e30-6	0.000	0.015	0.000	0.000	0.000	2.154	0.000	5.550	1.101
e30-7	0.000	0.000	0.000	0.000	0.000	0.653	0.000	8.266	0.993
e30-8	0.000	0.000	0.000	0.000	0.000	1.197	0.000	10.389	4.146
e30-9	0.000	0.000	0.000	0.000	0.000	1.003	0.027	11.927	5.898
e30-10	0.000	0.000	0.000	0.000	0.000	0.895	0.000	7.219	2.339
e50-1	0.000	0.000	0.000	0.000	0.000	0.515	0.000	6.594	2.985
e50-2	0.000	0.000	0.000	0.000	0.000	0.818	0.020	9.548	1.674
e50-3	0.000	0.000	0.000	0.000	0.000	0.687	0.000	9.547	3.896
e50-4	0.000	0.000	0.000	0.000	0.000	0.688	0.000	8.382	3.571
e50-5	0.061	0.076	0.061	0.061	0.061	0.602	0.061	5.644	0.817
e50-6	0.618	0.564	0.320	0.618	0.618	0.892	0.618	10.203	2.450
e50-7	0.000	0.000	0.000	0.000	0.000	0.862	0.000	10.141	4.054
e50-8	0.000	0.000	0.000	0.000	0.000	0.262	0.000	5.977	1.897
e50-9	0.000	0.000	0.000	0.000	0.000	0.834	0.000	9.302	3.104
e50-10	0.000	0.000	0.000	0.000	0.000	1.031	0.101	8.407	2.251
e70-1	0.000	0.089	0.000	0.000	0.000	0.263	0.000	8.756	2.479
e70-2	0.000	0.000	0.000	0.000	0.000	0.288	0.000	6.974	2.984
e70-3	0.204	0.125	0.000	0.204	0.204	0.786	0.051	6.893	2.533
e70-4	0.000	0.000	0.000	0.000	0.000	0.427	0.000	6.259	2.916
e70-5	0.307	0.303	0.000	0.307	0.307	0.657	0.000	7.010	2.955
e70-6	0.000	0.000	0.000	0.000	0.000	0.296	0.000	6.764	3.291
e70-7	0.000	0.000	0.000	0.000	0.000	0.385	0.000	6.162	2.139
e70-8	0.000	0.000	0.000	0.000	0.000	0.481	0.000	9.679	3.602
e70-9	0.000	0.000	0.000	0.000	0.000	0.525	0.000	7.754	2.545
e70-10	0.000	0.000	0.000	0.000	0.000	0.614	0.000	11.663	6.979
e100-1	0.000	0.000	0.000	0.000	0.000	0.321	0.000	6.617	2.556
e100-2	0.000	0.133	0.000	0.000	0.000	0.120	0.000	8.568	4.696
e100-3	0.000	0.004	0.000	0.000	0.000	0.137	0.000	5.362	2.171
e100-4	0.000	0.023	0.000	0.000	0.000	0.197	0.000	5.522	2.131
e100-5	0.155	0.014	0.000	0.155	0.155	0.406	0.155	8.290	4.430
e100-6	0.000	0.000	0.000	0.000	0.000	0.375	0.000	9.775	6.242
e100-7	0.000	0.000	0.000	0.000	0.000	0.093	0.000	6.049	2.077
e100-8	0.000	0.000	0.000	0.000	0.000	0.426	0.000	8.223	4.578
e100-9	0.076	0.081	0.076	0.076	0.076	0.525	0.091	8.247	2.613
e100-10	0.000	0.002	0.000	0.000	0.000	0.187	0.000	9.167	5.052

For the symmetric problems (Table 1), GKRUS, GDIJK, VCA, GRKRUS and GRDIJK produce average GAPs of 4.781%, 5.902%, 3.636%, 7.339% and 11.940%, and average minimums of 4.781%, 2.489%, 3.031%, 3.873% and 2.897% respectively. The quality of these bounds is, on average, unpredictable. The nature of these approaches poses a limitation in obtaining consistent quality solutions for the symmetric problems. For the euclidean problems (Table 2),

GKRUS, GDIJK, VCA, GRKRUS and GRDIJK produce average GAPs of 0.047%, 0.049%, 0.047%, 0.681%, and 8.210%, and average minimums of 0.047%, 0.023%, 0.047%, 0.036% and 3.053% respectively. Thus, we see that these greedy approaches produce, on average, reasonably tight upper bounds on the euclidean problems. The above approaches are extremely fast, usually taking less than 0.1s of cpu time. We observe that GRDIJK is not as efficient as the other greedy algorithms.

Table 3: Results for SA, SUMT and NN on Symmetric problems.

Problem	SA			SUMT			NN			
	avg	min	cpu	avg	min	cpu	avg	min	iters	con
s30-1	1.44	0.00	15.66	0.66	0.00	7.40	7.05	1.86	2050.2	9
s30-2	2.22	0.00	14.62	1.52	0.00	6.92	2.96	0.45	1885.7	9
s30-3	0.48	0.00	18.62	0.08	0.00	6.51	3.39	0.00	1797.2	10
s30-4	0.69	0.00	13.88	0.67	0.17	8.18	0.17	0.00	1246.2	10
s30-5	1.74	0.00	20.12	1.01	0.00	6.44	12.34	0.00	1875.0	4
s30-6	3.39	0.84	13.90	4.38	0.00	7.50	24.12	1.88	2001.0	5
s30-7	0.03	0.00	13.78	0.49	0.00	0.48	3.32	0.46	1751.1	10
s30-8	3.40	2.26	13.74	0.58	0.00	8.08	2.37	0.77	1803.4	9
s30-9	2.07	0.43	14.96	1.19	0.43	6.90	4.87	1.24	2008.2	6
s30-10	1.16	0.00	16.84	1.14	0.00	7.70	1.76	0.19	1793.6	8
s50-1	3.94	0.90	103.72	2.96	0.90	23.63	10.48	4.70	2371.8	5
s50-2	2.33	0.00	109.38	6.28	0.00	30.90	22.27	1.64	2468.8	6
s50-3	2.56	1.39	142.20	1.42	0.00	32.24	10.57	1.25	2204.4	8
s50-4	1.78	0.00	117.82	1.05	0.00	24.20	18.69	2.35	2105.0	5
s50-5	3.21	0.88	105.30	1.97	0.51	31.30	4.33	0.95	2185.8	5
s50-6	2.19	0.22	102.20	2.22	0.90	37.06	2.62	0.30	2251.6	6
s50-7	4.53	1.54	98.66	2.15	0.00	28.08	23.85	3.00	2059.8	8
s50-8	1.03	0.00	107.28	0.93	0.39	31.13	24.52	1.95	2129.2	6
s50-9	1.07	0.00	129.82	1.00	0.42	43.89	9.16	4.95	2194.5	5
s50-10	3.00	1.38	116.24	2.08	0.43	36.37	4.15	1.03	2150.4	5
s70-1	6.05	4.64	378.26	2.98	0.70	82.04	170.15	170.15	2891.0	1
s70-2	4.89	2.46	425.00	3.11	1.23	105.59	28.58	8.25	2621.0	2
s70-3	3.94	0.19	375.68	4.31	1.64	75.73	44.30	20.23	2817.0	3
s70-4	3.91	2.55	458.26	2.28	1.06	69.80	25.56	25.56	3000.0	1
s70-5	3.52	0.74	413.26	2.13	0.07	89.72	-	-	-	-
s70-6	4.29	2.63	435.52	2.82	0.60	82.26	8.10	8.00	2478.5	2
s70-7	5.05	1.26	376.66	2.88	1.26	95.65	-	-	-	-
s70-8	2.69	0.37	430.44	1.48	0.30	85.98	225.87	4.40	3285.3	3
s70-9	5.52	2.98	398.42	3.77	1.44	95.48	173.68	138.87	3730.0	2
s70-10	10.47	8.58	368.18	8.84	7.24	111.25	36.45	36.45	4500.0	1

The SA produces average GAPs of 1.897% and 3.09% and average minimums of 0.835% and 1.55% for the euclidean and symmetric problems respectively. Thus, while, on average, SA produces better results than the greedy and greedy random heuristics, this is at the cost of much larger computational effort.

The SUMT produces average GAPs of 0.058% and 2.284% and average minimums of 0.042% and 0.659% for the euclidean and symmetric problems respectively. The SUMT outperforms all other methods for solution quality. This is perhaps because we start with the MST as an initial solution, coupled with the fact that we use a general minimisation technique for minimising the overall cost.

The NN produces average GAPs of 8.601% and 9.65% and average minimums of 2.843% and 1.435% for the euclidean and symmetric problems respectively. These averages were only taken over problems for which the network converged in at least 25% of the attempts. The NN did not solve the larger euclidean problems ($N = 100$) and 3 other problems in the time that was allotted. With more emphasis on increasing the proportion of converged solutions, the GAPs can be improved. We observe that the network converges to a feasible solution in only 56% of the test problems. However, when it does converge to feasible solutions, the best minimum found is normally very good.

Table 4: Results for SA, SUMT and NN on Euclidean problems.

Problem	SA			SUMT			NN			
	avg	min	cpu	avg	min	cpu	avg	min	iters	con
e30-1	1.23	0.21	14.35	0.00	0.00	0.19	4.1	0.0	1635.8	10
e30-2	0.21	0.00	17.35	0.00	0.00	0.20	7.5	0.6	2127.7	5
e30-3	0.28	0.00	16.77	0.00	0.00	0.18	3.9	0.0	1354.8	8
e30-4	0.57	0.00	13.12	0.00	0.00	0.15	2.2	1.4	2479.3	10
e30-5	0.66	0.00	13.64	0.00	0.00	6.38	6.1	2.5	2208.9	10
e30-6	0.39	0.00	13.53	0.06	0.02	4.85	6.5	0.0	2031.8	6
e30-7	0.69	0.00	15.37	0.00	0.00	0.19	2.7	1.8	1899.7	8
e30-8	0.98	0.35	12.73	0.00	0.00	0.19	4.8	3.1	2414.4	10
e30-9	0.35	0.00	14.64	0.00	0.00	0.20	6.9	2.7	2198.0	10
e30-10	0.69	0.08	14.53	0.00	0.00	0.19	3.1	0.0	1500.2	10
e50-1	1.15	0.17	91.66	0.00	0.00	0.76	5.8	2.7	2968.7	8
e50-2	1.05	0.30	90.64	0.00	0.00	0.73	2.9	1.9	2606.1	4
e50-3	1.19	0.35	97.06	0.00	0.00	0.77	10.9	4.5	2935.2	7
e50-4	0.95	0.12	96.08	0.00	0.00	0.80	4.4	2.9	1864.4	6
e50-5	1.18	0.20	88.00	0.03	0.00	14.74	1.6	0.4	1978.7	8
e50-6	0.99	0.32	88.54	0.00	0.00	7.48	6.1	5.4	2358.3	3
e50-7	1.34	0.44	100.00	0.00	0.00	0.70	7.9	1.5	2243.7	9
e50-8	0.81	0.41	86.88	0.00	0.00	0.71	6.6	2.4	2809.5	4
e50-9	1.66	0.13	112.78	0.04	0.00	1.31	5.0	2.2	2564.7	2
e50-10	1.16	0.26	107.52	0.00	0.00	0.75	7.3	4.1	2645.6	4
e70-1	1.36	0.27	365.74	0.00	0.00	8.40	15.4	9.4	2857.0	2
e70-2	1.51	0.30	327.84	0.00	0.00	1.87	-	-	-	-
e70-3	1.25	0.71	365.03	0.00	0.00	62.81	13.0	3.9	3690.7	6
e70-4	1.85	0.35	337.30	0.00	0.00	1.94	18.7	2.3	4096.2	4
e70-5	2.81	1.00	325.42	0.37	0.30	105.48	15.5	4.3	3685.2	6
e70-6	1.64	0.59	359.03	0.00	0.00	1.90	9.1	2.8	3338.3	7
e70-7	1.55	0.56	351.60	0.00	0.00	1.82	25.0	12.0	2349.9	2
e70-8	1.39	0.39	338.57	0.00	0.00	1.88	16.0	2.1	3407.7	6
e70-9	2.20	0.89	316.84	0.00	0.00	1.91	13.9	2.6	2625.0	3
e70-10	2.20	0.55	339.89	0.00	0.00	1.89	15.1	1.6	4236.9	4
e100-1	2.86	1.19	1689.95	0.40	0.37	344.590	-	-	-	-
e100-2	4.45	3.51	1574.98	0.36	0.36	265.640	-	-	-	-
e100-3	3.33	1.67	1366.69	0.24	0.09	321.790	-	-	-	-
e100-4	3.88	1.85	1465.66	0.00	0.00	5.350	-	-	-	-
e100-5	4.22	1.96	1264.39	0.14	0.10	260.430	-	-	-	-
e100-6	5.24	2.96	1222.30	0.00	0.00	5.480	-	-	-	-
e100-7	3.87	3.03	1330.35	0.00	0.00	5.560	-	-	-	-
e100-8	4.17	3.32	1366.35	0.00	0.00	5.730	-	-	-	-
e100-9	3.97	2.53	1338.44	0.58	0.39	379.760	-	-	-	-
e100-10	4.43	2.29	1437.39	0.04	0.00	123.340	-	-	-	-

We do not report the cpu times directly for the neural network. Instead, we present the iterations and the times per iteration for each problem size. This is because the Neural Network is designed

to be implemented in FPGA hardware – see Botros and Abdul-Aziz(1994), where it would run as a truly parallel algorithm. For comparison the times per iteration are 0.11s, 0.403s and 1.015s for the 30, 50 and 70 node problems respectively. Given that an iteration of the hardware implementation of the neural network can be run in less than 0.1 milliseconds, almost regardless of the size of the problem, the neural net approach presents a very useful algorithm for solving large practical-sized DCMSTs.

4 Conclusions

In this paper, we have presented a wide range of heuristic algorithms for solving the degree constrained minimum spanning tree problem. We have also tested the performance of these algorithms on a wide range of problems and compared the computational performance with the performance of an exact solution approach.

The method recommended for finding good solutions to the DCMST problem depends, to a large extent, on the problem type. The greedy algorithms and greedy random algorithms tend to solve euclidean problems efficiently. However, SUMT performs consistently well on all problem classes and sizes.

5 Acknowledgements

The authors thank Ton Volgenant for kindly supplying codes and data, and Kate Smith for constructive discussions and comments.

6 References

- N. Botros and M. Abdul-Aziz, Hardware implementation of an artificial neural network using field programmable gate arrays (FPGA's), *IEEE Trans. on Industrial Electronics*, 41 (1994).
- N.E. Collins, R.W. Eglese, and B.L. Golden, Simulated annealing: An annotated bibliography, *American Journal of Mathematical and Management Sciences*, 8 (1988) 209.
- G. Craig, M. Krishnamoorthy and M. Palaniswami, The degree constrained minimum spanning tree: a comparison of heuristics, *Tech. Report DMS-C95/61, Division of Mathematics and Statistics, CSIRO (submitted to IEEE Trans. in Neural Networks)*, (1995).
- G. Craig, M. Krishnamoorthy and M. Palaniswami, A neural network approach to the degree constrained minimal spanning tree problem,

- Tech. Report DMS-C95/62, Division of Mathematics and Statistics, CSIRO*, (1995).
- E.W. Dijkstra, A note on two problems in connexion with graphs, *Numerische Mathematik*, 1 (1959) 269.
- A.V. Fiacco and G.P. McCormick, *Nonlinear programming techniques: sequential unconstrained minimisation techniques*, (Wiley, NY, 1968).
- M.R. Garey and D.S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*, (Freeman, San Francisco, 1979).
- B. Gavish, Topological design of centralized computer networks - formulations and algorithms, *Networks*, 12 (1982) 355.
- B. Gavish, Formulations and algorithms for the capacitated minimal directed tree problem, *Journal of the Association for Comp. Machinery*, 30 (1983) 118.
- B. Gavish and S. L. Hantler, Sidney, An algorithm for optimal route selection in sna networks, *IEEE Trans. on Comms.*, 31 (1983) 1154.
- B. Gavish, Augmented lagrangean based algorithms for centralized network design, *IEEE Trans. on Comms.*, 33 (1985) 1247.
- L. Gouveia, A comparison of directed formulations for the capacitated minimal spanning tree problem, *Telecom. Systems*, 1 (1993) 51.
- J.P. Hart and A.W. Shogun, Semi-greedy heuristics: an empirical study, *Operations Research Letters*, 6 (1987) 107.
- M. Held and R.M. Karp, The travelling-salesman problem and minimum spanning trees, *Operations Research*, 18 (1970) 1138.
- M. Held and R.M. Karp, The travelling-salesman problem and minimum spanning trees: Part II, *Mathematical Programming*, 1 (1971) 6.
- J.J. Hopfield and D.W. Tank, "neural" computation of decisions in optimization problems, *Biological Cybernetics*, 52 (1985) 141.
- A. Joppe, H.R.A Cardon, and J.C. Bioch, A neural network for solving the travelling salesman problem on the basis of city adjacency in the tour, *Intl. Joint Conference on Neural Networks*, 3 (1990) 961.
- J. Kruskal, On the shortest spanning subtree of a graph and the travelling-salesman problem, *Proceedings of the American Mathematical Society*, 7 (1956) 48.
- W.E. Lillo, M.H. Loh, S. Hui, and S.H. Zak, On solving constrained optimisation problems with neural networks: A penalty method approach, *IEEE Trans. on Neural Networks*, 4 (1993) 931.
- C. Looi, Neural network methods in combinatorial optimization, *Computers and Operations Research*, 19 (1992) 191.
- G.P. McCormick, *Nonlinear programming*, (Wiley, NY, 1983).

- S.C. Narula and C.A. Ho, Degree constrained minimum spanning tree, *Computers and operations research*, 7 (1980) 239.
- K. Nygard, P. Juell, and N. Kaddaba, Neural networks for selecting vehicle routing heuristics, *ORSA Journal on Comp.*, 2 (1990) 353.
- I.H. Osman, Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem, *Annals of Operations Research*, 41 (1993) 421.
- J. Potvin, The travelling salesman problem: A neural network perspective, *ORSA Journal on Comp.*, 5 (1993) 328.
- R.C. Prim, Shortest connection networks and some generalizations, *Bell System Technical Journal*, 36 (1957) 1389.
- C.R. Reeves, editor, *Modern heuristic techniques for combinatorial problems*, (Halsted Press, NY, 1993).
- M. Savelsbergh and T. Volgenant, Edge exchanges in the degree-constrained minimum spanning tree problem, *Computers and Operations Research*, 12 (1985) 341.
- K. Smith, M. Krishnamoorthy, and M. Palaniswami, Traditional heuristic versus hopfield network approaches to a car sequencing problem, *Tech. Report DMS-C94/64, Division of Mathematics and Statistics, CSIRO (submitted to European Journal of Operational Res.)*, (1994).
- K. Smith, M. Krishnamoorthy, and M. Palaniswami, A hybrid neural approach to combinatorial optimization problems, *Tech. Report DMS-C95/23, Division of Mathematics and Statistics, CSIRO (submitted to Computers and Operations Research)*, (1995).
- A. Volgenant, A langrangean approach to the dcmst problem, *European Journal of Operational Res.*, 39 (1989) 325.
- A. Volgenant and R. Jonker, The symmetric travelling salesman problem and edge exchanges in minimum 1-trees, *European journal of Operational Res.*, 12 (1983) 394.
- S.R. White, Concepts of scale in simulated annealing, in: *Proc IEEE Int. Conference on Computer Design*, (1989) p. 646.
- R. Wilson and R. Sharda, Neural networks, *ORMS Today*, 19 (1992) 36.
- X. Xu and W.T. Tsai, Effective neural algorithms for the travelling salesman problem, *Neural Networks*, 4 (1991) 193.
- Y. Yamamoto, The held-karp algorithm and degree-constrained minimum 1-trees, *Mathematical Programming*, 15 (1978) 228.
- J. M. Zurada, *Introduction to artificial neural systems*, (St. Paul, West, 1992).

An Aggressive Search Procedure for the Bipartite Drawing Problem

Rafael Martí

*Departamento de Estadística e I. O.
Facultad de Matemáticas. Universidad de Valencia
Dr. Moliner 50. 46100 Burjassot (Valencia). Spain
E-mail: Marti@mac.uv.es*

Abstract:

Graphs are used to represent reality in several areas of knowledge. This has generated considerable interest in graph drawing algorithms. Arc crossing minimization is a fundamental aesthetic criterion to obtain a readable map of a graph. The problem of minimizing the number of arc crossings in a bipartite graph (BDP) is NP-complete. In this paper we present an aggressive search scheme for the BDP based on the Intensification, Diversification and Strategic Oscillation elements of Tabu Search. Several algorithms can be obtained with this scheme by implementing different evaluators in the move definitions. In this paper we propose two variants. Computational results are reported on a set of 300 randomly generated test problems. The two algorithms have been compared with the best heuristics published in the literature and, for limited-size instances, with the optimal solutions.

Key Words: Arc crossing minimization, graph drawing problem, neighbourhood search.

1. Introduction

Graphs are used to represent reality in several areas of knowledge. They are a very useful aid in human thinking processes. Drawings of graphs are called maps and their effectiveness is widely recognised in various fields of the economic, social and computational sciences.

The main quality desired for maps is readability: The information of the model or the system represented by the graph is easily captured by the way it is drawn. It is extremely difficult to make a clear readable map of a graph representing real systems by hand, even when the graph size is relatively small. Therefore, tools for automatic graph drawing are indispensable.

Researchers in the graph-drawing field have proposed several aesthetic criteria that try to model what a "good" map of a graph is. Although readability may depend on the problem and the map's users, the authors consider that crossing reduction is a fundamental aesthetic criterion in graph drawing. For a survey on graph drawing we refer the reader to the annotated bibliography by Di Battista *et al.* (1993).

The problem of minimizing the number of arc crossings in a bipartite graph is referred to as the Bipartite Drawing Problem (BDP). BDP is known to be NP-complete (Johnson, 1982) even if the positions of the vertices of one of the partitions remain fixed (Eades, McKay and Wormald, 1985).

Several heuristic algorithms have been proposed for BDP. These algorithms belong to the class of approaches known as simple descent heuristics. In general these heuristics progress from an initial solution along a path in the solution space that changes the number of crossings in a uniformly descending direction until no further improvement is possible. At the stopping point the solution obtained is a local optimum which, considering that this is a combinatorial problem, very rarely is also global.

To avoid getting trapped in a local optimum and to perform an aggressive search for the global optimum, we propose a new class of algorithms. Using the *Intensification*, *Diversification* and *Strategic Oscillation* elements of Tabu Search (Glover, 1989, 1990), we present a neighbourhood search scheme for the BDP. Although several algorithms can be obtained by implementing different move evaluators, we restrict our attention to two.

The remainder of this paper is organised as follows. Section 2 contains relevant definitions and properties. In Section 3 we review the

algorithms and results reported in the literature. In Section 4 we present the search scheme for the BDP.

Section 5 contains the experimental results of our study. Considering the average and the median functions in the definition of the moves, two algorithms are obtained with the scheme of Section 3. Tests have been carried out on a collection of 300 randomly generated instances. For comparison purposes, we consider the 4 best simple descent algorithms presented in the literature and the Tabu Thresholding algorithm of Valls, Martí and Lino (1994). We also have considered the optimal solutions for test problems with relatively small number of vertices. The final section contains some concluding remarks.

2. Preliminaries

A bipartite graph $G=(V,E)$ is a simple directed graph where the set of vertices V is partitioned into subsets L and R and where $E \subseteq L \times R$. Note that the direction of the arcs has no effect on crossings, so throughout this paper we consider G to be an undirected graph denoted by (L,R,E) and the arcs to be edges.

A bipartite graph is drawn so that the vertices in L and R are both on straight vertical lines (called levels or layers) and the edges are straight lines. Under this convention, the problem of minimizing the number of edge crossings becomes the problem of finding the optimal ordering of the vertices in L (left layer) and R (right layer).

For each vertex $u \in L$ the set of its adjacent vertices (neighbours) is denoted by $N_u = \{ v \in R / (u,v) \in E \}$ and its degree by d_u . Similarly for a vertex $u \in R$.

An ordering h of the set L is a labelling of the nodes of L by the integers $1, 2, 3, \dots, |L|$ so that each node is labelled by a unique integer. An ordering for R is similarly defined.

A drawing (h,r) of a bipartite graph is an ordering h of the set L of vertices and an ordering r of the set R of vertices.

A crossing is a set $\{ (u,v), (w,x) \} \subseteq E$ of two edges where

1. $u,w \in L$ and $v,x \in R$

2. $h(u) < h(w)$ and $r(v) > r(x)$ or, $h(u) > h(w)$ and $r(v) < r(x)$.

A drawing is optimal if there is no other with fewer crossings; i.e. if it is an optimal solution to the BDP.

A drawing (h,r) is left optimal when there is no other ordering h' of L so that (h',r) has fewer crossings. A drawing (h,r) is right optimal if there is no other ordering r' of R so that (h,r') has fewer crossings.

As it can be seen from the example shown in Eades and Kelly (1986), the condition of being simultaneously right optimal and left optimal does not necessarily imply that the drawing is optimal.

Suppose that r is a fixed ordering of R and i,j are two vertices of L . $K(i,j)$ is defined as the number of crossings that edges incident to vertex i make with edges incident to vertex j , when the vertex i precedes j in the ordering of L .

Note that given $i,j \in L$, the value of $K(i,j)$ only depends on the ordering of the vertices of R , and not on the position of the remaining vertices of L . The parameter $K(i,j)$ is similarly defined relative to i,j vertices of R , for an ordering h of L .

To compute $K(i,j)$ we can "merge" N_i with N_j considering the relative position between its elements.

Given two orderings h and r of L and R respectively, let $K(G,h,r)$ be the total number of arc crossings of G with respect to these orderings. $K(G,h,r)$ can be obtained as the sum of the $K(i,j)$ values with $h(i) < h(j)$ for all the vertices i,j of L (or similarly as the sum of all the $K(i,j)$ with $r(i) < r(j)$ for all the i,j of R):

$$K(G, h, r) = \sum_{h(i) < h(j)} K(i, j)$$

Given G and the orderings h and r of L and R respectively, where h is given by the permutation $(v_1, v_2, \dots, v_i, \dots, v_j, \dots, v_n)$. Let h' be the ordering of L obtained by removing vertex v_j from its position and inserting it

in position i (between vertices v_{i-1} and v_i). Then the number of crossings $K(G,h',r)$ can be obtained from $K(G,h,r)$ as follows:

$$K(G,h',r) = K(G,h,r) - \text{PREV}(i,j,h,r) \text{ where}$$

$$\text{PREV}(i,j,h,r) = \sum_{t=i}^{j-1} K(v_t, v_j) - \sum_{t=i}^{j-1} K(v_j, v_t)$$

Similarly we can obtain the expression of the number of crossings when one vertex is moved to a posterior position in the ordering of its layer, while the ordering of the other layer remains fixed. Given G and the orderings h and r of L and R respectively, where h is given by the permutation $(v_1, v_2, \dots, v_i, \dots, v_j, \dots, v_n)$. Let h' be the ordering of L obtained by removing vertex v_i from its position and inserting it in position j (between vertices v_j and v_{j+1}). Then:

$$K(G,h',r) = K(G,h,r) - \text{POST}(i,j,h,r) \text{ where}$$

$$\text{POST}(i,j,h,r) = \sum_{t=i+1}^j K(v_i, v_t) - \sum_{t=i+1}^j K(v_t, v_i)$$

The parameters POST and PREV can be similarly obtained for a moving vertex in R while L remains fixed.

3. The Bipartite Drawing Problem

Watkins (1970) was the first to study the BDP. He introduces a special crossing number for bipartite graphs as an open research problem: the Bipartite Crossing Number (BCN). The BCN of a bipartite graph is the number of crossings of the optimal solution of the BDP.

Warfield (1977) develops a crossing reduction theory. Using a matrix to represent a bipartite graph, he provides formulas to compute the number of crossings.

Carpano (1980) proposes the Relative Degree Algorithm. The algorithm assigns initial coordinates to the vertices of one layer of the bipartite graph. The coordinates of each vertex of the other layer are then calculated as the arithmetic mean of the coordinates of the adjacent vertices. With this method, the procedure alternately calculates the coordinates of the vertices of each layer, while the coordinates of the vertices of the other layer remain fixed. The

algorithm is applied until two successive iterations occur in which the relative positions of the vertices remain the same. The author shows that the algorithm produced a considerable reduction in the number of crossings (30 to 50 percent less) in real-life examples.

Sugiyama, Tagawa and Toda (1981) propose the Barycentric Method (BC) that is similar to Carpano's algorithm but improves upon it in that it preserves the original order when two vertices have the same barycentre. Computational results are given for bipartite graphs with 7 ($|L|=3 |R|=4$) and 11 ($|L|=5 |R|=6$) vertices.

Eades and Kelly (1986) present four heuristics (Greedy Insertion, Greedy Switching, Splitting and Averaging) for left optimality, which can be symmetrically applied for right optimality. To obtain optimality, they alternately apply the heuristics for left and right optimality until no reduction in crossings is made. The Averaging algorithm is the same as Carpano's (1980).

In order to compare the quality of the solutions given by the four algorithms, Eades and Kelly work with a sample of 100 randomly generated graphs with 40 vertices ($|L|=|R|=20$). With the ordering of R fixed, the algorithms were seeking left optimal solutions. The performance is measured by the ratio of the number of crossings given by the heuristic over the lower bound on the left optimal drawing. The results show that the Greedy Insertion algorithm is clearly the worst of the four. Of the other three, the Splitting (SP) and the Greedy Switching (GS) show the best performance.

Eades and Wormald (1986) present the median heuristic (ME) which considers two readability elements of graph drawing: Edge crossing and edge length. The procedure reorders the vertices of L while the vertices of R remain fixed. For each vertex u of the graph, it considers a coordinate $y(u)$. The new coordinate of each vertex u is the median of its neighbours' coordinates, that is, if $N_u=\{v_1, v_2, \dots, v_k\}$ where $y(v_1) \leq y(v_2) \leq \dots \leq y(v_k)$ then $y(u)=y(v_m)$ where $m = \lceil k / 2 \rceil$

Makinen (1990) performs a computational study to compare the BC and ME. He also considers two hybrid heuristics which differ from the ME heuristic only when a vertex has even degree. In this case the

Average Median heuristic assigns the arithmetic mean of the positions of the two middle adjacent vertices. The Semi Median heuristic assigns, when the vertex has even degree, the same position as the barycenter heuristic. In the computational study, the author considers the order of L fixed and applies the heuristic to reorder the vertices of R. The results show that the BC is the best heuristic followed by the Semi-Median.

In Martí (1993), we present a Branch and Bound algorithm to obtain the optimal solution for the BDP. The computational study shows the influence of the number and the distribution of the vertices in the performance of the algorithm.

Eades and Wormald (1994) perform a theoretical study of BC and ME heuristics. They prove that the median heuristic gives a drawing with no more than three times the left optimal number of crossings and whose total length is at most $2/\sqrt{3}$ the left optimal length. (A drawing is left-length optimal if no re-positioning of the vertices in L, holding R fixed, achieves a smaller total length). They show that, in worst case, the barycenter gives a solution with no more than k times the left optimal number of crossings, where k is $o(\sqrt{|L|})$. They also show an example with n vertices for which the solution of the BC is $n-4$ but the left optimal is 1.

In Valls, Martí and Lino (1994) we present a heuristic algorithm based on the Tabu Search optimization techniques to solve the BDP. In particular, we consider the Tabu Thresholding methods (Glover, 1992) that implement the fundamental concepts of Tabu Search without explicitly using memory structures. Tabu Thresholding combines elements of probabilistic Tabu Search with candidate list procedures derived from network optimization studies. We adapted some of the Tabu Thresholding methods to our specific implementation. A computational study shows that in each of the 250 examples tested, the solution of the algorithm is better than or equal to those given by the GS and SP heuristics.

When attempting to reduce the number of arc crossings, some graph drawing systems employ algorithms that are extensions and variations of the barycenter method. However if the graph is a 2-layered

hierarchy (bipartite graph), these algorithms are the same as some of those presented above.

4. A Search Scheme for the BDP

The Tabu Search methodology was proposed by Glover (1986). Tabu Search (TS) is an adaptive technique for solving hard combinatorial optimization problems based on general principles of intelligent problem solving. In essence it is a metaheuristic which may be used to guide any local search procedure in the aggressive search for a global optimum, using memory structures to avoid getting trapped in local optima. A complete and precise description of TS may be found in Glover (1989, 1990).

TS becomes significantly stronger by including intermediate and long term memory functions, designed to provide local *Intensification* and global *Diversification* in the search process. Thus, over the long term, TS induces an interplay between these two phases (Intensification and Diversification) that introduces a dynamic and aggressive search strategy.

A major aspect of tabu search derives from inducing a *Strategic Oscillation* over the long term. This permits the discovery of good solutions in new regions that normally are not explored. Using the Intensification, Diversification and Strategic Oscillation techniques of Tabu Search we propose a neighbourhood search procedure for a global optimum of the BDP.

Let X be the set of possible solutions of a given problem. In any local search method, each solution $x \in X$ has an associated set of solutions $N(x) \subseteq X$, known as the neighbourhood of x . Each solution $x' \in N(x)$ may be obtained directly from x by means of an operation known as move. Associated with each move there is a move value, which represents the change of the objective function as a result of the proposed move. The move value provides an evaluation of the quality of the move.

Our procedure consists of two alternately applied phases, an Intensification Phase and a Diversification Phase. In the Intensification Phase only improving moves may be performed and it ends with a

local optimum. In the Diversification phase both non-improving and improving moves may be performed with the aim of escaping from the local optimum and diversifying the search to other regions in the solution space.

Given a bipartite graph, let L and R be the partitions of the set of vertices and h and r some orderings of these subsets. The ordered set $M=L \cup R$ is defined in which the elements of L according to the order h are placed first followed by the vertices of R according to r .

4.1 Intensification Phase

The position of a vertex i can be calculated as a function of the positions of its adjacent vertices. For example we can use the average or the median as in some heuristics reviewed in Section 3. Let $p(i)$ be that position, we may consider removing i from its current position and inserting it in position $p(i)$. $m(i)$ denotes the insertion move associated with vertex i .

Let $c(i)$ be the current position of vertex i ; i.e. $c(i)=h(i)$ if $i \in L$ or $c(i)=r(i)$ if $i \in R$. The evaluation of the move $E(m(i))$ is given by the change in the number of crossings when the move is performed.

$$E(m(i)) = \begin{cases} \text{PREV}(p(i), c(i), h, r) & \text{if } p(i) < c(i) \\ \text{POST}(c(i), p(i), h, r) & \text{if } p(i) > c(i) \\ 0 & \text{if } p(i) = c(i) \end{cases}$$

Thus, if C is the variable that measures the total number of crossings in the current solution, by performing move $m(i)$, the update of this variable is: $C = C - E(m(i))$. Therefore, given that the problem is to minimize the number of arc crossings, a move is better the higher its evaluation.

In this phase the procedure explores the set M as a block ordered circular list. Thus in one iteration it takes a Block (a subset of consecutive vertices of M) and randomly selects a vertex v in this Block. If $E(m(v)) > 0$ the move is performed, updating the solution and

the order of M. In the next iteration it takes the following block, repeating the process until the stopping criterion is satisfied. Let BS be the Block Size.

Given that in this phase the algorithm only allows a move to be made when the number of crossings strictly diminishes, the stopping criterion is based on making a maximum number of iterations MAX_I where no improvement is produced. Thus, it will be considered that after MAX_I has been reached without improvement, the current ordering is sufficiently close to the local optimum to stop this phase.

It is worth mentioning that the structure of the problem makes it easy to update the order of M after each move. This progressive reordering along with the effect of the blocks is sufficient in our case to avoid cycling.

Intensification Phase Procedure

-
- 1.- Consider the solution given by the current orderings of L and R.
Construct $M = L \cup R$ as an ordered circular list.
 $no_change = 0$

 - 2.- WHILE($no_change < MAX_I$)
 - Let B be the block of M formed by the next BS elements.
 - Select a vertex i randomly in B.
 - Calculate the move $m(i)$.
 - IF($E(m(i)) > 0$)
 - Perform the move.
 - Update the solution and M.
 - $no_change = 0$
 - ELSE
 - Perform no move.
 - $no_change = no_change + 1$.
-

4.2 Diversification Phase

Suppose a vertex i of the graph; let j be the element immediately preceding i in the order of this layer. We may consider permuting both vertices, where $m(i)-$ denotes this move. Let t be the element immediately following i in the order of this layer. We may also consider permuting both vertices, where $m(i)+$ denotes this move. Therefore all the vertices have two associated movements, except for the first and the last vertex of each layer that only have one.

Given that j precedes i and i precedes t in the current ordering, the evaluation of the moves $m(i)-$ and $m(i)+$ is given by the following parameters:

$$E(m(i)-) = K(j,i) - K(i,j) \quad E(m(i)+) = K(i,t) - K(t,i)$$

In this phase the procedure explores the set M as a full ordered circular list. Thus in each iteration it randomly selects a vertex v of M and performs the best (highest evaluation) move. As in the intensification phase, by implementing movement $m(i)-$, the update of the number of crossings is: $C = C - E(m(i)-)$ and similarly for $m(i)+$. In this phase, both improving and non-improving moves may be performed with the aim of escaping from the local optimum, and diversifying the search to other regions in the search space. Let the parameter D_IT be the number of iterations in this phase.

Diversification Phase Procedure

- 1.- *Let the initial solution be that obtained in the last iteration of the previous Intensification Phase.*
Select t randomly between L and U . $n_it = 0$.

- 2.- **WHILE($n_it < D_IT$)**
 - Select a vertex i randomly.*
 - IF(i is the first or the last in its layer)*
 - Calculate the only move associated to i .*
 - Perform the move.*
 - Update the solution.*
 - ELSE*
 - Calculate $m(i)-$ and $m(i)+$.*
 - Perform the move with highest evaluation.*
 - Update the solution.*

$n_it = n_it + 1$.

In this phase the solution may get better or worse in each iteration; therefore the current solution should be contrasted with the best stored solution each time a move is made, updating when necessary.

The moves defined in the intensification are based on a positioning function while those defined in the diversification are based on permuting consecutive vertices. The use of different moves fulfils the role of altering the terrain visited in a way that is unlikely to occur by always applying the same kind of moves; thus reinforcing the non-monotonic search strategy.

4.3 Strategic Oscillation

The Strategic Oscillation (SO) provides an interplay between Intensification and Diversification over the intermediate and long term. SO operates by crossing certain boundaries that can be defined in terms of feasibility/unfeasibility or in terms of a region where the search tends to stay. The oscillation then consists of compelling the search to move out of this region and allowing it to return. The process of repeatedly crossing the boundary creates a form of oscillation that provides auxiliary forms of intensification and diversification. SO can be controlled by altering the objective function by penalties or generating modified move evaluations that dynamically change depending on the region currently visited.

In our approach we have adapted these ideas by means of a new evaluation of the moves: $E_2(\text{move}) = S * E(\text{move})$ where $S=-1,0,1$. The new evaluation function E_2 is used to select the moves in both phases. Once a move has been performed, the update in the number of crossings C is made with the original evaluator E described before; i.e., $C = C - E(\text{move})$.

The algorithm starts with $S=1$. If the best solution does not improve after SO_B iterations, then P is decreased to 0 and begins to oscillate following the SO-sequence: 1, 0, -1, 0, 1, 0, -1, When the best solution is improved P is re-initialized to 1.

If $S=0$ then $E_2(\text{move})=0 \forall \text{move}$, therefore in the intensification phase no move is accepted and in the diversification the search is completely random. If $S=-1$ then $E_2(\text{move})=-E(\text{move})$, therefore in the intensification phase only dis-improving moves are accepted and in the diversification the algorithm selects in each step the worst move associated with a given vertex (opposite effect).

The oscillation of the parameter S induces an oscillation in the search strategy among three states, normal ($S=1$), random ($S=0$) and opposite ($S=-1$) which provides an aggressive diversification scheme.

The global procedure controls the alternation between the Intensification and Diversification phases and the Oscillation status of

the search. The Diversification is always performed after the Intensification phase. Both phases are alternated until the global stopping criterion is met. The value of the parameter S manages the state of the search and is updated each time both phases have been performed. When the Intensification phase has finished, the number of crossings in the ordering obtained is contrasted with that obtained with the previous application of the phase; the algorithm finishes when after a number of complete iterations equal to G_IT (Global Iterations) the solution has not improved.

Global Procedure

1.- Consider an initial solution.

*Let BEST_C be the number of crossings of the initial solution.
change = 0, S = 1.*

2.- While(change < G_IT)

2.1 Perform the Intensification phase.

Let IC be the number of crossings of the solution obtained.

IF (IC < BEST_C)

BEST_C = IC, change = 0, S = 1

ELSE

change = change + 1.

2.2 Perform the Diversification phase.

*Let DC be the number of crossings of the best solution
obtained in this phase.*

IF (DC < BEST_C)

BEST_C = DC.

2.3 Strategic Oscillation.

IF (change > SO_B)

Update S with the next value in the SO-sequence.

The diagram of Figure 1 shows an example that illustrates the performance of the Strategic Oscillation. The x-coordinate represents the number of global iterations (Intens. + Diver.). The diagram shows the number of crossings (y-coordinate) of the solution obtained each time the Intensification phase finishes. In global iteration 86, the best solution stored (157 crossings) has not improved in the last 50 global iterations, so oscillation begins. Figure 1 shows the great perturbation in the objective function and how "far" solutions are visited when S=0 and S=-1. As a result, after four oscillations the best solution is

improved (152 crossings) re-initializing the oscillation status ($S=1$) and continuing the search.

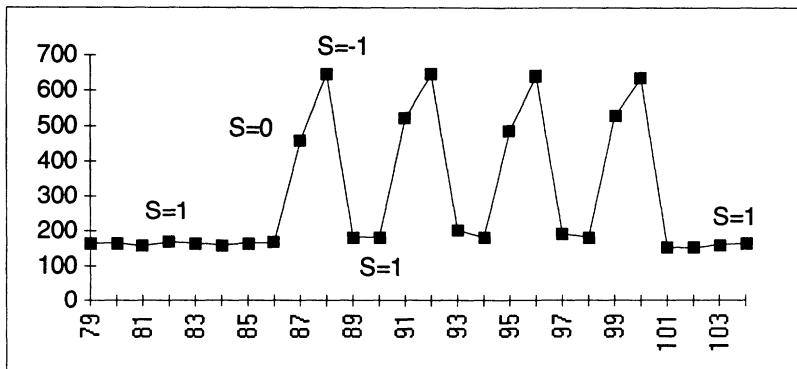


Figure 1: Number of crossings in the oscillation process.

5. Computational Results

Different algorithms can be obtained with the scheme of Section 4 by implementing different positioning functions $p(i)$ in the intensification phase. Taking into account the algorithms and results reviewed in Section 3 we have implemented two of them: the Barycenter function presented in Sugiyama, Tagawa and Toda (1981) and the Semi-Median function presented in Makinen (1990). Let A_{BC} be the algorithm obtained by implementing the Barycenter and A_{SM} the SemiMedian.

Bearing in mind the theoretical and computational studies mentioned in Section 3 we have compared the two new algorithms with the GS, SP, BC and SM heuristics. These 4 algorithms have been implemented to alternately reorder L and R while the other layer remains fixed until no further improvement is possible. We have also compared the new algorithms with our Tabu Thresholding procedure (TT) presented in Valls, Martí and Lino (1994) and with the optimal solution for relatively small test problems (up to 26 vertices).

Preliminary tests were carried out on a set of randomly generated instances in order to determine the values of key parameters in our search scheme. The algorithm for the random generation of bipartite graphs is described in Valls, Martí and Lino (1994). After experimentation, the following values produced best results in terms of

solution quality and computation time: MAX_I = 50, BS = 5, D_IT = 25*(|L|+|R|+|E|), G_IT = 100 and SO_B=75.

To measure and compare the efficiency of the new algorithms, tests have been carried out on a collection of 300 randomly generated instances. In all instances the number of vertices has been set to be the same on both layers ($|L| = |R| = n/2$, where n is the total number of vertices in the graph). The instances have been divided into 15 groups according to the size: $n=14, 16, 18, 20, 22, 24, 26, 30, 40, 50, 60, 70, 80, 90, 100$. For each size 20 instances were tested. In all the algorithms the initial solution is the one provided by the random generator.

The results are summarized in Table 1, where D_OPT is the average percentage distance to optimum, D_BEST is the average percentage distance to the best solution of the 7 algorithms, M_OPT is the maximum percentage distance to the optimum and M_BEST is the maximum percentage distance to the best solution of the 7 algorithms. D_OPT and M_OPT are calculated over the 140 problems for which the optimum is known.

Table 1: Average percentage distance to optimum.

	D_OPT	M_OPT	D_BEST	M_BEST
GS	42	238	22.4	238.4
SP	29	155	14.6	155
BC	25	166	12.8	166
SM	38	166	21.4	166.6
TT	0.9	15	0.8	15
A_BC	0.01	1.1	0.03	1.13
A_SM	0.008	0.39	0.03	0.45

The running times of the A_BC and A_SM algorithms, coded in C on a PC 486-66Mhz, range from 0.5 sec. ($n=14$) to 20 min. ($n=100$).

From the computational results it is concluded that A_BC and A_SM clearly outperforms the 4 simple descent heuristics: In all the examples tested, the solution of the two new algorithms improved up on that of the other four heuristics. Moreover, considering the 7 algorithms, the

A_SM appears to show the best performance, closely followed by the A_BC which performs better than TT.

6. Conclusions and Further Research

In this paper we have presented a neighbourhood search scheme for the bipartite drawing problem. This scheme is based on the Intensification and Diversification strategies of Tabu Search. Special emphasis is given to an aggressive diversification by means of the Strategic Oscillation technique. Two different algorithms (A_BC and A_SM) have been obtained with this scheme by implementing two positioning functions in the definition of the moves.

The computational study has been carried out on a set of 300 randomly generated problems. The two algorithms have been compared with the four best simple descent algorithms and with the Tabu Thresholding (TT) procedure. It is clear that A_BC, A_SM and TT provide significantly better solutions than the four descent algorithms. In addition, A_BC and A_SM perform better than TT.

This study can be extended in two ways to obtain a graph drawing system based on Tabu Search techniques: Generalize the algorithms to acyclic digraphs and introduce various objective functions that identify a simple set of aesthetic criteria of graph drawing. The search scheme presented can be easily adapted to optimize various objective functions by simply implementing suitable positioning functions in the definition of the moves. Given that the A_SM considers both edge crossing and edge length in its implementation, it may well be a good starting point for future research.

Acknowledgement

The author thanks professor Vicente Valls for his useful comments. The author is also grateful to professor Manuel Laguna for suggestions to improve the presentation of the paper.

7. References

M.J. Carpano, Automatic Display of Hierarchized graphs for computer aided decision analysis, *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-10, 11 (1980), 705.

- G. Di Battista, P. Eades, R. Tamassia and I.G. Tollis, Algorithms for drawing graphs: an annotated bibliography, Working paper, Department of Computer Science, Brown University (1993).
- P. Eades and D. Kelly, Heuristics for drawing 2-layered networks, *Ars Combinatoria*, 21 (1986) 89.
- P. Eades, B. McKay and N.C. Wormald, An NP-complete crossing number problem for bipartite graphs, Working paper, Department of Computer Science, 60, Univ. of Queensland, (1985).
- P. Eades and N.C. Wormald, The median heuristic for drawing 2-layered networks, Working paper, Department of Computer Science, 60, Univ. of Queensland, (1986).
- P. Eades and N.C. Wormald, Edge crossing in drawings of bipartite graphs, *Algorithmica*, 11 (1994) 379.
- F. Glover, Future Paths for Integer Programming and Links to Artificial Intelligence, *Computers and Operations Research*, 13 (1986) 533.
- F. Glover, Tabu Search: Part I, *ORSA Journal on Computing*, 1 (1989) 190.
- F. Glover, Tabu Search: Part II, *ORSA Journal on Computing*, 1 (1990) 4.
- F. Glover, Simple Tabu Thresholding in optimization, Working paper, School of Business, University of Colorado, Boulder. (1992).
- D. S. Johnson, The NP-completeness column: an ongoing guide, *Journal of Algorithms*, 3 (1982) 288.
- E. Makinen, Experiments on drawing 2-level hierarchical graphs, *International Journal of Computer Mathematics*, 36 (1990).
- R. Martí, Técnicas Exactas y Heurísticas en el Diseño Automático de Redes de Planificación de Proyectos, P.h. D. Thesis, Depto de Estadística e I. O., Univ. de Valencia, Spain, (1993).
- K. Sugiyama, S. Tagawa and M. Toda, Methods for visual understanding of Hierarchical system structures, *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11, 2 (1981) 109.
- V. Valls, R. Martí and P. Lino, A Tabu Thresholding Algorithm for Arc Crossing Minimization in Bipartite Graphs, in: *Annals of Operations Research. vol. 60. Metaheuristics in Combinatorial Optimization* (Baltzer Science Publishers, Amsterdam, 1995).
- J.N. Warfield, Crossing theory and hierarchy mapping, *IEEE Trans. on Systems, Man, and Cybernetics*, SMC-7, 7 (1977) 505.
- M.E. Watkins, A special crossing number for bipartite graphs: a research problem, *Ann. New York Acad. Sci.*, 175, 1 (1970) 405.

Guided Search for the Shortest Path on Transportation Networks

Yazid M. Sharaiha and Richard Thaiss

Operational Research and Systems

The Management School

Imperial College of Science, Technology and Medicine

53 Prince's Gate, London SW7 2PG, United Kingdom

Email: y.sharaiha@ic.ac.uk

Abstract

This paper addresses the well-known problem of locating a shortest path between a start and a terminal vertex on a network with non-negative arc costs. We present a new approach which guides the search of the shortest path algorithms under investigation. The ordering of vertices in the vertex list is performed by dividing the network into a set of disjoint regions, each represented by a queue. Each queue holds a set of vertices which fall within a range given by a heuristic measure of their current proximity to the end vertex. Computational results show that the guided algorithm outperforms the original algorithms for transportation networks with Euclidean arc costs. The results are analyzed against the queue size and the accuracy of the heuristic function used.

Key Words: Networks, search, shortest path, transportation.

1. Introduction

This paper addresses the well-known problem of locating a shortest path between a start and a terminal vertex on a network with non-negative arc costs. A number of papers comparing the theoretical and

computational performance of different shortest path algorithms have appeared in the operational research literature (see Golden (1976), Dial *et al* (1979), Glover *et al* (1985), Gallo and Pallottino (1988), Hung and Divkoy (1988) and Bertsekas (1993)). The performance of each algorithm is related to the network structure (e.g., sparsity vs. density) and arc costs (e.g., variance and maximum value). These papers have typically addressed the problem of finding the shortest path tree from a named start vertex to all other vertices in the network. In the case of locating a single shortest path between two specified vertices, research in the artificial intelligence community has demonstrated how Dijkstra's classical algorithm (1959) can be adapted by guiding the shortest path using search heuristics. The objective is to limit the search space of the algorithm without compromising the optimality of the final solution. We will refer to such algorithms as guided algorithms. Hart, Nilsson and Raphael (1968) introduced the main idea of utilizing a heuristic measure into the shortest path algorithm in order to improve its efficiency. Later, Pohl (1970) and Nemhauser (1972) extended their results for a guided version of Dijkstra's algorithm for different types of networks, and showed how the search space for the single shortest path problem can be reduced significantly. Golden and Ball (1978) presented their results for the case of lattice networks with Euclidean arc costs.

We first review the guided version of the generic algorithm for solving the single shortest path problem from s to t on the network $\mathbf{G}=(\mathbf{V}, \mathbf{A})$. The search directs the path towards t by using a heuristic function $h(v)$ to estimate the path length from v to t . Let:

$c(u,v)$ be the cost of arc (u,v) ,

(if no arc exists and $u \neq v$, let $c(u,v)$ take a very large positive number, and $c(v,v)=0$ for all vertices $v \in V$),

$l(v)$ be the path length label (not necessarily minimum) from s to v ,

$h(v)$ be a real-valued function which satisfies the following property:

$$h(u) \leq h(v) + c(u,v) \text{ for all arcs } (u,v) \in A.$$

If $h(t) = 0$, then $h(v)$ is a lower bound on the shortest path length from v to t .

The steps of the generic guided search algorithm are as follows (see Nemhauser (1972)):

Step 0: Initialize: $S = V - \{s\}$, $l(s) = 0$, $l(v) = c(s, v)$ for $v \neq s$.

Step 1: Select v^* from the list S such that:

$$l(v^*) + h(v^*) = \min_{v \in S} [l(v) + h(v)].$$

If $v^* = t$, terminate: $l(t)$ is the length of the shortest path from s to t .

Otherwise, let $S \leftarrow S - \{v^*\}$.

Step 2: For all $v \in S$, let:

$$l(v) = \min [l(v), l(v^*) + c(v^*, v)].$$

Return to Step 1.

Hart, Nilsson and Raphael (1968) and Nemhauser (1972) showed that this algorithm will always find a shortest path from s to t . When $h(v) = 0$ for all $v \in V$, then the above algorithm reduces to Dijkstra's original algorithm. Whereas an ordered queue implementation of Dijkstra's algorithm would sort the vertices of the candidate list S by their non-decreasing path length labels $l(.)$ and choose the one with the minimum label, the guided implementation would now sort them out in the order of their $[l(.) + h(.)]$ labels. In the case of networks whose arc costs are given by a metric distance, then clearly that metric can be used as the heuristic function for guiding the search. In this case, the function immediately satisfies the lower bound constraint on the shortest path length to the end vertex. In particular, Golden and Ball (1978) studied the case where the arc costs are given by the Euclidean distances on the lattice network. For such networks, the results showed that on the average, the guided algorithm expands less than 8% of the area that would be searched by Dijkstra's algorithm.

In the case of the s to all t problem, the shortest path algorithms presented in the literature vary from Dijkstra's generic algorithm in the way they store the candidate vertex list, the policy used for placing/replacing vertices in the list, and the way in which they select the next vertex to be removed from this list at each iteration. The

algorithms can be divided into two distinct types: label correcting and label setting. The main difference between these two types of algorithms is that the label setting algorithms organize the list of vertices on the search boundary in such a way that the next vertex removed is always the one with the smallest label, and, therefore, is permanently labelled since all arc costs are non-negative. In the case of the label correcting algorithms, a vertex which is removed from the list may only be temporarily labelled and may, therefore, be added back to the list in the future. The following three algorithms (Bellman-Ford (1957,1956), D'Esopo-Pape (1980,1983) and Bertsekas's Shortest Label First (1993)) are label correcting algorithms and the following two (Dijkstra's Ordered Queue and Dial (1969)) are label setting. All of the algorithms under consideration use a queue (or set of queues) to store the list of vertices on the search boundary. Moreover, they all select the vertex at the head of the queue (or the head of the first non-empty queue) as the next vertex to be considered.

In this paper, we will first present a new guided search strategy for solving the s to t shortest path problem. In particular, the strategy is based on partitioning the search space into disjoint sets of oval-shaped regions, each represented by a queue. We study the case of large-scale transportation networks with Euclidean arc costs. The strategy has the advantage of avoiding the computational expense of ordering the vertices in the candidate list, since this is naturally embedded in the queuing system. Moreover, the search strategy can be adapted to both label setting and label correcting algorithms. We will adapt the new guided search strategy to four existing shortest path algorithms, and present comparative computational results. We also present results comparing the performance of the search strategy on each algorithm against the accuracy of the heuristic function.

2. The Algorithm

We will study the case of transportation networks with Euclidean arc costs. Such a network is defined by the graph $G(V,A)$ with a set of

vertices V and a set of arcs A , such that each arc $(u,v) \in A$ receives a cost $c(u,v)$ given by the Euclidean distance between u and v . Let the Euclidean distance between the start vertex (s) and the end vertex (t) be d , and suppose that a path (not necessarily minimal) has been found from s to another vertex v , which has a length label $l(v)$, as illustrated in Figure 1. The heuristic function $h(v)$ can be taken as the Euclidean distance from v to t since it is a valid lower bound on the shortest path length from v to t . Hence, the length of the shortest path from s to t which includes the path from s to v will be greater than or equal to $l(v) + h(v)$. We can now define a ratio of this value to the length d as follows:

$$r(v) = \frac{l(v) + h(v)}{d} \quad (1)$$

Since d is the minimum possible path length between s and t , then clearly $r(v) \geq 1$ for all v . The ratio $r(.)$ gives a measure of how useful a path going through v may be: vertices which have values of $r(.)$ which are closer to 1 are more likely to be part of a shortest path from s to t . Utilizing the $r(.)$ label could be achieved by keeping the list of vertices on the search boundary sorted in the order of their values of $r(.)$. However, this would suffer from the same disadvantage of the guided Ordered Queue implementation of Dijkstra's algorithm (i.e., the expense of maintaining the list of vertices in order). We propose a new method here which divides the vertices of the network into a series of 'ovals' as follows (Figure 1). The range of values which $r(.)$ can take is divided into a series of disjoint intervals of scalar length k , where k is a positive constant. Each vertex v can then be labelled according to the oval region number $n(v)$ that it belongs to. Since $r(v) \geq 1$, for all vertices v in the network, then $n(v)$ is the integer value which satisfies:

$$1 + k \cdot n(v) \leq r(v) < 1 + k \cdot (n(v) + 1).$$

Hence, $n(v)$ will be the integer part of:

$$\frac{r(v) - 1}{k} = \frac{l(v) + h(v) - d}{kd} \quad (2)$$

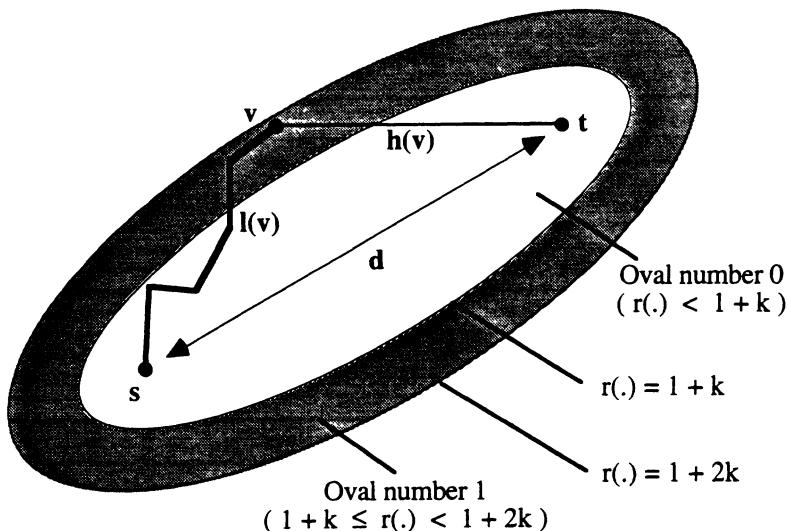


Figure 1. The division of the search space into oval regions using the $r(\cdot)$ intervals.

The vertex list is essentially stored as a set of queues, each corresponding to an oval region. The range of values of $r(\cdot)$ which is represented by each oval region is determined by the user-defined constant k . A relationship between the value of the interval size k and the number of ovals, and hence queues, Q , needs to be established. Since all of the vertices which have their forward stars examined are taken from the same queue, which corresponds to the lowest range of values of $r(\cdot)$, we need to establish the maximum increase in the value of $r(\cdot)$ for any of the vertices in these forward stars. Suppose then that v is a vertex which has just been removed from the vertex list and that a new shortest path has been found to vertex u which includes arc (v,u) . The increase in the ratio $r(\cdot)$ from v to u will therefore be given by:

$$\delta r = \frac{\delta l + \delta h}{d}$$

where δl is the increase in $l(\cdot)$ from v to u and δh is the increase in $h(\cdot)$ from v to u .

Since $\delta l = l(u) - l(v) = c(v,u)$, the maximum possible value of δl ($\delta_{\max} l$) will be the maximum arc cost in the network, c_{\max} . Moreover, since the function $h(.)$ is the Euclidean distance from a vertex to the end vertex, the maximum possible value of δh ($\delta_{\max} h$) will also be equal to c_{\max} (since $\delta h = h(u) - h(v) \leq c_{\max}$ by the triangle inequality). Thus the maximum possible increase in the ratio $r(.)$ from one oval to the next will be:

$$\delta_{\max} r = \frac{\delta_{\max} l + \delta_{\max} h}{d} = \frac{2c_{\max}}{d}$$

and the maximum possible increase in the oval number, $\delta_{\max} n$, will be:

$$\delta_{\max} n = \frac{2c_{\max}}{kd}$$

If we define Q as the number of queues required for the algorithm, then Q must satisfy the following inequality:

$$\delta_{\max} n + 1 \leq Q$$

Hence, for a given value of k , we have:

$$Q \geq \frac{2c_{\max}}{kd} + 1 \quad (3)$$

Alternatively, if Q is fixed, we must ensure that:

$$k \geq \frac{2c_{\max}}{(Q-1)d} \quad (4)$$

To divide the network into a series of ovals for the four algorithms (Bellman-Ford, D'Esopo-Pape, SLF and Ordered Queue), the vertex list is stored as a series of queues, where each queue corresponds to one of the range of values of $r(.)$. Thus, when a new shortest path is found to a particular vertex, the label of the vertex and the heuristic function are used to determine which oval's queue the vertex will belong to. If the vertex is already in this queue, or is not presently in any queue, the normal procedure for that algorithm is used to determine at which position in the queue the vertex should be placed. If the vertex was previously in a different queue, the vertex has to be removed from its old queue and added to the queue corresponding to its new value of $r(.)$. The normal criteria for the algorithm can then be used to determine at which position in the new queue the vertex should

be placed. The method of dividing the network into a series of ovals can, therefore, be applied to all of the shortest path algorithms under consideration in this paper. However, since Dial's algorithm would require a set of c_{\max} buckets for each oval, it is therefore not very suitable for this modification since the total number of queues required for a fixed value of k would be proportional to c_{\max}^2 .

3. Computational Results

In this section, we present some computational experiments performed using each of the algorithms. Comparative results are reported for the original algorithms and the guided version of the algorithms for the single shortest path problem. The test network used for these experiments was a road map of the United Kingdom with 15300 vertices and 24600 arcs. Similar results were also obtained for other road networks taken from Europe. Such networks are characterized as large and sparse. The computational results are obtained on a Silicon Graphics Indigo workstation. The number of iterations and CPU time (in seconds) taken by the algorithm to find the shortest path from the start to the end vertex are recorded. Average results for the number of iterations and CPU time are obtained by taking the means over 100 random runs for each algorithm (i.e., 100 random s, t pairs). For comparative purposes, the same set of 100 pairs were chosen for testing each algorithm (both for the original and guided versions).

We first compare the relative computational performance of the original shortest path algorithms for finding a single shortest path. In Figs. 2(a) and 2(b), average results over the 100 runs are shown on the bar charts, and the maximum and minimum over all runs are shown by the central straight line segment. The results for the Bellman-Ford algorithm are not plotted since the number of iterations and CPU time are much higher than the other algorithms and would, therefore, extend the y axis excessively. The mean results for this algorithm are in the range between 3 to 10 times larger in terms of number of iterations, and 3 to 8 times slower in terms of CPU time, relative to the slowest and

fastest algorithms, respectively. The poor performance of this algorithm is because once a shorter path length is found for a vertex, it is always added to the end of the vertex queue. This means that all of the other vertices in that queue have to be processed before any improvements in path lengths to other vertices can be propagated. This leads to the algorithm 'cycling round' sections of the network before it is labelled correctly. The D'Esopo-Pape algorithm performed very well, and was much better than any of the other label correcting algorithms. The reason for this is probably because changes in a vertex's label can be propagated quickly to other parts of the network, and thus minimizes the 'cycling' effect seen with the Bellman-Ford algorithm. The Small Label First algorithm performs well but is still much slower than the D'Esopo-Pape algorithm. The SLF strategy has outperformed the D'Esopo-Pape algorithm on several types of networks (see Bertsekas (1993)), but for the transport networks being considered here, it is the D'Esopo-Pape algorithm that performs best. Both label-setting algorithms, Ordered Queue and Dial, clearly take the same number of iterations. However, Dial's algorithm performed considerably better in terms of CPU time and has the best average overall performance. Dial's bucket storage is well-suited for transport networks which have a sparse structure.

The results of our guided version of each of the algorithms are reported in Figs. 3(a) and 3(b). For each of the guided shortest path algorithms, the number of iterations and CPU time taken with the large value of k (0.1) were significantly less than those of the original ones. With the small value of k (0.001), the reductions were further still, with each of the algorithms taking about the same number of iterations - less than a third of the number of iterations taken by Dial's original algorithm (the best performing of the original algorithms). The relative percentage reduction in CPU time for each algorithm is less than the reduction in the number of iterations for that algorithm because the cost per iteration has increased. This is due to the necessity to calculate the

value of $r(\cdot)$ (and hence the value of $n(\cdot)$) each time a new label is found for a vertex.

Although the average number of iterations was about the same for each algorithm, there is some, although quite small, difference in the CPU times taken, with the Bellman-Ford version taking the least time, followed by the D'Esopo-Pape algorithm, the SLF algorithm and finally Ordered Queue algorithm. By superimposing the oval search strategy, the computational performance of the guided algorithms converges. The slight differences in performance can be explained by the relative expense (in terms of CPU time) of the different strategies for placing vertices in the queue of each oval. Since the range of values of $r(\cdot)$ that each oval represents is very small, each queue corresponding to an oval will only contain a few vertices and, therefore, the way in which that queue is ordered becomes less important. Thus the ordering of the vertices in the vertex list (in terms of how 'useful' each vertex is) is predominantly performed by the division of the network into ovals rather than the strategy for placing each vertex in a queue. From this we can conclude that, with small values of k , the strategy used for placing vertices into a queue has been made redundant by the use of the guided algorithm and therefore the strategy for placing a vertex in a queue which is least expensive will be the one which leads to the fastest implementation of the guided algorithm. This is exactly what the results show since the Bellman-Ford has the simplest and fastest placement strategy.

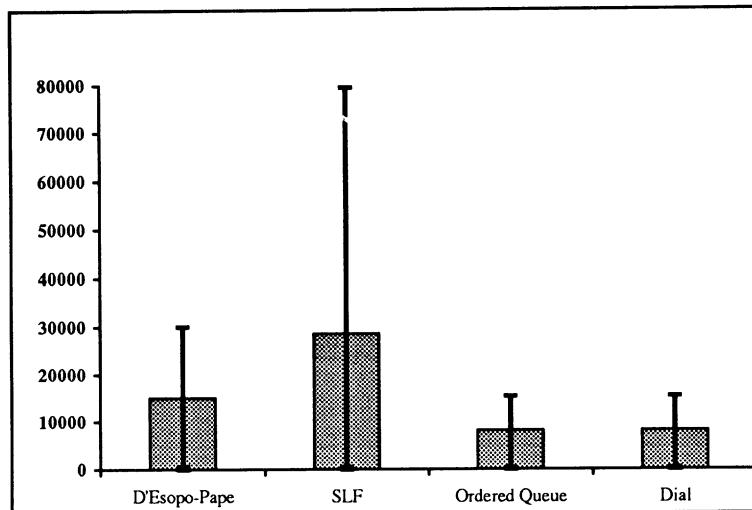


Figure 2(a): Average number of iterations
for the original algorithms.

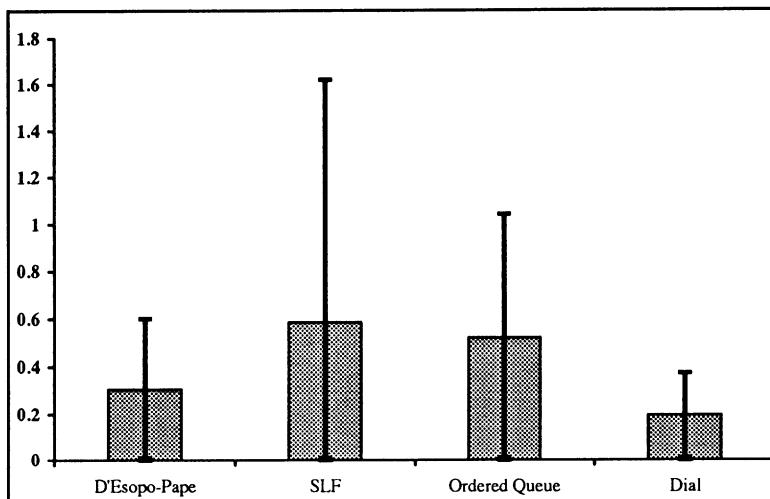


Figure 2(b): Average CPU time in seconds
for the original algorithms.

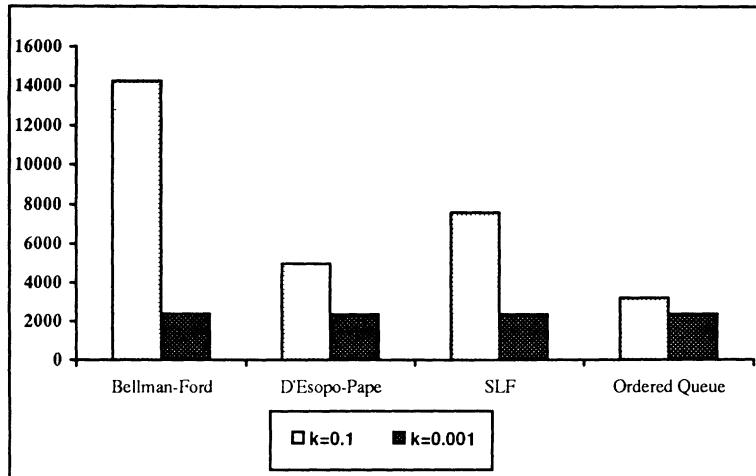


Figure 3(a): Average number of iterations using the guided algorithms against the two values of k .

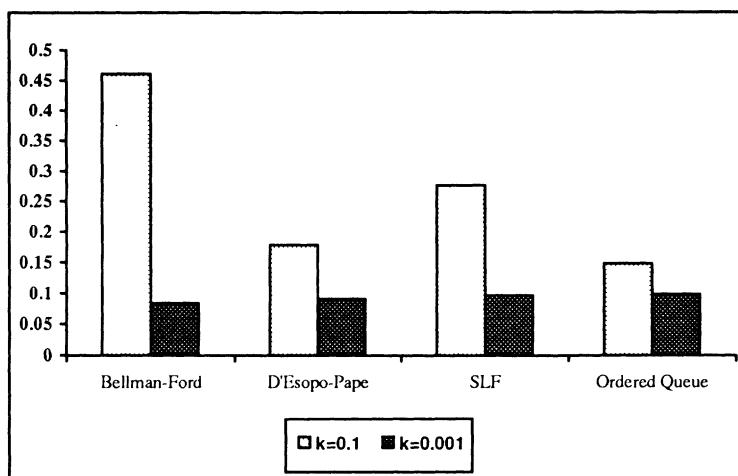


Figure 3(b): Average CPU time in seconds using the guided algorithms against the two values of k .

It is clear that the value of k , and hence the number of queues that need to be used, is very significant in determining the efficiency of the algorithms. For each guided algorithm, Figure 4 is a plot of the CPU time against a range of values of k . Empirical results show that, for each of the algorithms, the average CPU time remains constant for values of k below 0.001 approximately. This is probably because, for these very low values of k , for most (if not all) of the random paths in the test, the value of k is being clipped to its minimum 'safe' value for the maximum arc cost in the network and the (fixed) number of queues being used. Therefore, for this network type, the 'best' value of k for each of the shortest paths will probably be this minimum value of k . This will not always be the case if a very large number of queues are available. This is because the time taken for searching through the larger number of queues will outweigh the benefit gained from having more ovals with fewer vertices in each.

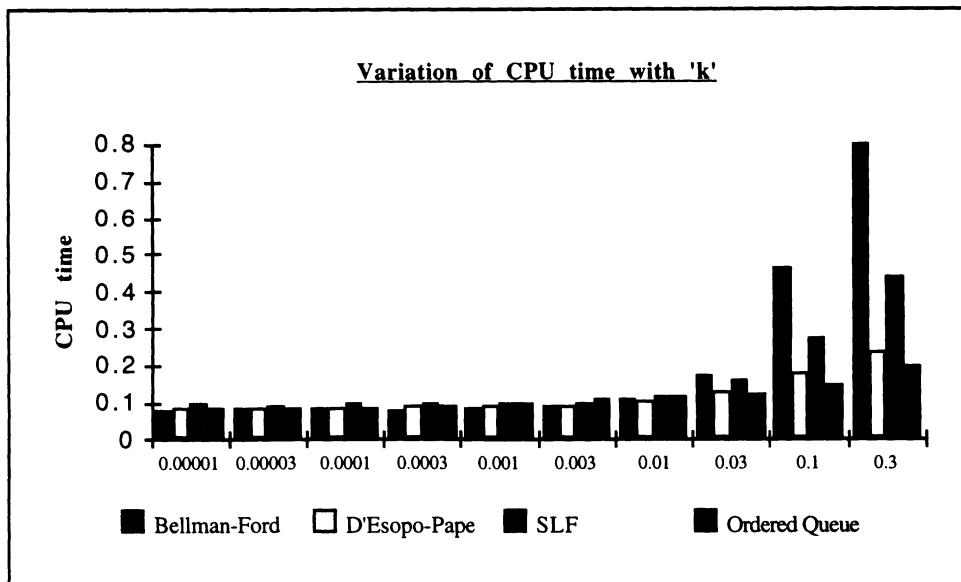


Figure 4: Variation in average CPU time in seconds with the value of k for the guided algorithms.

The results presented so far have only been for a single heuristic function. That is, $h(\cdot)$, which is the Euclidean distance between vertices. Since the arc costs in the transportation network are given by the Euclidean distance between the vertices at either end of the arc, the heuristic function can therefore give a very good estimate. In general, it will not always be possible to find such a good heuristic function as this. We therefore introduce two other heuristic functions in order to investigate the effect that the 'accuracy' of the function has on the performance of the guided algorithms. Let v be any vertex in V , and Δx and Δy be the difference in x- and y-coordinates between v and the end vertex t . Consider the following heuristic functions:

$$h(v) = \sqrt{[(\Delta x)^2 + (\Delta y)^2]} \quad (5)$$

$$h'(v) = \frac{|\Delta x| + |\Delta y|}{\sqrt{2}} \quad (6)$$

$$h''(v) = 0 \text{ for all } v \in V. \quad (7)$$

It can be easily shown that $h''(v) \leq h'(v) \leq h(v)$ for all $v \in V$. Since $h(v)$ is a valid lower bound on the shortest path length from any given v to t , the other two functions are also valid lower bounds. Indeed, $h''(\cdot)$ is a general lower bound which can be used for any network with non-negative arc costs. The guided algorithms were now run using the two new heuristic functions using the same 100 s, t pairs, and average results are reported for the case where $k=0.001$ in Figures 5(a) and 5(b). The dependence of the number of iterations on the accuracy of the heuristic function used is clear. For all of the guided algorithms, the size of the search space is in the order $h(\cdot) < h'(\cdot) < h''(\cdot)$, with $h(\cdot)$ taking the least number of iterations. For the CPU time, however, $h'(\cdot)$ outperformed $h(\cdot)$ for all but the SLF algorithm. This implies that it may be more efficient to use a less accurate heuristic function which is faster to compute than use the most accurate function available. The heuristic function $h''(\cdot)$ typically took about twice as long as the other two, but this function is still competitive with Dial's original algorithm.

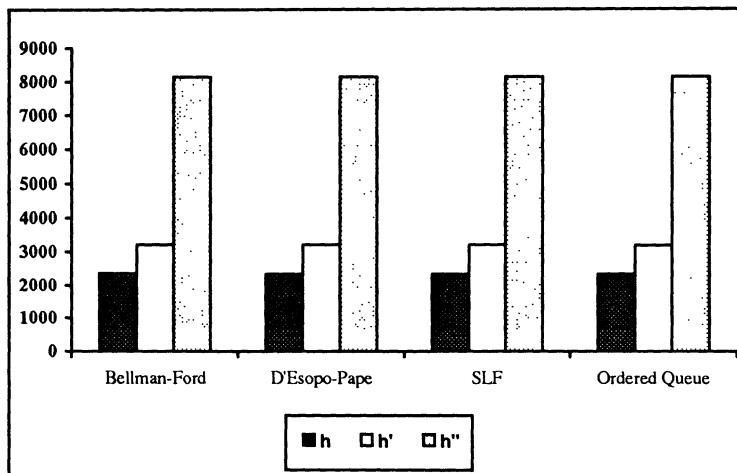


Figure 5(a): Average number of iterations for the guided algorithms against the heuristic function used for $k=0.001$.

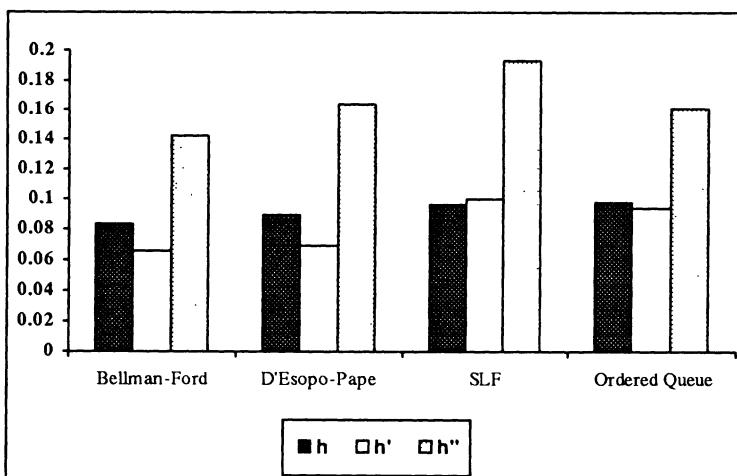


Figure 5(b): Average CPU time in seconds for the guided algorithms against the heuristic function used for $k=0.001$.

4. Conclusion

For transportation-type networks, the use of a heuristic function to direct the search can clearly provide a very significant improvement in the efficiency of shortest path algorithms. We presented a guided search strategy which places vertices in the vertex list by dividing the network into oval regions. We have shown that, on average, the best improvement in performance can be achieved using a large number of queues and an accurate heuristic function. For networks in which the arc costs are given by a metric, it is likely that this metric can be used as an accurate heuristic function. For transportation networks with arc costs reflecting travel times, a valid lower bounding function can be established using vehicle speed and distance. In the general case where only an inaccurate heuristic function can be found, or where it is not possible to find such a function, the guided algorithm can still be successfully applied and provide an increase in efficiency. In particular, the heuristic evaluation function $h''(.) = 0$ can be used for any network which has non-negative arc costs. Using a large number of queues, the guided algorithm using this inaccurate evaluation function remains competitive even with Dial's original algorithm. The guided algorithm can adjust the size of the parameter k to accommodate the number of queues required and the maximum arc cost, and it can also be applied for non-integer arc costs. There is potential for further research in the area of applying the oval search strategy for other network structures.

5. References

- R. Bellman, *Dynamic Programming*, (Princeton University Press, Princeton, NJ, 1957).
- D. P. Bertsekas, A Simple and Fast Label Correcting Algorithm for Shortest Paths, *Networks* 23, (1993) 703-709.
- R. B. Dial (1969). Algorithm 360: Shortest path forest with topological ordering, *Communications of the ACM* 12 (1969) 632-633.
- R. Dial, F. Glover, D. Karney and D. Klingman, A computational analysis of alternative algorithms and labeling techniques for finding shortest path trees, *Networks* 9 (1979) 215-248.

- E. W. Dijkstra, A note on two problems in connection with graphs, *Numerical Mathematics* 1 (1959) 269-171.
- L. R. Ford, Jr., Network flow theory, The Rand Corporation, Report No. P-293, Santa Monica, CA (1956).
- G. Gallo and S. Pallottino, Shortest path algorithms, *Annals of Operations Research* 13 (1988) 3-79.
- F. Glover, D. Klingman, N. Phillips and R.F. Shneider, New polynomial shortest path algorithms and their computational attributes, *Management Science* 31 (1985) 1106-1128.
- B. Golden, Shortest-path algorithms: a comparison, *Operations Research* 44 (1976) 1164-1168.
- B.L. Golden and M. Ball, Shortest paths with Euclidean distances: an explanatory model, *Networks* 8 (1978) 297-314.
- P. Hart, N. Nilsson and B. Raphael, A formal basis for the heuristic determination of minimum cost paths, *IEEE Transactions on Systems Science and Cybernetics* SSC-4, 2 (1968) 100-107.
- M.S. Hung and J.J. Divkoy, Computational study of efficient shortest path algorithms, *Computers and Operations Research* 15 (1988) 567-576.
- G. Nemhauser, A generalized permanent label setting algorithm for the shortest path between two specified nodes, *Journal of Mathematical Analysis and Applications* 38 (1972) 328-334.
- U. Pape, Implementation and efficiency of Moore algorithms for the shortest path problem. *Mathematical Programming* 7 (1974) 212-222.
- U. Pape, Algorithm 562: Shortest path lengths, *ACM Transactions on Mathematical Software* 6, 3, (1980) 450.
- U. Pape, Remark on algorithm 562, *ACM Transactions on Mathematical Software* 9, 2 (1983) 260.
- I. Pohl, Heuristic search viewed as path finding in a graph, *Artificial Intelligence* 1 (1970) 193-204.

A Metaheuristic for the Timetabling Problem.

H. Abada

E. El-Darzi

SCSISE,

University of Westminster,

115 New Cavendish Street,

London W1M 8JS, UK.

Email: eldarze@westminster.ac.uk

Abstract:

A metaheuristic algorithm based on Simulated Annealing and Tabu Search techniques is adapted to timetabling problems. The technique is designed with a certain flexibility to allow different schools and faculties at the University of Westminster to be able to develop their own timetables.

Keywords: Timetabling, education, simulated annealing, tabu search.

1. INTRODUCTION

In its simplest form the timetabling problem (TTP) consists of scheduling a set of meetings between lecturers and students over a period of time, requiring some resources and satisfying some additional constraints.

The TTP can be illustrated by the following example:

A computer science lecturer at the University of Westminster (UoW) is to be scheduled to teach software engineering for BSc students at levels (years) 1 and 3. He is also supervising a number of students at

Master and PhD levels. This lecturer cannot be available on Mondays as he is supervising a software development system. On Thursdays and Fridays he is committed to his research students at some specific hours. In his teaching materials, a computer room must be dedicated for the tutorials while a lecture theatre must be dedicated for his first year students. On the other hand, the students have to attend other modules with other lecturers with different requirements. At this stage the Head of the School found himself in front of a problem, which he called a timetabling problem.

Due to its complexity, this problem has been the subject of investigation by many researchers and mathematicians. A generally accepted notation or definition of the TTP has not been achieved up to the present though numerous articles appeared about this subject in the last forty years or so. However, the existence of any timetable must satisfy some fundamental conditions.

Before attempting to describe these conditions it is necessary to define the following terms:

Course: is a set of students following the same curriculum containing a set of core and optional modules (e.g. First year BSc in Computer Science) .

Group: is a subset of a course, in which students follow the same set of optional modules.

Module: is a teaching unit.

Event: is the lecture or the assignment that links a lecturer, students, a module and a room.

Time-slot: is the period of time during which an event takes place.

The necessary conditions for the existence of any timetable can be summarised as follows:

- (1) The total number of events should not exceed the total number of resources (rooms) available.
- (2) The largest number of students attending a given module should not exceed the capacity of the largest available room.
- (3) No student or lecturer is expected to be assigned to more than one place at any given time-slot.

In order to construct a timetable a scheduler should also take into account other constraints related to lecturers, students and resources.

1.1. Lecturers Constraints

Information about the availability of every lecturer, his or her status and preferable times of teaching should be available to the scheduler.

1.2. Students Constraints

Any educationally sound timetable should allow students some freedom in the choice of the optional modules. Lectures and modules should be distributed evenly during the week to allow the students to have affordable ability of understanding and accumulating knowledge.

1.3. Resources Constraints

In large institutions, like universities with limited resources (rooms), this factor can be very sensitive. In order to be able to subdivide the resources amongst schools and courses, the scheduler should be allowed some freedom in the control of the resources accordingly. However, the scheduler should take into account:

- (1) The heavy load that can be inflicted on areas like the refectory, the library and other utilities if large number of students are released at the same period of time.
- (2) The need for resources with special requirements (such as laboratories and seminar rooms).

In general the TTP has been shown to be NP-hard, Carter and Tovey (1992), which means that the running time for any algorithm currently known to guarantee an optimal solution is an exponential function of the size of the problem.

In this paper a metaheuristic algorithm based on Simulated Annealing (SA) and Tabu Search (TS) techniques is constructed to solve the TTP at the UoW taking into consideration the above mentioned factors.

In the next section a description of the problem at the university is presented. The model adopted in this paper is described in section 3. In section 4, brief descriptions of (SA) and (TS) techniques are given, followed by a description of the metaheuristic algorithm developed. Results of applying this algorithm to the TTP at UoW are presented in section 5. Finally conclusions about the performance and suitability of the algorithm are drawn in the last section.

2. Background and Problem Description

The University of Westminster, formerly the Polytechnic of Central London, trebled its size during the last two decades due to its merger with several colleges of Higher Education and increasing student numbers. It has more than 18,000 full and part-time students and about 1300 members of staff. As a result the institution is currently spread over 22 sites in London and offers a wide variety of degree courses ranging from Biotechnology to Photography.

The UoW consists of four Faculties. Each Faculty comprises 3 or 4 Schools and each School is further divided into a number of Divisions.

The timetable is constructed at School level. Each individual scheduler generates a "virtual" timetable, namely assigns events to time-slots without specifying the rooms where these events will be held. Based on this timetable, he/she then submits booking requests to the Accommodation Services (AS), a central unit responsible for the estate strategy of the institution. After collecting the booking requests of all Schools, AS allocate teaching rooms to each School trying to satisfy its requirements. If two (or more) Schools request the same space for the same time-slots there follows a series of internal negotiations until these conflicts are resolved. Finally, Schools are expected to release any space they do not need for other users.

The main difficulty arising during this process is that neither individual schedulers nor AS have a system which would accurately determine the minimum space each School needs to deliver its timetable.

Each scheduler somehow generates a feasible timetable trying not to waste space, but this can almost never be satisfied as the scheduler actually does not know whether there is an alternative timetable which uses less space.

Hence, it is essential to develop a computerised system which will offer schedulers the possibility to produce alternative feasible timetables and assess each of them in terms of space utilisation.

In this paper, the data collected is based on previous timetables delivered by each School, including the allocated space offered to each School by the Accommodation Service.

The system that is adopted at the UoW at the time being and which has been required by the Accommodation Services allows the schedulers at each School to schedule all modules in blocks of three or four hour time-slots, to a maximum of three blocks per day and up to 13 blocks over the whole week, as Wednesday afternoons have to be free for students to allow them to participate in national sport activities.

Although this system allows the AS to distribute the resources easily amongst the Faculties and Schools and offers the schedulers a reduced number of combinations, it has not been very acceptable by many lecturers and students because of educational requirements.

It has also been found that using this system the resources have not been efficiently utilised.

3. Model Description

Given the difficulties mentioned in the previous section we decided to

investigate a teaching pattern based on a one and half (1.5) hour time-slot. Modules that require more than one time-slot can be repeated during the week to complete the required teaching time.

There are twenty two (22) time-slots that are available for scheduling over five days (Monday to Friday). Lecturers attend between 6 and 10 time-slots during the week, while students attend at least 10 time-slots.

Our aim in this paper is to find a feasible course schedule for each School satisfying a set of constraints (conditions) that must be satisfied. These constraints have been specified by the scheduler based on the requirements of his/her School. They were given different penalties based on feasibility, high desirability, medium desirability and other requirements relevant only to specific schools. These constraints can be described as follows:

3.1. Feasibility Constraints

- Each lecturer is assigned at most one module at a time.
- Each group is assigned at most one module at a time.
- The capacity of a room assigned to a module must be higher than the number of students attending that module.

3.2. High Desirability Constraints

- Each lecturer/group is assigned at least two time-slots and not more than two successive time-slots on any given day of the week.
- The gap between two time-slots should not be more than one time-slot for any lecturer/group at any given day of the week.
- Every lecturer is offered a choice of a teaching-free day for research activities and other duties.
- No module is taught more than once at any given day of the week.
- Every room should be used at least 50% of the time.

3.3. Medium Desirability Constraints

- Each lecturer/group is assigned at most three time-slots at any given day of the week.

- No module is taught on two successive days.
- Every room should be used at least 80% of the time.

Some other constraints that have not been mentioned above are only relevant to a particular School or Faculty and the algorithm can be easily modified to cater for any additional constraints.

3.4. Objective Function

The objective function is made up of the penalties imposed on each constraint as described below:

Objective Function = Feasibility Penalty + High desirability Penalty + Medium desirability Penalty + Other constraints Penalties.

Our aim is to satisfy feasibility and minimize undesirabilities. Therefore a solution satisfying the desirability constraints will have a zero value objective function.

4. The Metaheuristic Algorithm

The metaheuristic used in this paper is based on a hybridisation of SA and TS techniques, which have been successfully used for solving the TTP.

4.1. Simulated Annealing

SA is based on randomized search and acceptance strategies, emerged from the analogy between the annealing process of solids and the problem of solving combinatorial optimization problems. A detailed description of the SA algorithm can be found in Dowsland (1993).

The success of SA lies most importantly on the choice of the cooling schedule and other generic decisions. For a recent classification of cooling schedules we refer to Osman and Christofides (1994).

In an investigation of SA, Abada and El-Darzi (1995a), have experimented with four different cooling schedules. The best results for all the problems experimented with, were obtained using a non-monotonic cooling schedule introduced in Osman (1993). This cooling

schedule is used in the hybrid algorithm.

Successful applications of SA to the TTP have also been reported by Abramson (1991), Abramson and Dang (1993) and Dowsland (1990).

4.2. Tabu Search

TS is based on constraining and freeing search. It is drawn from the general tenets of intelligent problem solving. A detailed description of the TS algorithm can be found in Glover and Laguna (1993).

The success of TS lies most importantly on the choice of the short-term memory components (tabu restrictions, tabu tenure, aspiration criteria, etc.).

In an investigation of TS, Abada and El-Darzi (1995b), have experimented with different memory strategies. The best results for all the problems experimented with, were obtained using a memory strategy based on varying the tabu tenure. This strategy will be used in the hybrid algorithm. Successful applications of TS to the TTP have also been reported by Alvarez-Valdes et.al (1993), Hertz (1991) and Hertz (1992).

The success of these techniques, as it is always the case, has been limited by the drawbacks or disadvantages imbedded in the technique itself or disadvantages caused by applying the technique for a particular application. To overcome some of these limitations a hybridisation of metaheuristics techniques has been the subject of investigation by many researchers in recent years. Simulated annealing and tabu search ideas have been combined to obtain hybrid algorithms that obtained better results than either simulated annealing or tabu search alone. For more details, we refer to Osman (1993), Osman and Christofides (1994), Osman (1995) and Thangiah et.al (1994).

In this paper SA and TS have been used for the development of a hybrid algorithm to solve the TTP for each school at the UoW.

4.3. The Algorithm

Our algorithm can be summarised using the following four main steps:

Step 1 : Initialisation of solution and parameters

Step 2 : Evaluation of the solution

Step 3 : Generation of a new solution

Step 4 : Up_date of solution and parameters

The detailed description of these steps is given below:

4.3.1. Initialisation

The initial solution is obtained from the scheduler and recorded as the best solution. However, the following parameters have to be set:

- The initial temperature (T_s) and the final temperature (T_f).
- The maximum number of iterations (N_{max}) and the initial number of iterations per cooling schedule (N_s).
- The decrement cooling constant (C) between zero and one.
- The initial tabu tenure value.

From the experiments carried out using SA, we found that most of the best results are obtained at the middle range of the cooling schedule. Hence, the number of iterations per cooling schedule are increased and decreased gradually to allow more search for a better solution when the temperature is in that range.

4.3.2. Evaluation

The constraints penalties described in Section 3 were based on the schedulers advice and are given the following values:

Feasibility penalty = 9999.

High desirability penalty = 3.

Medium desirability = 1.

Other constraints penalties are valued according to the scheduler requirements at each school.

The solution is represented by a list of pairs (event,time-slot) indicating that a given event is occurring at a given time-slot.

By scanning the list of pairs (event,time-slot), we check the feasibility and desirability status of all events and add penalties accordingly. Consequently any event that is found undesirable or infeasible is recorded.

4.3.3. Generation of a New Solution

The new solution is obtained from the best recorded solution by removing the infeasibilities and undesirabilities recorded at the evaluation stage.

If the infeasibility or undesirability of an event is due to a lecturer constraint then this event is inserted in the first available time-slot. This insertion takes place after all the elements (groups,modules,rooms) associated with this event are checked for feasibility. Subsequently the infeasible or undesirable event is removed. On the other hand, if the infeasibilities and/or undesirabilities are due to a group, a module or a room then the infeasibilities and undesirabilities are removed using a similar procedure.

To avoid coming back to an infeasible or undesirable event a tabu list is introduced to record all the events that are found infeasible or undesirable and the list is up_dated after every iteration of the algorithm. The removed events are recorded in the tabu list and are forbidden for a number of iterations depending on the tabu tenure value.

4.3.4. Up_Date of Solution

The solution is up_dated if the cost of the new solution is lower than the cost of the best recorded one. In this case the tabu tenure value is increased and a new solution is generated.

We have observed that increasing the tabu tenure value by one each time the solution is improving produces better results than the alternative schemes. This is due to the fact that the more the solution is improving the longer the undesirable assignments are kept tabu.

If the new solution is worse than the best recorded solution then:

- If all the events are recorded tabu, then the event with the lowest tabu tenure value is released and a new solution is generated.
- If not all the events are tabu, then the current solution is accepted with a certain probability ($e^{-D/T}$) using the metropolis acceptance criterion, see for instance Dowsland (1993) and Brusco and Jacobs (1993). Subsequently, if the solution is rejected by the probability function, the temperature is increased (heating) according to the following equation:

$$T = T / (1 - \Gamma * T) \quad (1)$$

The heating process is reversed if the system is overheated according to the specified criteria.

- If the system is overheated or the specified number of solutions by cooling schedule is reached, the system is cooled by the following equation:

$$T = T / (1 + \beta * T) \quad (2)$$

The algorithm is terminated when the number of iterations exceeds a given maximum, the minimal temperature specified is reached or a completely desirable solution is obtained. Otherwise, the whole process is repeated after the values of Beta and Gamma are updated according to the following equations:

$$\beta = (1 - C) / (N * T * C) \quad (3)$$

$$\gamma = 2 * \beta / N \quad (4)$$

where

Beta is the reduction factor, Gamma is the increase factor, (T) is the current temperature, (N) is the number of iterations at the current cooling schedule and (C) is the decrement cooling constant defined at the initialisation stage.

5. Computational Results

Due to the TTP structure which can be easily represented in a form of

arrays and matrices, two and three dimensional arrays were used to represent the lecturers' timetable, the groups' timetable and the rooms' timetable.

For ease of programming, efficient handling of arrays and availability, the FORTRAN programming language was used.

The algorithm has been successfully used in solving the TTP in five different Schools. These Schools vary significantly in terms of the courses they offer, the nature of the modules taught and the particular requirements each module has. School 5, for instance, is the School of Languages; most of its courses consist of only a few compulsory modules and many optional ones. On the other hand, School 3, the School of Computer Science offers courses that have more compulsory modules, which require the use of computer laboratories.

The data supplied by the schedulers for the five Schools is summarised in Table 1 as follows:

Table 1: Number of lecturers, groups, modules, rooms and events.

	Lecturer	Group	Module	Event	Room
School 1	15	5	28	73	9
School 2	23	15	36	203	11
School 3	13	17	63	178	16
School 4	22	16	81	227	19
School 5	25	22	96	261	30

The following abbreviations are used for the computational results

shown in Table 2:

IC : Initial cost.

BC : Best cost.

IH : Initial number of high undesirabilities.

BH : Number of high undesirabilities at the best cost.

IM : Initial number of medium undesirabilities.

BM : Number of medium undesirabilities at the best cost.

IR : Initial number of rooms used.

BR : Number of rooms used at the best cost.

Cr : Convergence rate (BC/IC) for the hybrid algorithm.

Cr (SA): Convergence rate for the simulated annealing.

Cr (TS): Convergence rate for the tabu search.

Table 2: Computational Results for the five Schools.

	School 1	School 2	School 3	School 4	School 5
IC	40	105	212	173	206
BC	8	17	37	29	26
IH	9	22	47	33	37
BH	1	2	6	4	4
IM	13	39	71	74	95
BM	5	11	19	17	14
IR	9	11	16	19	30
BR	5	6	12	14	19
Cr	0.20	0.16	0.17	0.16	0.12
Cr (SA)	0.275	0.18	0.18	0.179	0.13
Cr (TS)	0.25	0.18	0.179	0.179	0.13

Since a large number of iterations did not seem to improve the solution significantly, despite the increase in computation time, the total number of iterations (Nmax) and the initial number of iterations

per cooling schedule (N_s) were set to 1000 and 2 respectively.

The other parameters were set based on the experiments carried out using SA and TS separately for the same problems. The initial temperature (T_s) and the final temperature (T_f) were set to 0.99 and 0.005 respectively, while the decrement cooling constant (C) was set to 0.69 and the initial tabu tenure to one.

It can be seen in Table 2 that the hybrid algorithm performed better than either simulated annealing or tabu search alone for all the problems.

The results obtained using the hybrid algorithm indicate that in all Schools the number of medium undesirabilities and the overall cost in the final solution are considerably smaller than the corresponding initial values. More importantly the number of the highly undesirable events and the number of rooms used have also been reduced. This can lead to a better utilisation of space, by releasing rooms that can be used for other purposes or by other Schools.

All the problems performed almost similarly except for the fifth School in which the convergence rate was even better though the problem size was larger and the number of initial undesirabilities was higher than all the other problems. This can only be explained by the number of rooms available which implies that the constraints for School 5 are less binding, i.e:offering more chances of finding better solutions.

6. Conclusions

In this paper we presented an algorithm for solving the TTP. The objective of this algorithm was to assist schedulers in designing a feasible and satisfactory timetable in a short time while using the resources efficiently.

The algorithm is a hybrid technique employing ideas of Simulated Annealing and Tabu Search. Its main differences to other approaches

consist in varying the tabu tenure and the cooling schedule.

In all the problems considered, the hybrid algorithm produced better results than either simulated annealing or tabu search alone.

Using our algorithm the scheduler can easily modify the timetable according to the requirements specified by the Accommodation Service, such as resolving conflicts between two or more schools over a request for a room. As we were not able to obtain a solution with a zero cost because of the tightness of the constraints, the scheduler is left with a choice of solutions and the associated lists of unsatisfied desirabilities. Then the scheduler can select the most appropriate solution.

The algorithm was used for all the Schools described earlier without any major alterations (only the sizes of the arrays and some minor constraints and penalties were changed), ensuring portability between the different Schools.

All the results were obtained using a 486 machine running at 33 Mhz. The running times for all the problems were under 20 minutes. Given that timetabling is an activity which takes place twice a year (once each semester), this was considered perfectly acceptable.

The algorithm performed well in all cases in the sense that it reduced undesirabilities and released space for alternative usage. Consequently, we believe that our technique is flexible, versatile and can be used to produce satisfactory solutions to real timetabling problems.

Acknowledgements

We wish to thank Dr Ioannis Giannikos and the anonymous referees for their helpful comments. We are also grateful to the University of Westminster for providing us with the financial support to carry out this work.

7. References

- H.Abada and E.El-Darzi, Solving the Timetabling Problem using Simulated Annealing, Paper under preparation, University of Westminster, England, (1995a).
- H.Abada and E.El-Darzi, Solving the Timetabling Problem using Tabu search, Paper under preparation, University of Westminster, England (1995b).
- D.Abramson, Constructing School Timetables using Simulated Annealing: Sequential and Parallel Algorithms, *Management Science* 37(1991)1.
- D.Abramson and H.Dang, School Timetables: A Case Study in Simulated Annealing: in *Lecture Notes in Economics and Mathematical Systems*, Ed. Rene V.V.Vidal 396(1993)104.
- R.Alvarez-Valdes,G.Martin and J.M.Tamarit, Constructing Good Solution for School Timetabling Problem, Technical Report, Department of Statistics and O.R, University of Valencia, Spain, (1993).
- M.J.Brusco and L.W.Jacobs, A Simulated Annealing Approach to the Solution of Flexible Labour Scheduling Problems, *Journal of Operational Research Society* 44(1993)12.
- M.W.Carter and A.Tovey, When is the Classroom Assignment Problem Hard ?, *Operations Research* 40(1992)1.
- K.A.Dowsland, A Timetabling Problem in which Clashes are Inevitable, *Journal of Operational Research Society* 41(1990)10.
- K.A.Dowsland, Simulated Annealing, in: *Modern Heuristics for Combinatorial Problems*, Ed.C.R.Reeves,(Blackwell Oxford,1993).
- F.Glover and M.Laguna, Tabu Search, in: *Modern Heuristics for Combinatorial Problems*, Ed.C.R.Reeves,(Blackwell Oxford,1993).
- A.Hertz, Tabu Search for Large Scale Timetabling Problem, *European Journal of Operational Research* 54(1991)39.
- A.Hertz, Finding a Feasible Course Schedule using Tabu Search, *Discrete Applied Mathematics* 35(1992)255.
- I.H.Osman, Metastrategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem, *Annals of Operations Research* 41(1993)421.

- I.H.Osman and N.Chrisofides, Capacitated Clustering Problems by Hybrid Simulated Annealing and Tabu Search, *International Transactions in Operational Research* 1(1994)317.
- I.H.Osman, Heuristics for the Generalised Assignment Problem: Simulated Annealing and Tabu Search Approaches, Forthcoming in *OR Spektrum* 17(1995).
- S.M.Thangiah, I.H.Osman and T.Sun, Metaheuristics for The Vehicle Routing Problems with Time Windows, Internal Report, Institute of Mathematics and Statistics, University of Kent, England (1994).

10

Complex Sequencing Problems and Local Search Heuristics

Peter Brucker and Johann Hurink

Universität Osnabrück, FB Mathematik/Informatik
D-49069 Osnabrück, Germany

E-mail: pbrucker@dosuni1.rz.uni-osnabrueck.de
johann@mathematik.uni-osnabrueck.de

Abstract: Many problems can be formulated as complex sequencing problems. We will present problems in flexible manufacturing that have such a formulation and apply local search methods like iterative improvement, simulated annealing and tabu search to solve these problems. Computational results are reported.

Key Words: Assembling, batching, flexible manufacturing, sequencing problem, simulated annealing, tabu search, tooling.

1. Introduction

Local search heuristics like iterative improvement, simulated annealing, and tabu search are popular procedures for providing solutions for discrete optimization problems. Clearly, a good method should take into account structural properties of the problem. Usually this is accomplished by choosing a neighborhood which is based on specific properties. Also specific organization of tabu list may reflect structural properties of the problem.

In this work we will consider optimization problems which are combinations of a sequencing problem and another type of optimization problem P , e.g. partitioning problem or location problem. Therefore, for problems of such type the solution sets consists of all possible combinations of a sequence and a description of a so-

lution of P . More precisely, all solutions of the combined problem are given by a set $\{(\pi, r) | \pi \in S_n; r \in R\}$, where S_n is the set of all permutations of a set with n elements and R defines all possible instances of P . Denote by $f(\pi, r)$ the corresponding objective value. For our approach to solve such combined problems we will consider only the set of sequences as solutions. For a given sequence π we solve the optimization problem $\min\{f(\pi, r) | r \in R\}$ and do local search on the set of sequences. The corresponding optimization problems are solved by polynomial algorithms or sophisticated heuristics that take into account structural properties of the problem. This approach reduces the cardinality of the search space drastically. Furthermore, it is possible to choose relatively easy and general neighborhoods for the local search on the set of sequences since structural properties are already incorporated in the solution of the optimization problem. The prize for this advantage is a more complex objective value for a given solution (sequence).

In Section 2 we will give a formal definition of complex sequencing problems and describe the specific methods we used to solve these problems. We apply these methods to bin location and assembly sequencing (Section 3), switching tools on a flexible machine (Section 4), and batching problems (Section 5). Computational results show that these methods yield very good results.

2. Complex Sequencing Problems

A complex sequencing problem may be formulated as follows. Let X be a subset of all permutations of a finite set. Associated with each permutation π in X there is an optimization problem $OPT(\pi)$. Let $f(\pi)$ be the optimal solution value of $OPT(\pi)$. Find a permutation π^* which optimizes f over all permutations in X . Usually $OPT(\pi)$ is a problem which can be solved efficiently (e.g. a linear program, a shortest path problem, etc.)

To solve complex sequencing problems different local search heuristics are considered. To apply these heuristics to a specific problem, an appropriate neighborhood structure on X has to be defined. As mentioned in the introduction, we consider simple neighborhood structures which are given by operator sets $OP(\pi)$ defined for permutations π . An operator $op \in OP(\pi)$ associates with the permutation π a new permutation $op(\pi)$. $\{op(\pi) | op \in OP(\pi)\}$ is called

the neighbor set of π . The neighborhood structures we use are given by the following operators:

- shift operators

A shift operator s_{ij} , applied to a permutation $\pi = (\pi_1, \dots, \pi_n)$ moves the element in π that is in position i into position j . Furthermore, if $i < j$ ($j < i$) the elements in positions $i + 1, \dots, j$ ($j, \dots, i - 1$) are shifted by one position to the left (right). The corresponding neighborhood is called shift neighborhood.

- interchange operators

An interchange operator c_{ij} , applied to a permutation π , interchanges the elements at positions i and j . The corresponding neighborhood is called interchange neighborhood.

- adjacent pair interchange operators

An adjacent pair interchange operator c_i interchanges elements at positions i and $i + 1$ ($i = 1, \dots, n - 1$). The corresponding neighborhood is called adjacent pair interchange neighborhood.

Using these neighborhoods, the following three local search heuristics are applied to complex sequencing problems:

- Iterative Improvement (see Papadimitriou and Steiglitz (1982))
In each iteration the best neighbored solution is selected as next solution.

- Simulated Annealing

For the simulated annealing method we haven chosen the simple cooling scheme described by Laarhoven and Aarts (1987) with instance specific initial and final temperature.

- Tabu Search (see, e.g. Glover (1989,1990))

For the bin location problem (Section 3) and the tool switching problem (Section 4) we applied a static tabu list and for the batching problem (Section 5) a dynamic tabu list. Instead of choosing a fixed tabu list length, we have chosen a tabu list length depending on the size of the neighborhoods and/or the

size of the instances. The method will stop if after a specified number of iterations no improvement of the best objective value has occurred.

The entries of the tabu list are chosen in the following way: For the tool switching problem we make sure that cycles of length smaller or equal to the size of the tabu list will not occur. This results at least for the first two neighborhoods to rather restrictive tabu conditions. Therefore, aspiration conditions are incorporated. For the other two problems the tabu conditions are less restrictive. If an operation is applied, its characteristics will be inserted in the tabu list and it will be forbidden to apply the operation again. This will not prevent that a solution is visited again within a number of iterations that is smaller or equal to the size of the tabu list. However, the second time we have to leave the solution with another operation as the first time, since this operation is still in the tabu list. Therefore, we will not get stuck in cycles of length smaller or equal to the size of the tabu list.

The concrete parameters for the different methods have been specified after some preliminary testing.

In the next sections we will apply combinations of iterative improvement (*II*), simulated annealing (*SA*), tabu search (*TS*) with shift neighborhood (*S*), interchange neighborhood (*I*), adjacent pair interchange neighborhood (*API*) to complex sequencing problems derived from flexible manufacturing. Performances of different combinations are compared.

3. Bin locating and assembly sequencing

In connection with printed circuit assembly Foulds and Hamacher (1993) consider the following problem. On a two dimensional rectangular board n parts $1, \dots, n$ of different type have to be inserted, one by one, at specified locations P_1, \dots, P_n by a robot. The parts can be inserted in an arbitrary order. They are stored in bins B_1, \dots, B_m around the board. For each bin B_j there is a feasible region R_j of locations, where the bin may be placed ($j = 1, \dots, m$). Furthermore, each bin contains only parts of one type. We denote by r_i the bin in which part i is stored ($i = 1, \dots, n$).

The problem is to find a sequence π in which the parts will be inserted and feasible locations X_1, \dots, X_m for the bins such that the total distance traveled by the robot is minimized.

If we denote by $d(a, b)$ the distance between the points a and b then the length of a robot tour for a sequence π is given by

$$\sum_{i=1}^n \left(d(X_{r_{\pi(i)}}, P_{\pi(i)}) + d(P_{\pi(i)}, X_{r_{\pi(i+1)}}) \right), \quad (1)$$

where $\pi(n+1) = \pi(1)$.

Thus, $OPT(\pi)$ is the problem of finding locations $X_j \in R_j$, $j = 1, \dots, m$, such that (1) is minimized.

It is clear that the placement of bin B_i is independent of the placement of bin B_j for $i \neq j$. Therefore, for a given robot tour π one has to find for each $j = 1, \dots, m$ a placement $X_j \in R_j$ for bin B_j which minimizes

$$f_j(X) = \sum_{\substack{i=1 \\ r_{\pi(i+1)}=j}}^n (d(P_{\pi(i)}, X) + d(X, P_{\pi(i+1)})). \quad (2)$$

These location problems for the bins can be solved by iterative methods, based on contour lines (see Foulds and Hamacher (1993)).

We denote the problem of finding a location X_j which minimizes (2) by $OPT_j(\pi)$. $OPT(\pi)$ decomposes into m independent problems $OPT_j(\pi)$, $j = 1, \dots, m$. For speeding up the heuristics, we exploited the fact that when moving from one permutation to the next usually only a few problems $OPT_\sigma(\pi)$ are changing. More precisely, for the neighborhoods (*API*) and (*S*) at most 3 location problems and for the neighborhood (*I*) at most 4 location problems change. Furthermore, for the methods (*II*) and (*TS*) we used the results for the changed location problems for many neighbors. The computational amount of work has been reduced considerably by taking into account these facts.

Methods (*II*), (*SA*), and (*TS*) have been implemented in connection with the neighborhood (*S*), (*I*), and (*API*). For testing these methods, we generated 15 instances with $n \in \{10, 20, 30, 40, 50\}$

and $m \in \{10, 20, \dots, n\}$. The positions on the board are chosen randomly and the assignment of types to the parts is done by a uniform distribution. Furthermore, the feasible regions for all bins are equal to the same region. This region is defined as the plane without a rectangle that contains the board.

The computational experiments have shown that the neighborhood (*API*) does not lead to good results. Therefore, we will only present results for the neighborhoods (*S*) and (*I*). Since the neighborhoods (*S*) and (*I*) in combination with (*SA*) provided very similar results, we present only the results for the combination of (*SA*) with (*I*) ((*I*) was in average about 0.1% better than (*S*)).

For constructing initial solutions we used a greedy type heuristic. This method first calculates the best positions of the bins if only the travel distances from the bins to the part locations are considered (i.e. the travel distances from the part locations to the bins are ignored). Afterwards for these bin positions a good insertion sequence is constructed by a nearest neighbor heuristic for the TSP. Since the nearest neighbor heuristic depends on the starting point, we may use this greedy type heuristic to calculate different initial sequences for the bin location problem.

In Table 1 the computational results are summarized. Each entry represents the average relative error between the heuristic value and a lower bound over 3 runs (we generated 3 different initial solutions and applied the local search heuristics). The lower bound has the property that its quality increases with decreasing $\frac{n}{m}$ value (see Hollmann (1993)).

Concerning the quality of the solutions, the results can be summarized as follows. For (*II*) and (*TS*) the shift neighborhood (*S*) gave on average slightly better results than the interchange neighborhood (*I*). However, the differences between (*S*) and (*I*) are rather small and as mentioned above in connection with (*SA*) the neighborhood (*I*) is even competitive with (*S*). We can state that tabu search (*TS*) leads to the best results, followed by simulated annealing (*SA*). Iterative improvement (*II*) gave the worst results.

However, if we compare the running times of the different methods we get the opposite relation. (*II*) always finishes within 5 minutes, whereas (*SA*) (with 100000 iterations) uses up to 25 minutes

and (TS) up to 5 hours for the hardest problem ($n = 50$, $m = 10$) on a SUN SPARC-station 10/20. In an additional series we tested whether (SA) could reach the same quality as (TS) if we spend as much running time as for (TS). The additional time led to an improvement of the results of (SA), but the quality of the (TS) could not be reached.

Table 1: Average relative error of the heuristics

n	m	greedy	$II + S$	$II + I$	$SA + I$	$TS + S$	$TS + I$	<i>alter</i>
10	10	12.32	3.61	4.79	3.03	3.03	3.03	8.53
20	10	6.21	2.45	2.99	1.95	1.38	1.68	3.29
20	20	5.83	2.41	2.83	0.67	0.67	0.67	2.97
30	10	8.20	2.45	2.84	2.40	2.33	2.38	2.85
30	20	6.48	2.20	2.85	1.51	1.04	1.29	1.76
30	30	4.63	1.83	1.74	0.76	0.85	1.33	1.73
40	10	6.61	3.59	3.55	3.50	3.45	3.41	3.50
40	20	3.57	0.76	0.85	0.86	0.28	0.51	0.57
40	30	4.10	0.40	0.55	0.56	0.18	0.23	1.18
40	40	2.06	0.25	0.25	0.44	0.13	0.25	0.17
50	10	11.30	3.82	4.66	3.93	3.29	3.53	5.66
50	20	5.11	1.29	2.09	1.54	1.20	1.39	2.14
50	30	4.44	1.00	1.36	0.58	0.16	0.36	0.46
50	40	3.88	0.82	0.93	0.71	0.34	0.64	0.43
50	50	2.79	1.07	1.42	0.55	0.28	0.72	0.24

We also applied a heuristic given by Foulds and Hamacher (1993) that alters between finding optimal locations for a given sequence and finding an optimal sequence for given locations (see entry *alter* in Table 1). This heuristic is rather quick (running times at most 7 minutes), but due to its better solution quality TS clearly outperformed this heuristic.

4. Tool switches on a flexible machine (Crama et al. (1991))

Suppose that n jobs have to be successively processed, one at a time, on a single flexible machine. Each job requires a subset of tools, which have to be placed in a tool magazine of the machine before the job can be processed. The tool magazine has a limited capacity, i.e. it can accommodate at most C tools, each of which fits in one slot of the magazine. As the number of tools needed to produce all jobs is generally larger than C it is sometimes necessary

to change tools between two jobs in a sequence. When this occurs, some tools have to be removed from the tool magazine and are replaced by a same number of tools retrieved from a storage area. We assume that the tool magazine is loaded up to its capacity C . The process of replacing tools is called a tool switch. The following tool switching problem is then of practical relevance: determine a job sequence and an associated sequence of loadings for the tool magazine, such that all tools required by the j -th job are present in the j -th loading, and the total number of tool switches is minimized.

For a given sequence π we have the tool loading problem $OPT(\pi)$: Find an optimal tool loading for π .

Tang and Denardo (1988) show that the so-called Keep Tool Needed Soonest (*KTNS*) policy solves the loading problem $OPT(\pi)$. A *KTNS*-policy prescribes that, whenever a situation occurs where a tool needed is not in the magazine, then those tools which are needed soonest for a future job should be removed last. Similarly, at the beginning the magazine is loaded up to its capacity with those tools which are needed earliest.

Methods (*II*), (*SA*) and (*TS*) have been implemented in connection with all three neighborhoods. Similar to Crama et al. (1991) we considered different problem types as test instances. Each type is characterized by the number n of jobs, by the total number t of tools, by two parameters *min* and *max*, and by the capacity C of the tool magazine. For each job the number of needed tools is chosen randomly from the interval $[min, max]$ and the assignment of tools to jobs is also done randomly. Table 2 describes the different problem types that were considered.

Table 2: Problem types

n	t	<i>min</i>	<i>max</i>	C
10	10	2	4	{4, 5, 6, 7}
15	20	2	6	{6, 8, 10, 12}
30	40	5	15	{15, 17, 20, 25}
40	60	7	20	{20, 22, 25, 30}

For each problem type (n, t, min, max) we generated 10 job-tool assignments and combined these instances with all values of C given in Table 2. Methods (*SA*) and (*TS*) have been applied in

connection with all three neighborhoods on a 486/33MHz.

Since we detected no influence of the parameter C on the quality of the neighborhoods and the methods, we will present only results for the smallest C -value in each row of Table 2. Furthermore, each entry in the following tables will be an average value over all 10 generated job-tool assignments for each problem type.

First we compare the three different neighborhoods. Since the differences between the neighborhoods were similar for all methods, in Table 3 we present only the results of (TS) with the three neighborhoods.

Table 3: TS results

(n, t, C)	API	I	S
(10, 10, 4)	11.0	9.3	9.3
(20, 15, 6)	24.7	21.1	20.7
(40, 30, 15)	127.7	103.9	106.2
(60, 40, 20)	242.7	204.1	205.4

Again, the two neighborhoods (S) and (I) are close together and clearly outperform (API). However, this time (I) is slightly better than (S).

In Table 4 we present results reached by the heuristics using the best neighborhood (these values were always reached in connection with neighborhood (S) or (I)). Again, each entry represents the average value over all 10 job-tool assignments. The row OBJ contains the value of the objective function and the row CPU shows the corresponding computational times (in seconds).

Table 4: Average results of the heuristics

(n, t, C)		II	SA	TS	SA/TL
(10, 10, 4)	OBJ	10.1	10.1	9.3	10.0
	CPU	0.2	0.5	0.8	0.3
(20, 15, 6)	OBJ	22.0	21.5	20.7	20.5
	CPU	2.8	1.6	28.3	3.3
(40, 30, 15)	OBJ	107.5	108.5	103.9	103.6
	CPU	163.7	23.5	888.2	62.9
(60, 40, 20)	OBJ	209.2	212.7	204.1	203.2
	CPU	624.8	91.7	5262.7	318.1

As before, (TS) gave the best results, whereas (SA) and (II) led to similar results. However, to reach the good quality with (TS) we have to spend much more computational time than for the other two methods. Again, we tested the influence of a larger number of iteration for (SA) . Now, (SA) produces better results than (II) , but the quality of (TS) could not be reached.

Since for SA we have the advantage of small computational times and for (TS) we have the advantage of good results, we tried to combine elements of these two methods. This was done by incorporating in SA a tabu list, i.e. in each step the set of neighbors is restricted by a tabu list. We denote this method by (SA/TL) . The results of this method are also presented in Table 4. The computational times for (SA/TL) range slightly above those for (SA) . However, the quality of the solutions improved considerably compared with SA and is competitive with the quality of (TS) . Therefore, this approach seems to be a good compromise between computational effort and solution quality.

Since no efficient exact solution methods for the tool switching problem are known, we calculated lower bounds suggested by Crama et al. (1991) to determine the quality of the heuristic solutions. However, the gap between lower bounds and best heuristic solution was quite large for all considered instances. Therefore, we solved some of the small instances (10-10) to optimality by brute force enumeration. In all cases the optimal value was equal to the best heuristic solution and therefore far above the lower bounds. This indicates that the heuristics perform quite well for the tool switching problem.

5. One machine batching problems

For a one machine batching problem, n jobs J_1, \dots, J_n with processing times p_1, \dots, p_n have to be scheduled on one machine. Between the jobs precedence relations are given. The jobs are processed in batches. A batch is a set of jobs which are processed jointly. The length of a batch is equal to the sum of the processing times of its jobs. The jobs of a batch are available if the batch is finished. Thus, the finish time C_i of job J_i coincides with the completion time of its batch, i.e. all jobs of a batch have the same finish time. The production of a batch requires a machine setup of s time units. This

setup time is independent of the sequence of the batches and the number of jobs in a batch.

The problem is to find a sequence of the jobs which is compatible with the precedence relations and a partitioning of this sequence into batches such that the sum of weighted flow times $\sum_{i=1}^n \alpha_i C_i$ is minimized. The values α_i are nonnegative job weights ($i = 1, \dots, n$). Thus, a solution can be characterized by a sequence of the jobs and a partition of this sequence into batches.

Given a sequence π of all jobs which is compatible with the precedence relations $OPT(\pi)$ is the problem of finding a partition of π into batches which minimizes $\sum_{i=1}^n \alpha_i C_i$. $OPT(\pi)$ can be solved as follows. Let J_1, \dots, J_n be an arbitrary but fixed job sequence. A partitioning of this job sequence into batches has the form

$$J_{i_1}, \dots, J_{i_2-1} \mid J_{i_2}, \dots, J_{i_3-1} \mid \dots \mid J_{i_{k-1}}, \dots, J_{i_k-1}.$$

It can be described by a sequence $1 = i_1 < i_2 < \dots < i_k = n + 1$. The corresponding weighted sum of flow times can be written in the form

$$\sum_{\nu=1}^{k-1} c_{i_\nu i_{\nu+1}}$$

where

$$c_{ij} = \left(\sum_{k=i}^n \alpha_k \right) \left(s + \sum_{k=i}^{j-1} p_k \right) \quad (3)$$

is the contribution of batch J_i, \dots, J_{j-1} to the weighted sum of flow times.

In order to solve the optimization problem $OPT(\pi)$ we consider a network $G = (V, E, c)$ with $V = \{1, \dots, n + 1\}$ and $E = \{(i, j) \mid i, j = 1, \dots, n + 1; i < j\}$. The nodes represent the jobs (n+1 is a dummy job) and an edge (i, j) represents a batch J_i, \dots, J_{j-1} . The length of an edge $(i, j) \in E$ is given by costs c_{ij} induced by the corresponding batch.

Each path from 1 to $n + 1$ in G now represents a partition of the job sequence into batches and vice versa. Therefore $OPT(\pi)$ can be solved by finding a shortest path from 1 to $n + 1$ in G . Due to the structure of the graph (only edges (i, j) with $i < j$) and the costs (3) we need not to construct the graph explicitly, but can use

some dominance criteria to construct iteratively the shortest paths from 1 to i , $i = 1, \dots, n+1$ in G . This results in an $O(n)$ algorithm to solve the problem $OPT(\pi)$ (see Albers and Brucker (1993)).

We have investigated the three local search procedures in combination with each of the three neighborhoods. For generating test instances we have chosen the following values for the variables:

$$\begin{aligned} n &\in \{20, 50, 100, 200\} \\ p_i &\in [1, 100] \text{ randomly} \\ \alpha_i &\in [1, 30] \text{ randomly} \\ d &\in \{0, 0.1, 0.3, 0.5\} \\ s &\in \{1, 10, 25, 50, 100\}, \end{aligned}$$

where d denotes the density of the precedence constraints.

We will present two series of results. Table 5 contains the results for two fixed parameters of s and d ($s = 25$, $d = 0.1$) and different values of n and Table 6 contains the results for two fixed parameters of s and n ($s = 25$, $n = 100$) and different values of d . The data given in rows OBJ are values $\frac{1}{n} \sum_{i=1}^n \alpha_i C_i$. To compare the methods (II) , (SA) , and (TS) we listed the best value taken over all neighborhoods. To compare the neighborhoods we listed the best value taken over all heuristics. In rows CPU the corresponding computation times (in sec.) on a SUN SPARC-station 10/512 are listed.

Concerning the neighborhoods, (S) and (I) lead to much better results than (API) . The last neighborhood is only successful in connection with (SA) for instances with up to 50 jobs. Between the neighborhoods (S) and (I) only small differences occur, whereby (S) has a small advantage over (I) . However, the computational times needed for (S) in connection with (II) and (TS) are twice the computational times needed for (I) . We have replaced S by the neighborhood S_k in which the set of shifts s_{ij} has been restricted to those where the distance $|i - j|$ between position i and j is bounded by k . Tests have shown that $k \approx n/5$ leads approximately to the same quality of solutions as S . However, the computational effort is reduced by a factor 2.

Table 5: Results for $s = 25$, $d = 0.1$

n		20	50	100	200
<i>Init</i>	<i>OBJ</i>	7557	26063	43027	85673
<i>II</i>	<i>OBJ</i>	4450	17610	24061	46510
	<i>CPU</i>	0.7	26	352	6526
<i>SA</i>	<i>OBJ</i>	4450	17732	25297	49304
	<i>CPU</i>	2.1	4.3	8.8	17
<i>TS</i>	<i>OBJ</i>	4450	17610	24061	46510
	<i>CPU</i>	1.0	48	734	12730
<i>API</i>	<i>OBJ</i>	4450	17732	36718	81868
	<i>CPU</i>	2.1	4.3	8.2	16
<i>S</i>	<i>OBJ</i>	4450	17644	24061	46515
	<i>CPU</i>	0.7	51	876	16105
<i>I</i>	<i>OBJ</i>	4450	17610	24061	46510
	<i>CPU</i>	0.5	26	352	6526

Table 6: Results for $s = 25$, $n = 100$

d		0.0	0.1	0.3	0.5
<i>Init</i>	<i>OBJ</i>	43027	42176	41524	40153
<i>II</i>	<i>OBJ</i>	24061	31161	34603	35301
	<i>CPU</i>	352	169	124	90
<i>SA</i>	<i>OBJ</i>	25297	31295	36052	35382
	<i>CPU</i>	8.8	9.1	9.1	9.3
<i>TS</i>	<i>OBJ</i>	24061	30967	34447	34853
	<i>CPU</i>	734	1400	621	878
<i>API</i>	<i>OBJ</i>	36718	36007	37402	36661
	<i>CPU</i>	8.2	8.2	8.2	8.1
<i>S</i>	<i>OBJ</i>	24601	30967	34447	34853
	<i>CPU</i>	876	1400	621	878
<i>I</i>	<i>OBJ</i>	24601	31228	36052	35382
	<i>CPU</i>	352	57	9.1	9.3

If we compare the quality of the solutions for the three heuristics, we find (*TS*) as a clear winner, followed by (*II*) and last (*SA*). However, the chosen number of iterations (10000) for (*SA*) leads to rather small computational times in comparison with the other two methods. If we enlarge this number, we get slightly better results than for (*II*) but not as good results as for (*TS*) with comparable computational times.

Summarizing, if we are willing to spend much computational time, (TS) in connection with (I) or ($S_{n/5}$) will be the best choice. Otherwise, (SA) in connection with neighborhoods (S) or (I) is preferable.

Since for the batching problem no efficient exact methods and no good lower bounds are known, we applied the heuristics to instances of special cases, for which polynomial algorithms are known. We generated some instances with 100 jobs. In all cases the best heuristic solution was equal to the optimal value and combinations of methods and neighborhoods showed the same behavior as for general instances. This, and the large improvements of initial solutions produced by the heuristics indicate that the quality of the solutions is quite good.

5. Concluding remarks

We solved complex sequencing problems by calculating for each sequence a corresponding optimization problem and applying local search procedures on the set of sequences. Due to the fact that problem specific properties are taken in account by methods for solving the corresponding optimization problems this approach yields good results even for very general neighborhoods based on shift operators and interchange operators. The rather simple neighborhood based on adjacent pair interchange operators did not produce good results. A reason for this might be that (API) allows only marginal changes of the sequence and therefore a diversification of the solutions will not take place.

A similar decomposition approach has been used by Brucker, Hurink, Werner (1994,1995) to solve other NP -hard scheduling problems. This approach is based on polynomial algorithms for reaching a local optimal solution starting from some arbitrary solution. Using such polynomial algorithms a neighborhood on sets of local optima can be defined by operations which first move from a local optimum to a solution s which is not locally optimal and in a second step move from s to a new locally optimal solution.

The quality of our algorithms depend on the quality of the reoptimization procedures. Recently powerful reoptimization procedures have been developed for the batching problems discussed in Section 5.

Acknowledgment We acknowledge the computational work done by Stefan Hollmann, Vera Ruhe, and Christian Ziemer in connection with their Diploma Thesis.

6. References

- S. Albers, P. Brucker, The complexity of one-machine batching problems, *Discrete Applied Mathematics* 47 (1993), 87 - 107.
- P. Brucker, J. Hurink, F. Werner, Improving local search heuristics for some scheduling problems I, to appear in: *Discrete Applied Mathematics* (1994).
- P. Brucker, J. Hurink, F. Werner, Improving local search heuristics for some scheduling problems II, to appear in: *Discrete Applied Mathematics* (1995).
- Y. Crama, A.W.J. Kolen, A.G. Oerlemans, F.C.R. Spieksma, Minimizing the number of tool switches on a flexible machine, *Rutcor Research Report* (1991), 11-91.
- L.R. Foulds, H.W. Hamacher, Optimal bin location and sequencing in printed circuit board assemble, *European Journal of Operation Research* 66 (1993), 270 - 290.
- F. Glover, Tabu Search, Part I, *ORSA Journal on Computing* 1 (1989), 190-206.
- F. Glover, Tabu Search, Part II, *ORSA Journal on Computing* 2 (1990), 4-32.
- S. Hollmann, Verfahren der Lokalen Suche zur Lösung eines kombinierten Standort- und Reihenfolgeproblems, Diplomarbeit, Fachbereich Mathematik/Informatik, Universität Osnabrück (1993).
- P. J. M. Laarhoven, E. H. L. Aarts, Simulated Annealing: Theory and Applications, (D. Reidel Publishing Company, Dordrecht, Holland, 1987).
- C. H. Papadimitriou, K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, (Prentice Hall, New Jersey, 1982).

- V. Ruhe, Lokale Suchverfahren für Einmaschinenbatchingprobleme, Diplomarbeit, Fachbereich Mathematik/Informatik, Universität Osnabrück (1994).
- C. S. Tang, E.V. Denardo, Models arising from a flexible manufacturing machine, Part 1: Minimizing of the number of tool switches, Operations Research 36 (1988), 767 - 777.
- C. Ziemer, Minimierung der Anzahl von Werkzeugwechseln einer flexiblen Maschine, Diplomarbeit, Fachbereich Mathematik/ Informatik, Universität Osnabrück (1995).

Heuristic Algorithms for Single Processor Scheduling with Earliness and Flow Time Penalties

Mauro Dell'Amico

Dipartimento di Elettronica e Informazione

Politecnico di Milano

E-mail: dellamic@elet.polimi.it

Silvano Martello, Daniele Vigo

Dipartimento di Elettronica, Informatica e Sistemistica

Università di Bologna

E-mail: silvano@boder1.cineca.it

Abstract:

We consider the problem of scheduling a set of n tasks on a single processor. Each task has a processing time, a deadline, a flow time penalty and an earliness penalty. The objective is to minimize the total cost incurred by the penalties. The problem is NP-hard in the strong sense. Exact enumerative algorithms from the literature can solve random instances with $n \leq 50$. We study a tabu search approach to the approximate solution of the problem and show, through computational experiments, that instance with $n = 300$ are effectively solved with short computing times.

Key Words: Earliness, just-in-time, scheduling, tabu search.

1. Introduction

The Just-In-Time (*JIT*) approach has originated a strong impact in

production systems, introducing, among others, the notion of inventory reduction. This imposes not to accept the ordered goods from the manufacturer before their associated due dates. Thus products that are completed early must be stored until the due date by the manufacturer which, therefore, incurs in additional inventory cost.

The need to avoid task earliness has produced the development, in recent years, of scheduling models with earliness penalties, able to capture the scheduling dimension of the *JIT* approach (see, e.g., Baker and Scudder (1990) for a broad review of models and algorithms). A natural way to prevent task earliness is to allow processor idle time. Nevertheless this hinders the widely used objective of the reduction of raw materials storage costs, capital carrying costs, and processor use maximization. In scheduling theory these needs are generally modeled with flow time penalties.

A more general approach is to combine the contrasting objectives above by determining a schedule which minimizes the so-called *total inventory cost*, a weighted sum of work-in-progress and finished product storage costs. The total inventory cost of a task is then defined as the sum of the task completion time, multiplied by its flow time penalty, plus the task earliness, multiplied by its earliness penalty. The result is a *non regular* performance measure, i.e., an objective function which can decrease when the completion times increase.

In this paper we study a scheduling problem on a single processor where each task has an associated deadline and the objective is to minimize the total inventory cost.

1. The problem

Given a processor which can handle at most one task at a time, and a set $T = \{T_1, \dots, T_n\}$ of n tasks (available at time zero) with associated *processing times* p_1, \dots, p_n , *deadlines* d_1, \dots, d_n , *flow time penalties* $\alpha_1, \dots, \alpha_n$ and *earliness penalties* β_1, \dots, β_n , we

consider the problem (called II in the following) of determining a task schedule that minimizes the total cost incurred by the penalties, while preserving the deadline requirements of each task. The cost associated with each task T_j is equal to its *completion time*, C_j , multiplied by the flow time penalty, plus its *earliness*, $E_j = d_j - C_j$, multiplied by the earliness penalty.

Problem II is strongly NP-hard, since it is a generalization of the single processor scheduling problem calling for the minimum weighted flow time sequence with no tardy task ($1|d_j|\sum \alpha_j C_j$) which is known to be NP-hard in the strong sense (see Lenstra, Rinnooy Kan and Brucker (1977)).

Because of the deadline requirements, an instance of II does not necessarily have a feasible solution. Feasibility can be checked in $O(n \log n)$ time, since a well known necessary and sufficient condition requires feasibility of the schedule produced by the following greedy approach (known as the *earliest due date (EDD)* rule, Jackson (1955)):

Step 1 reorder the tasks according to nondecreasing deadlines;

Step 2 $C_1 := p_1$; **for** $j := 2$ **to** n **do** $C_j := C_{j-1} + p_j$.

It is convenient to transform the problem so that the pair of flow time and earliness penalties associated with each task is replaced by the overall *penalty*, $w_j = \alpha_j - \beta_j$. To this end, the objective function can be written as:

$$\begin{aligned} z^*(\text{II}) &= \min \sum_{j=1}^n (\alpha_j C_j + \beta_j (d_j - C_j)) \\ &= \sum_{j=1}^n \beta_j d_j + \min \sum_{j=1}^n w_j C_j = \sum_{j=1}^n \beta_j d_j + z^*(P), \end{aligned} \quad (1)$$

Problem P calls for the minimization of the total weighted completion time with no tardy task, with positive and negative weights, w_j . The sign of the overall penalty determines the behavior of the tasks in an optimal schedule: those with $w_j < 0$ require to be pro-

cessed as late as possible (i.e., to be completed as close as possible to their deadline), whereas those with $w_j > 0$ must be scheduled as early as possible. The remainder of the paper is concerned with the solution of problem P .

A dynamic programming algorithm for the exact solution of this problem has been developed by Bard, Venkatraman and Feo (1993). A more effective mixed approach (branch-and-bound/ dynamic programming) has been recently proposed by Dell'Amico, Martello and Vigo (1995). The same paper also gives a (traditional) approximation algorithm, *JOIN*, which determines, in $O(n^2)$ time, a feasible solution to P by recombining split tasks obtained from a relaxation of P ; the resulting solution is then improved by optimally determining idle times through the $O(n)$ procedure described in Feo, Venkatraman and Bard (1991) (see also Dell'Amico (1994) for a more general approach). Feo, Venkatraman and Bard (1991) have presented a heuristic algorithm based on a *greedy randomized adaptive search procedure (GRASP)*.

Throughout the paper it is assumed that all input data (processing times, deadlines, flow time and earliness penalties) are positive integers. Given a feasible schedule C_1, \dots, C_n , we assume that task T_j is processed in time interval $]C_j - p_j, C_j]$.

3. Local search

In the next sections we describe a tabu search approach to the approximate solution of P , consisting of a selective exploration of the set \mathcal{S} of feasible solutions to the problem. For any $s \in \mathcal{S}$, let $z(s)$ denote the corresponding objective function value, and $N(s) : \mathcal{S} \rightarrow 2^{\mathcal{S}}$ a *neighborhood* of s , consisting of a set of feasible solutions obtained from s by modifying the schedule according to some specified *move*. At each iteration, given the current solution s , a new solution $s' \in N(s)$ is selected according to some criterion. If the criterion is to select s' such that $z(s') = \min_{r \in N(s)} \{z(r) : z(r) < z(s)\}$,

we have a traditional heuristic which terminates as soon as a *local minimum* is encountered. If instead s' is selected without requiring that $z(s') < z(s)$, we have a so-called *hill-climbing* heuristic, which can escape from local optima and determine better solutions. In the latter case the algorithm must include techniques to avoid loops on sequences of solutions.

In the present section we discuss the neighborhood we chose for problem P . In the next section we show how it can be used to obtain an efficient hill-climbing approach.

3.1 Neighborhood

Neighborhoods for scheduling problems are usually obtained from swap moves or insert moves (see, e.g., Glover and Laguna (1993)). Given a feasible solution to a single processor scheduling problem, let $\pi(i)(i = 1, \dots, n)$ denote the index of the i -th task in the sequence. A *swap move* is: (a) select two tasks $T_{\pi(i)}, T_{\pi(j)}$; (b) place $T_{\pi(i)}$ in j -th position and $T_{\pi(j)}$ in i -th position. An *insert move* is: (a) select a task $T_{\pi(i)}$ and a position $j \neq i$; (b) if $i < j$ then for $k := i + 1$ to j place $T_{\pi(k)}$ in position $k - 1$, otherwise ($i > j$) for $k := i - 1$ down to j place $T_{\pi(k)}$ in position $k + 1$; (c) place $T_{\pi(i)}$ in j -th position. After each move one has to determine the optimal completion times for the resulting sequence and the new objective function value.

In order to explore a high number of solutions $s \in \mathcal{S}$, it is advisable that the computing time requested at each move for evaluating the new solution is small. For problem P , the computation from scratch of the objective function value given by a task sequence requires $O(n)$ time, so exploration of the full neighborhood produced by swap or insert moves would require $O(n^3)$ time. Since such time would be too high for a convenient exploration of \mathcal{S} , we obtained a neighborhood from *adjacent swap moves* (*A-moves* hereafter) consisting in swaps between adjacent tasks $T_{\pi(i)}, T_{\pi(i+1)}$ only. Since $n - 1$ *A*-moves are possible, exploration of the full neighborhood

requires $O(n^2)$ time at most. In addition, as we will show in the following, in most cases the new objective function value can be computed in constant time or by examining a small subset of tasks. Our restriction of the neighborhood does not affect the possibility of obtaining the globally optimal solution, as proved by the following

Theorem 1 *For each solution $s \in \mathcal{S}$ there exists a sequence of A -moves leading from s to a globally optimal solution s^* .*

Proof. The required sequence is as follows. Take the task which is last in s^* , and move it to the last position in s . Iterate with the task which is last but one in s^* , and so on. It is easily seen that each A -move of this sequence produces a feasible solution. \square

In order to show how the objective function variation produced by an A -move can be efficiently computed, we need to examine some characteristics of a solution to P .

3.2 Feasible solution structure

Given any subset S of task indices, let $W(S) = \sum_{T_j \in S} w_j$ denote the total penalty of S .

Observation 1 *Given any feasible completion times assignment, assume that all tasks in $S \subseteq \{T_1, \dots, T_n\}$ can be shifted by Δ ($\Delta \geq 0$) time units: in this case the objective function value varies by $\Delta \cdot W(S)$, the contribution of S becoming $\sum_{T_j \in S} w_j(C_j + \Delta)$.*

Let us consider the schedule obtained from a sequence π through optimal idle times insertion (see Feo, Venkatraman and Bard (1991)). The schedule can be subdivided into blocks: we define a *block* as a set

$$B_i = \{T_{\pi(h)}, T_{\pi(h+1)}, \dots, T_{\pi(k)}\}$$

of consecutive tasks with no idle time between each other, and with idle times between B_i and B_{i+1} . More formally, B_i must satisfy:

- a) $C_{\pi(j)} + p_{\pi(j+1)} = C_{\pi(j+1)}$ ($j = h, \dots, k - 1$);
- b) time instant $C_{\pi(h)} - p_{\pi(h)}$ is preceded by an idle time or is zero (if $h = 1$);
- c) time instant $C_{\pi(k)}$ is followed by an idle time, or $k = n$.

We will assume, without loss of generality, that any subset S of consecutive tasks such that $W(S) = 0$ has been scheduled as early as possible. From this assumption and Observation 1 one easily has the following

Property 1 *If block B_i is preceded by an idle time then $\sum_{j=h}^l w_{\pi(j)} < 0$ for $h \leq l \leq k$.*

Property 2 *If block B_i is preceded by an idle time then at least one deadline constraint is active for a task of B_i (i.e., $C_{\pi(j)} = d_{\pi(j)}$ for at least one task $T_{\pi(j)} \in B_i$).*

Corollary 1 *Any block B_i , but possibly B_1 , has $W(B_i) < 0$ and an active deadline. The same holds for B_1 if it is preceded by an idle time; if instead $W(B_1) \geq 0$ then B_1 is scheduled starting from time zero.*

Given any block B_i , let $\pi(\alpha)$ be the largest index of a task of B_i for which the deadline constraint is active, if any; if $i = 1$ and $W(B_1) \geq 0$, let $\alpha = h$. We will call $d_{\pi(\alpha)}$ the *critical deadline* of B_i . From Observation 1 we obtain

Property 3 *For any block we have $\sum_{j=l}^k w_{\pi(j)} \geq 0$ for $\alpha < l \leq k$.*

3.3 Efficient neighborhood exploration

Let us consider the current sequence π and let $x = \pi(\nu)$ and $y = \pi(\nu + 1)$ denote the indices of the pair of adjacent tasks which are swapped by an A -move. The move is obviously examined only if

$\sum_{j=1}^{\nu+1} p_{\pi(j)} \leq d_x$, i.e., the resulting sequence π' (defined as $\pi'_j = \pi_j$ if $j < \nu$ or $j > \nu + 1$; $\pi'(\nu) = y, \pi'(\nu + 1) = x$) is feasible. Let (C_j) be the optimal completion times for sequence π , with objective function value $z = \sum_{j=1}^n w_j C_j$. Let $B_i \ni T_y$ and observe that, by Property 2, any block B_q with $q > i$ has a critical deadline. Hence one easily obtains the following

Lemma 1 *When a sequence π' is obtained through an A-move on the current sequence π , the optimal completion times for π' and π are the same for all tasks belonging to blocks following the block containing T_y .*

In the following theorem we examine a *normal swap*, defined as an A-move in which (see Figure 1):

- (i) T_x and T_y belong to the same block B_i ;
- (ii) d_y is not the critical deadline of B_i ;
- (iii) $d_x \geq C_y$.

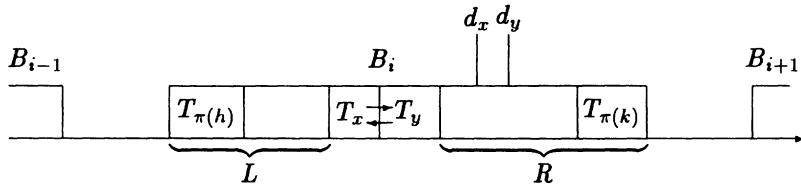


Figure 1: Normal swap

Theorem 2 *After a normal swap, if (α) $w_x < 0$ and $w_y < 0$, or (β) $w_x \geq 0$ and $w_y \geq 0$, or (γ) $w_x \geq 0$ and $w_y < 0$, then the schedule (C'_j) obtained by setting $C'_x = C_y$, $C'_y = C_y - p_x$ and $C'_j = C_j$ for $j \notin \{x, y\}$ is optimal for the new sequence, hence the new objective function value is*

$$z' = z + w_x p_y - w_y p_x \quad (2)$$

In the remaining case ((δ) $w_x < 0$ and $w_y \geq 0$), C'_j is optimal for all $T_j \in B_q$ with $q > i$, hence only the idle times for the tasks of i blocks (at most) must be re-computed.

Proof. It is immediate to see that schedule (C') is always feasible. Since the swap does not change the value of $W(B_i)$, no shift of the entire block B_i can improve the value of this schedule. Indeed, if $W(B_i) < 0$, an improvement could only be obtained by shifting right B_i (see Observation 1), which is forbidden by the active deadline (see Corollary 1 and assumption (ii)); if instead $W(B_i) \geq 0$, no shift left is possible since $B_i \equiv B_1$, scheduled from time zero (see Corollary 1). Hence the new schedule could only be improved by splitting the block and shifting some part of it. We next show that this cannot happen unless (δ) occurs.

Hence assume that $w_x \geq 0$ or $w_y < 0$, and let $L = \{T_j \in B_i : C_j < C_x\}$ and $R = \{T_j \in B_i : C_j > C_y\}$ (see Figure 1). We have to prove two facts: **(F1)** *no sub-block $\{T_{\pi'(h)}, \dots, T_{\pi'(l)}\}$ ($l = h, \dots, k$) can decrease the objective function value by shifting left;* **(F2)** *no sub-block $\{T_{\pi'(l)}, \dots, T_{\pi'(k)}\}$ ($l = h, \dots, k$) can decrease the objective function value by shifting right.*

(F1) If $B_i \equiv B_1$, scheduled from time zero, then no left shift is possible. Otherwise, from definition of π' and Property 1 we immediately have $W(\{T_{\pi'(h)}, \dots, T_{\pi'(l)}\}) = W(\{T_{\pi(h)}, \dots, T_{\pi(l)}\}) < 0$ for $l \neq \nu$ so, from Observation 1, any left shift would increase the objective function value. The same holds for $l = \nu$ (i.e., $\{T_{\pi'(h)}, \dots, T_{\pi'(l)}\} = L \cup \{T_y\}$) since: (a) if $w_y < 0$ then $W(L) + w_y < 0$; (b) if $w_y \geq 0$, hence $w_x \geq 0$, then $W(L) + w_y < 0$ follows from $W(L) + w_x + w_y < 0$ (Property 1). Observe that this is also true for the case $\nu = h$, $L = \emptyset$.

(F2) If sub-block $\{T_{\pi'(l)}, \dots, T_{\pi'(k)}\}$ includes a task with active deadline then no right shift is possible. Otherwise, from definition of π' and Property 3 we have $W(\{T_{\pi'(l)}, \dots, T_{\pi'(k)}\}) \geq 0$ for $l \neq \nu+1$ so, from Observation 1, no right shift would decrease the objective function value. The same holds for $l = \nu+1$ since: (a) if $w_x \geq 0$ then $w_x + W(R) \geq 0$; (b) if $w_x < 0$, hence $w_y < 0$, then $w_x + W(R) > 0$

follows from $w_x + w_y + W(R) \geq 0$ (Property 3 and assumption (iii)). Observe that this is also true for the case $\nu + 1 = k$, $R = \emptyset$.

We have thus proved that (C'_j) is optimal in cases (α), (β) and (γ): correctness of (2) immediately follows by observing that $p_y = C_y - C_x$.

It remains to examine case (δ): $w_x < 0$ and $w_y \geq 0$. Observe that all considerations made in (F1) and (F2) hold, except those at cases (b). Hence, if $W(L) + w_y \geq 0$ then $L \cup \{T_y\}$ must shift left and join B_{i-1} ; the resulting block could then shift left (if $W(B_{i-1}) + W(L) + w_y \geq 0$), and so on. In the worst case this requires re-computation of the idle times for the tasks in the first $i-1$ blocks and in $L \cup \{T_y\}$. If $w_x + W(R) < 0$ then $\{T_x\} \cup R$ shifts right, possibly joining B_{i+1} (if no deadline becomes active). No task in B_q ($q > i$) must shift, by Lemma 1. □

We now consider the special cases occurring when assumptions (i)–(iii) are not satisfied. Observe that assumptions (i) and (ii) are violated just once per block. Although some of the considerations of Theorem 2 could be extended to such situations, the computational improvement would be very limited. Hence, for such cases, we re-compute the optimal completion times for the tasks in blocks B_i, B_{i-1}, \dots, B_1 (at most), where B_i denotes the block containing T_y . (The current completion times remain optimal for the tasks in the remaining blocks, by Lemma 1.)

When assumptions (i) and (ii) are satisfied but (iii) is violated, we know that the new schedule (C'_j) considered in the proof of Theorem 2 ($C'_x = C_y$, $C'_y = C_y - p_x$ and $C'_j = C_j$ for $j \notin \{x, y\}$) violates d_x , so we make it feasible by setting $C'_x = d_x$ and $C'_{\pi(l)} = C'_{\pi(l+1)} - p_{\pi(l+1)}$ ($l = \nu, \nu - 1, \dots, h$). If the idle time between B_{i-1} and B_i is less than $C_y - d_x$, also the tasks in B_{i-1} are shifted left properly, and so on. It is not difficult to see that fact (F1) in the

proof of Theorem 2 remains true. Hence, in cases (α) , (β) and (γ) , completion times $C'_{\pi(1)}, C'_{\pi(2)}, \dots, C'_x$ are optimal, and new idle times must be computed only for the remaining tasks of B_i (see Lemma 1). In case (δ) instead, new idle times are computed for all tasks in blocks B_q with $q \leq i$.

4. Tabu search

The results of the previous section have been used to obtain the tabu search algorithm $T\text{-}DMV$. At each iteration the algorithm considers the neighborhood defined by all the feasible solutions produced by an A -move applied to the current solution. For each of these solutions the objective function value is evaluated through the techniques developed in Section 3.3. Let \bar{s} denote the candidate solution with smallest value, $z(\bar{s})$: if $z(\bar{s}) \geq z^*$ (where z^* denotes the best incumbent solution value) then the non-tabu solution with smallest value is selected for the next iteration; otherwise an aspiration criterion is applied and $z(\bar{s})$ is selected, even if tabu.

The tabu restriction is based on the pair of attributes defined by the indices of the swapped tasks: when T_i is exchanged with T_j , the exchange of T_j with T_i is forbidden for the next t iterations. The tabu tenure, t , is initially set to $n/2$ and dynamically varied, every 100 moves, in the interval $[3, 3n/4]$ according to the following rule: if z^* was updated during the last 100 moves, then $t := \min(3n/4, t+2)$; otherwise $t := \max(3, t - 2)$.

A first version of $T\text{-}DMV$ was obtained using the final solution s^{**} obtained by algorithm *JOIN* (see Section 2) as starting solution. We also tried to start from solution s^* obtained by *JOIN* before post-optimization: it turned out that a number of instances were efficiently solved by one of the two approaches but not by the other, and vice versa. Hence we also experimented a sort of *path-relinking* (see, e.g., Glover and Laguna (1993)) by determining, through the technique developed in the proof of Theorem 1, the sequence of the

m A-moves leading from s^* to s^{**} , and starting with the solution obtained after the first $m/2$ moves. A fourth starting solution was obtained, in the same way, using the sequence leading from s^{**} to s^* (which has nothing to deal with the previous one considered in reverse order).

The final algorithm performs the four attempts above, each time limiting the number of iterations to a maximum of $20n$ if $n \leq 50$, or to a maximum of $500 + 10n$ if $n \geq 50$.

5. Computational Experiments

We have coded in C language the tabu search procedure *T-DMV* described in the previous sections, the approximation algorithm *JOIN* and the *GRASP* heuristic.

We executed computational experiments on a Digital DECstation 5000/240 by randomly generating instances of problem II according to data sets which extend those described in Feo, Venkatraman and Bard (1991), Bard, Venkatraman and Feo and Dell'Amico, Martello and Vigo (1995). For each task T_j the values of p_j , α_j and β_j are uniformly randomly generated in range [1, 10] so that:

Class 1) $\alpha_j \leq \beta_j$ for approximately 50% of the tasks;

Class 2) $\alpha_j \leq \beta_j$ for approximately 66% of the tasks;

Class 3) $\alpha_j \leq \beta_j$ for approximately 33% of the tasks.

The deadline of each task T_j is uniformly randomly generated in range $[\beta^- \sum_{j=1}^n p_j, \beta^+ \sum_{j=1}^n p_j]$, with seven (β^-, β^+) pairs: (0.00, 1.00), (0.00, 1.25), (0.25, 1.25), (0.25, 1.75), (0.50, 2.50), (0.75, 1.25) and (0.75, 1.75).

Tables 1 and 2 refer to small-size instances, with $n \in \{10, 20, 30, 40, 50\}$. For each value of n , for each class and for each pair (β^-, β^+) ,

Table 1. Small-size instances; all classes. Average values over 105 instances.

<i>n</i>	<i>GRASP</i>			<i>JOIN</i>			<i>T-DMV</i>		
	time	% err.	opt.	time	% err.	opt.	time	% err.	opt.
10	0.30	0.32	89	0.00	0.07	100	0.45	0.01	102
20	2.27	0.60	48	0.02	0.14	79	1.15	0.06	94
30	7.60	0.62	17	0.06	0.15	58	2.11	0.11	75
40	17.83	0.95	5	0.14	0.25	34	3.64	0.13	55
50	34.81	0.86	4	0.26	0.19	26	5.41	0.11	46
total	12.56	0.67	163	0.09	0.16	297	2.55	0.08	372

five feasible instances were generated, producing a total of 525 instances.

Table 1 summarizes the results obtained by giving, for each value of n and each algorithm, the average CPU time, the average percentage error and the number of optimal solutions found, computed over all the 105 instances (of different classes). The errors were computed with respect to the optimal solution values found by the enumerative algorithm of Dell'Amico, Martello and Vigo (1995). The last line of the table gives the same information for the grand total of 525 instances.

Table 2 details the results of Table 1 for the two pairs (β^-, β^+) producing the most difficult instances, i.e., those giving highest errors. The results show that *JOIN* outperforms *GRASP*, producing very good solutions with CPU times one order of magnitude smaller. *T-DMV* considerably improves the solutions found by *JOIN*: the average errors are halved and the computing times remain very reasonable and much smaller than those of *GRASP*.

Tables 3 and 4 refer to 525 large-size instances, with $n \in \{100, 150, 200, 250, 300\}$. The entries give average CPU times and average percentage errors computed with respect to lower bound L proposed by Dell'Amico, Martello and Vigo (1995). The *GRASP* algorithm

Table 2. Small-size instances. Average values over 5 instances.

Class (β^-, β^+)	n	GRASP			JOIN			T.DMV		
		time	% err.	opt.	time	% err.	opt.	time	% err.	opt.
1 (0.25,1.25)	10	0.29	0.00	5	0.00	0.00	5	0.44	0.00	5
	20	2.25	0.45	1	0.01	0.21	2	1.00	0.00	5
	30	7.65	0.76	1	0.06	0.09	3	1.77	0.00	4
	40	17.66	1.26	0	0.12	0.88	1	3.62	0.46	2
	50	34.41	1.06	0	0.18	0.56	0	5.08	0.28	2
1 (0.25,1.75)	10	0.30	0.00	5	0.00	0.00	5	0.57	0.00	5
	20	2.39	1.26	2	0.02	0.00	5	1.14	0.00	5
	30	7.34	0.69	0	0.06	0.59	1	2.21	0.38	2
	40	17.46	1.06	1	0.12	0.29	2	3.45	0.18	3
	50	35.06	0.94	0	0.23	0.05	2	5.07	0.02	3
2 (0.25,1.25)	10	0.30	0.04	4	0.00	0.00	5	0.51	0.00	5
	20	2.25	0.77	3	0.01	0.43	3	1.29	0.04	4
	30	7.60	1.40	0	0.06	0.11	1	2.15	0.06	2
	40	17.87	0.94	0	0.10	0.02	4	3.37	0.02	4
	50	34.77	0.56	0	0.16	0.18	2	4.97	0.08	3
2 (0.25,1.75)	10	0.30	1.16	3	0.00	0.00	5	0.57	0.00	5
	20	2.34	0.64	1	0.02	0.17	4	1.26	0.00	5
	30	7.71	0.67	1	0.05	0.06	3	2.13	0.04	3
	40	18.52	1.03	0	0.11	0.36	2	3.62	0.04	3
	50	34.03	1.26	0	0.24	0.37	0	5.34	0.12	0
3 (0.25,1.25)	10	0.29	0.00	5	0.00	0.16	4	0.41	0.16	4
	20	2.13	0.94	2	0.02	0.00	5	1.21	0.00	5
	30	7.56	0.15	3	0.05	0.11	2	1.91	0.00	5
	40	17.55	0.51	0	0.10	0.15	1	3.63	0.08	3
	50	34.46	0.56	0	0.25	0.15	1	5.51	0.04	2
3 (0.25,1.75)	10	0.29	0.00	5	0.00	0.03	4	0.31	0.02	4
	20	2.22	0.09	3	0.02	0.00	5	1.10	0.00	5
	30	7.29	0.30	1	0.05	0.75	3	2.02	0.74	3
	40	17.20	0.32	0	0.12	0.10	0	3.51	0.02	3
	50	33.38	0.66	0	0.24	0.08	0	5.19	0.06	0

was not run for instances with more than 100 tasks, since the computing times were extremely large and steeply increasing with n . The good behavior of the tabu search heuristic is confirmed, although the improvement over *JOIN* is smaller: the errors do not exceed one percent and the computing times are always less than six minutes.

Table 3. Large-size instances; all classes. Average values over 105 instances.

<i>n</i>	<i>GRASP</i>			<i>JOIN</i>			<i>T_DMV</i>		
	time	% err.	time	% err.	time	% err.	time	% err.	time
100	271.30	1.38	2.02	0.77	21.08	0.72			
150	—	—	7.18	0.55	54.44	0.52			
200	—	—	19.19	0.45	113.47	0.41			
250	—	—	44.76	0.35	212.78	0.33			
300	—	—	88.99	0.33	343.97	0.30			
total	—	—	32.43	0.49	149.15	0.46			

Table 4. Large-size instances. Average values over 5 instances.

Class (β^- , β^+)	<i>n</i>	<i>GRASP</i>			<i>JOIN</i>			<i>T_DMV</i>		
		time	% err.	time	% err.	time	% err.	time	% err.	time
1 (0.25,1.25)	100	258.02	1.71	1.70	1.03	20.05	0.94			
	150	—	—	7.35	0.63	49.48	0.60			
	200	—	—	16.47	0.52	103.88	0.48			
	250	—	—	37.97	0.40	183.69	0.38			
	300	—	—	85.96	0.40	323.95	0.36			
1 (0.25,1.75)	100	264.47	1.31	1.62	0.98	20.45	0.91			
	150	—	—	6.83	0.75	47.73	0.70			
	200	—	—	14.71	0.42	85.22	0.38			
	250	—	—	42.04	0.40	190.87	0.36			
	300	—	—	72.42	0.35	285.96	0.34			
2 (0.25,1.25)	100	266.07	1.74	2.25	1.19	21.69	1.02			
	150	—	—	4.81	1.01	42.45	0.92			
	200	—	—	17.31	0.56	111.11	0.50			
	250	—	—	46.34	0.57	204.21	0.56			
	300	—	—	91.68	0.53	349.41	0.52			
2 (0.25,1.75)	100	269.35	2.19	1.88	1.20	23.33	1.06			
	150	—	—	6.02	0.57	55.57	0.56			
	200	—	—	19.91	0.68	108.79	0.66			
	250	—	—	38.08	0.55	193.05	0.50			
	300	—	—	115.21	0.53	355.22	0.50			
3 (0.25,1.25)	100	276.54	1.06	2.35	0.56	22.11	0.55			
	150	—	—	5.37	0.42	50.51	0.38			
	200	—	—	18.41	0.57	125.95	0.50			
	250	—	—	36.84	0.31	192.66	0.28			
	300	—	—	87.88	0.27	369.71	0.26			
3 (0.25,1.75)	100	271.19	1.51	1.94	0.61	19.71	0.59			
	150	—	—	7.25	0.43	53.62	0.43			
	200	—	—	17.39	0.40	105.73	0.36			
	250	—	—	42.35	0.34	214.78	0.30			
	300	—	—	61.64	0.32	300.06	0.30			

6. Acknowledgments

This work has been supported by Ministero dell'Università e della Ricerca Scientifica e Tecnologica, and by Consiglio Nazionale delle Ricerche, Italy.

7. References

- K.R. Baker and G. Scudder, Sequencing with Earliness and Tardiness Penalties: a Review, *Operations Research* 38 (1990), p. 22.
- J. Bard, K. Venkatraman and T. Feo, Single Machine Scheduling Problem with Due Dates and Earliness Penalties, *Journal of Global Optimization* 3 (1993), p. 289.
- M. Dell'Amico, Optimal Idle Time Insertion in One Machine Scheduling Problems, Tech. Report, Politecnico di Milano (1994).
- M. Dell'Amico, S. Martello and D. Vigo, Minimizing the Sum of Weighted Completion Times with Unrestricted Weights, *Discrete Applied Mathematics*, to appear (1995).
- T. Feo, K. Venkatraman and J. Bard, A GRASP for a Difficult Single Machine Scheduling Problem, *Computers and Operations Research* 18 (1991), p. 378.
- F. Glover and M. Laguna, Tabu Search in: *Modern Heuristic Techniques for Combinatorial Problems*, ed. C.R. Reeves (Blackwell Scientific Publications, Oxford, 1993).
- J.R. Jackson, Scheduling a Production Line to Minimize Maximum Tardiness, Research Report 43, Management Science Research Project, University of California, Los Angeles (1955).
- J.K. Lenstra, A.H.G. Rinnooy Kan and P. Brucker, Complexity of Machine Scheduling Problems , *Annals of Discrete Mathematics* 1 (1977), p. 343.

Heuristics for the Optimal Control of Thermal Energy Storage

Gregor P. Henze

University of Colorado at Boulder

Department of Civil Engineering

E-Mail: henze@colorado.edu

Manuel Laguna

College of Business

E-Mail: laguna@colorado.edu

Moncef Krarti

Department of Civil Engineering

E-Mail: krarti@colorado.edu

Abstract:

A dynamic programming based search algorithm was developed to determine the control strategy that minimizes the total operating cost of a thermal energy storage system. A mixed integer programming model was used to validate the findings of the DP-based heuristic. To expedite the search procedure, a second heuristic was developed that determines the approximate value of the two search parameters. The DP algorithm then searches within a smaller domain marked by a search radius around the MIP approximations. Significant savings in run time were realized at the cost of only marginal deviations in the minimum objective function value compared to the full DP-based search.

Keywords: Optimal control, thermal energy storage, heuristics, dynamic programming, mixed integer programming.

1. Introduction

Thermal energy storage (TES) has been shown to be effective in reducing operating cost of cooling equipment. By operating the refrigeration equipment (“chiller”) during off-peak hours to recharge a thermal storage medium and discharge the storage during on-peak

hours, a significant fraction of the on-peak electrical demand, i.e. peak power consumption, and energy consumption is shifted to off-peak periods. Cost savings are realized because energy rates favor leveled energy consumption patterns. The variable energy rates reflect the high cost of providing energy during relatively short on-peak periods. Hence, they constitute an incentive to reduce or avoid operation during on-peak periods by cool storage. Large differential between on- and off-peak energy and peak consumption rates make cool storage economically attractive.

Thermal energy storage for cooling systems use either chilled water, storing on the order of 20 Btu/lb, or ice, storing approximately 140 Btu/lb. Due to phase change, ice can be stored in relatively small storage tanks.

Experience with operating TES systems shows that a significant shortfall of the current technology is the poor performance of the control strategies currently employed. The best known control mechanism for TES is *chiller-priority*. This simple heuristic consists of using direct cooling (no storage usage) anytime during off-peak hours, and during on-peak hours as long as (a) the current cooling load can be met by the chiller (often down-sized in capacity), or (b) an imposed limit on the peak electrical power consumption is not yet exceeded by the cooling plant. The cooling plant consists of the chiller, pumps, fans, and all other equipment needed to provide cooling. In simple terms, the cooling load is that amount of heat that has to be removed so that the room temperature remains within a comfortable level. Though chiller-priority can be an appropriate strategy under peak cooling load conditions, it generally makes poor use of the available storage during swing seasons, i.e. typically, spring and fall seasons. With chiller-priority being the most widely used control, the potential of TES is only partially exploited.

The need for an *optimal controller* for TES systems was expressed in a report on case studies by Akbari *et al.* (1992). According to this report, only a few of the TES systems take advantage of the daily variations in climate and operating conditions to optimize charging and discharging strategies in the system. With no adaptive control currently available, the resources are ineffectively utilized and the full potential of TES cannot be achieved. Therein lies the motivation for the development of an optimal controller.

2. Model of Cooling System

The predominant feature of any storage device is its ability to bridge the temporal gap between supply and demand. In a system without storage, the cooling load has to be met instantaneously, therefore the corresponding power consumption of the cooling plant cannot be shifted to periods when electricity is cheaper. In thermal energy storage there are two choices for a cooling source: direct cooling from the chiller and the discharge of the storage tank. At night, when only little or no cooling is required, the tank can be recharged. This system has only one significant *state variable*, the state-of-charge x of the storage tank (where $x_{min} \leq x \leq x_{max}$). At any point in time, the question is to either charge, discharge or remain inactive (Note: This does not necessarily mean that there is no cooling load; the chiller may be required to meet that load directly, i.e. without storage.).

The control variable u represents the charging or discharging rate of the storage device, and is bounded as follows

$$u_{min}(k) \leq u(k) \leq u_{max}(k), \quad (1)$$

where u_{min} is the negative value of the maximal discharging rate at time k , i.e. a parameter determined by the heat transfer capabilities of the ice storage system, and u_{max} is the chiller capacity in the ice-making mode at time k . For the ice storage system the *state transition equation* is simply

$$x(k+1) = x(k) + \frac{u(k)}{SCAP} \quad (2)$$

where $SCAP$ is the storage capacity.

The total power consumption $P(k)$ of the cooling plant is a function of the control u , time k , and the state-of-charge x (for the most general case), i.e. $P = f(u(k), x(k), k)$. The task of minimizing operating cost is thereby transformed into the problem of finding the optimal sequence of charging and discharging rates $u(k)$ subject to bound constraints on u and x . The value of $P(u(k), x(k), k)$ is the minimum power consumption for a given cooling load and ambient conditions. This value is found by a so-called *plant optimization*, described in details in Krarti *et al.* (1995).

3. Dynamic Programming Approach

The cost function in this problem is the total cost incurred from the electrical energy required to operate the cooling plant with all of its components. For commercial customers, the function consists of two elements:

- the cost of the *peak* electrical power consumption [kW] (also called *demand* charge) that occurred during the billing period

$$C_p = \sum_{\nu=0}^1 r_{p,\nu} P_{max,\nu},$$

where $P_{max,\nu}$ is the peak power consumption during off-peak ($\nu = 0$) and on-peak ($\nu = 1$) periods. Typical on-peak periods are Monday through Friday 8 a.m. to 6 p.m. while the remaining hours and weekend are considered off-peak. The $r_{p,\nu}$ values are the corresponding rates.

- the cost of the electrical *energy* [kWh] consumed over the billing period.

$$C_e = \sum_k^K r_{e,\gamma(k)} P(k)$$

where K is the total number of periods and the function $\gamma(k)$ specifies whether period k is on-peak or off-peak

$$\gamma(k) = \begin{cases} 0 & \text{if } k \text{ is off-peak} \\ 1 & \text{if } k \text{ is on-peak} \end{cases}$$

Optimal control in the given context is thus defined as the sequence of control values that minimizes the *combined* energy and peak power charge of the utility bill.

$$J = C_p + C_e = \sum_{\nu=0}^1 r_{p,\nu} P_{max,\nu} + \sum_k^K r_{e,\gamma(k)} P(k) \quad (3)$$

To illustrate the tradeoff between the peak power and energy cost, consider the following cases. Suppose the cooling system were operated while enforcing a low limit on the peak power consumption, the chiller would operate longer times at lower capacities to recharge the storage tank. The resulting peak charge C_p would be low, however, the energy charge C_e would increase, because the chiller's efficiency is reduced at low cooling capacities. Conversely,

if there were no peak consumption limit, the chiller could operate at its maximum capacity and efficiency and would thus be capable of recharging the system in a relatively short time. In this case the peak cost C_p would be high and the energy charge C_e would be low. Since the commercial customer is billed for the sum of both charges, there is an optimal set of consumption limits for which the total cost is minimized. To operate the cooling plant equipment efficiently, one needs to know from the beginning what the maximum power consumptions in each rate period ν will be.

A heuristic based on a dynamic programming algorithm embedded in a grid search aimed at finding the optimal peak limits was developed by Krarti *et al.* (1995). The approach is to add one *outer loop* for each rate period ν , and incrementally decrease the D_ν variables. These variables constitute peak consumption limits, each applying to one rate period. For each set of values of D_ν (for $\nu = 0, 1$) a complete optimization over the corresponding time window of K hours is carried out by the dynamic programming algorithm to be described later in this paper. For each hour k of the optimization, the total power consumption has to meet $P(k) \leq D_\nu$, i.e. at each hour the plant power consumption is restricted to the value D_ν of rate period ν at hour k . The power consumption of the cooling plant for any investigated control action is subject to the constraint that the current cooling load $Q(k)$ is met. Thus, any investigated admissible control implicitly meets the current load.

After each DP run, the total cost is calculated. It is assumed that the function $J = f(D_\nu)$ for which discrete values have been calculated, is continuous, allowing the determination of the approximate locus of the global minimum from the discrete values of a rather coarse grid.

The minimum cost is approximately found in three consecutive stages. Initially, the two *upper bounds* on the search domain have to be determined. The maximum power consumption limit $D_{1,max}$ of the on-peak period is found as the cooling plant power consumption when the storage is empty ($x = 0$), the requested control is neither charging nor discharging ($u = 0$), and the system has to meet the peak cooling load during the on-peak period, thus

$$D_{1,max} = P_{max,1}^{x=0,u=0}. \quad (4)$$

In essence, this is the peak power consumption of a cooling plant

without storage. If the ice storage system is to be useful, it has to shift energy demand from on-peak to off-peak. For the off-peak period $\nu = 0$, the maximum power consumption ceiling corresponds to the power consumption that is incurred when the cooling plant *simultaneously* meets the off-peak maximum cooling load through direct cooling *and* charges the ice storage at a capacity that results in a power consumption that equals the on-peak demand without storage, or formally

$$D_{0,max} = P_{max,0}^{x=0,u=0} + P_{max,1}^{x=0,u=0}. \quad (5)$$

In the first loop, the *lower bounds* $D_{\nu,min}$ are set to zero. Next, the search ranges ($D_{\nu,max} - D_{\nu,min}$) are divided into Z intervals ΔD_{ν} for each rate period ν . Starting from point $(D_{0,max}, D_{1,max})$, the search is carried out at each node on a line of constant D_0 by decreasing D_1 by ΔD_1 until a point is reached where the problem is infeasible (not all cooling loads can be met) with the given pair of power consumption limits. This is indicated by an infinitely high cost. Then D_0 is decreased by ΔD_0 and the process begins anew with the reduced D_0 . The first loop ends, as soon as D_0 is decreased to a value from which not even resetting D_1 to $D_{1,max}$ will result in a feasible control sequence. The first loop in the search for the global cost minimum is illustrated in Figure 1.

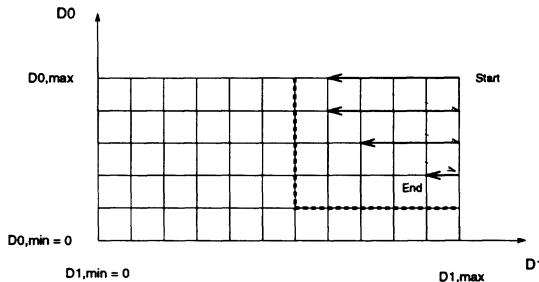


Figure 1: Domain of first search loop.

In the second search loop, the smallest values of D_{ν} for which feasible solutions were obtained—indicated by the bold dashed line in Figure 1—are increased by ΔD_{ν} , to become the new lower bounds $D_{\nu,min}$ for the second loop. The new ranges ($D_{\nu,max} - D_{\nu,min}$) are

again quantized into Z intervals. The bold dashed line in Figure 2 shows the quantization of the first loop with the normal solid lines showing the refined grid.

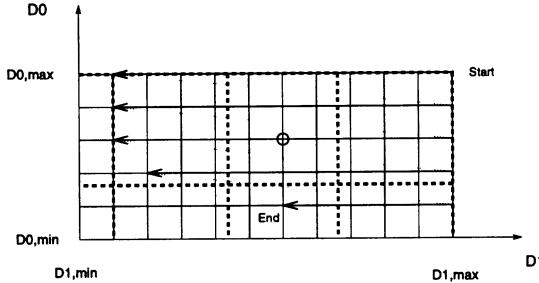


Figure 2: Second search loop with refined grid.

In the third and last loop, the pair of demand limits (D'_0, D'_1) for which the minimum cost is obtained (indicated by a bold circle in Figure 2), is now used to define two new pairs of upper and lower bounds for the final search stage:

$$D_{\nu,\min} = D'_\nu - \Delta D_\nu$$

and

$$D_{\nu,\max} = D'_\nu + \Delta D_\nu.$$

Again, the new ranges $(D_{\nu,\max} - D_{\nu,\min})$ are divided into Z intervals and for each pair of quantized power consumption ceilings the cost is calculated. The minimum cost that is found for this new search range, is the total cost associated with a near-optimal control sequence.

Figure 3 displays a sample plot of the total energy cost versus values of the power consumption limit variables D_0 and D_1 . In this plot, infinite costs corresponding to inadmissibly low power consumption ceilings, are depicted as zero costs for presentation purposes. One can see how jagged and alpine the overall landscape of the global cost function is; the global minimum, however, is expected to be found in a rather shallow valley. This confirms the appropriateness of the search described above, since a gradient-based procedure will likely get trapped in local minima.

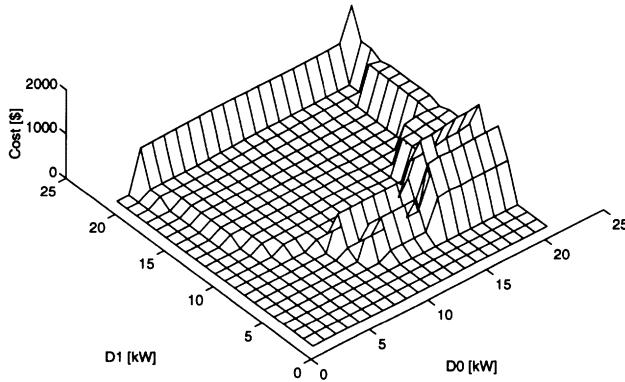


Figure 3: Sample plot of the total energy charge versus values of D_ν in the vicinity of the global minimum.

It should come as no surprise that the search for the optimal set of power consumption limits is computationally extremely expensive. For each investigated set of power consumption limits, a complete DP run over the entire time horizon is necessary. One significant advantage, however, is that the DP environment allows the use of cooling plant models of arbitrary complexity. As mentioned above, realistic plant models will account for the reduced efficiency of the plant at low capacities known as the *part-load performance*. They will also take into account the impact of ambient conditions (temperature T , relative humidity RH) and the state-of-charge x on the overall plant efficiency. In general, DP allows the consideration of all real-world influences such that the modeled mapping $\{u(k), x(k), Q(k), T(k), RH(k), \dots\} \rightarrow P(k)$ approaches the true relationship.

3.1 The DP Recursion

The *instantaneous cost* associated with the transition from one state $x(j)$ at time j through the control $u(j)$ for the given process of operating the cooling plant is the energy cost incurred during $j \rightarrow j + 1$

$$l[x(j), u(j), j] = r_{e,\gamma(j)} P[x(j), u(j), j]. \quad (6)$$

The total cost of starting at time k and ending at time K can be expressed by

$$J = \sum_{j=k}^K l[x(j), u(j), j]. \quad (7)$$

There will be a sequence of controls u that minimize J stated as

$$I(x, k) = \min_{u(j) \in U(j), j=k, \dots, K} \left\{ \sum_{j=k}^K l[x(j), u(j), j] \right\} \quad (8)$$

where $U(j)$ is the set of admissible controls. $U(j)$ contains the set of equidistant values of the control variable in the range $(u_{\max}(j) - u_{\min}(j))$. In deriving an iterative functional equation, the last expression can be rewritten as

$$I(x, k) = \min_{u(j) \in U(j), j=k, \dots, K} \left\{ l[x(k), u(k), k] + \sum_{j=k+1}^K l[x(j), u(j), j] \right\}$$

where $x = x(k)$. However, this expression is equivalent to

$$I(x, k) = \min_{u(k) \in U(k)} \min_{u(j) \in U(j), j=k+1, \dots, K} \left\{ l[x(k), u(k), k] + \sum_{j=k+1}^K l[x(j), u(j), j] \right\} \quad (9)$$

If the minimization operation is applied separately to the two arguments, the first term becomes

$$\begin{aligned} \min_{u(k) \in U(k)} \min_{u(j) \in U(j), j=k+1, \dots, K} & \{l[x(k), u(k), k]\} = \\ & \min_{u(k) \in U(k)} \{l[x(k), u(k), k]\} \end{aligned}$$

Defining the *state transition equation* g (Equation (2))

$$x(k+1) = g[x, u(k), k], \quad (10)$$

the second argument in Equation (9) can be expressed as

$$\begin{aligned} \min_{u(k) \in U(k)} \min_{u(j) \in U(j), j=k+1, \dots, K} & \left\{ \sum_{j=k+1}^K l[x(j), u(j), j] \right\} \\ & = \min_{u(k) \in U(k)} \{I(x(k+1), k+1)\} \\ & = \min_{u(k) \in U(k)} \{I(g[x, u(k), k], k+1)\} \end{aligned}$$

Therefore, while suppressing index k on $u(k)$, it is proven that

$$I(x, k) = \min_{u \in U} \{l[x(k), u(k), k] + I(g[x, u(k), k], k + 1)\} \quad (11)$$

and, as a result, an iterative relation is found that can be used in determining $I(x, k)$ for all $x \in X$, where X is the set of admissible states, entirely from the knowledge of $I(x, k + 1)$ for all $x \in X$. At each (x, k) -node the optimal control u^* is saved that yields $I(x, k)$. A near-optimal control sequence can be determined by starting from any $x(k = 0)$ in X and then move through time by choosing interpolated values of u^* which will lead to a new $x(k + 1)$ according to Equation (2).

4. Mixed Integer Programming Approach

A mixed integer programming (MIP) model was developed to validate the DP-based heuristic under some simplifying assumptions. The MIP formulation can only capture simple plant models. The realistic plant model requires a separate highly nonlinear optimization process. For example, the influence of the decision variable $u(k)$ on the overall plant efficiency known as the *electric input ratio* $EIR(k)$ cannot be modeled. Thus, the influence of part-load performance is neglected and only rather crude plant models can be utilized. For the purpose of validation, a simplified model will be incorporated in this part of the study.

4.1 Simplified Plant Model

The cooling plant is assumed to operate with a constant efficiency of $EIR_{(ch)} = 0.9 \frac{kW}{ton}$ in the chilled-water mode and with a reduced efficiency of $EIR_{(ice)} = 1.3 \frac{kW}{ton}$ in the ice-making mode. The higher EIR in icemaking mode reflects the increased power consumption when providing cooling at subfreezing temperatures. These values refer to the entire cooling plant. The total power consumption $P(k)$ of the simplified plant is

$$P(k) = \begin{cases} Q_{ch}(k)EIR_{(ice)} & \text{if } u > 0, \text{ i.e. charging} \\ Q_{ch}(k)EIR_{(ch)} & \text{if } u \leq 0, \text{ i.e. discharging} \end{cases}$$

where the *chiller load* $Q_{ch}(k)$ is defined as the sum of the cooling load $Q(k)$ and the discharging/charging rate $u(k)$

$$Q_{ch}(k) = Q(k) + u(k). \quad (12)$$

Since the model includes no adjustable plant parameters, plant optimization is not required in the MIP environment. The ice storage

tank is designed to store up to four times the peak cooling load Q_{peak} over the extent of one hour (Δt), i.e.

$$SCAP = 4Q_{peak}\Delta t \quad [\text{ton-hrs}],$$

while the chiller is downsized by 20% from the peak load to

$$CCAP = 0.80Q_{peak} \quad [\text{tons}].$$

Note that the unit “ton” is used in refrigeration systems and denotes a cooling capacity of $[\text{ton}] = 12,000 \frac{\text{BTU}}{\text{hr}} = 3.52\text{kW}$. This simplified model will be used in both the MIP formulation and the DP-based heuristic for validation purposes in Section 5.

4.2 MIP Model of the Simplified Ice Storage Problem

Because of the need for creating a variable for each decision and state of the system at any given hour, all time-dependent variables receive an *index* that maps to the corresponding hour in the original problem; thus $x(k)$ becomes x_k , etc. The control variable

$$u_{min}(k) \leq u(k) \leq u_{max}(k)$$

has been redefined as two positive control variables to account for the fact that the system is *either* in charging mode with the chiller making ice *or* in discharging mode with the chiller operating in the chilled-water mode:

$$0 \leq u_k^+ \leq u_{max,k}$$

$$0 < u_k^- \leq u_{min,k}$$

For those hours in which $CCAP > Q_k$, i.e. the chiller capacity exceeds the load, the decision to charge or discharge is made by choosing values for the binary variable y_k . This variable represents the operating mode of the cooling plant, where $y_k = 0$ means discharging and $y_k = 1$ means charging. To reduce the number of variables at every stage k , a preprocessing determines whether or not the chiller capacity exceeds the load. If $CCAP > Q_k$ the system can either charge or discharge, thus the three variables y_k , u_k^+ , and u_k^- have to be declared. If $CCAP \leq Q_k$, the system can only discharge, consequently only u_k^- has to be declared. The mathematical representation of this problem is:

Minimize

$$C = D_0 r_{p,0} + D_1 r_{p,1} + \sum_{k=0}^K r_{e,\gamma(k)} P_k \quad (13)$$

subject to

- For $k = 0$, i.e. at the beginning of the optimization period, a full storage is assumed ($x_0 = 1.0$). If $CCAP > Q_k$

$$x_1 - \frac{u_0^+ - u_0^-}{SCAP} = x_0$$

where u_0^+ is kept only for the sake of generality given that x_0 may be different from one, and if $CCAP \leq Q_k$

$$x_1 + \frac{u_0^-}{SCAP} = x_0.$$

- For all remaining hours $1 \leq k \leq K$ the state transition equation is reformulated for the new control variables. If $CCAP > Q_k$, the system can charge and discharge

$$x_{k+1} - x_k - \frac{u_k^+ - u_k^-}{SCAP} = 0$$

but if $CCAP \leq Q_k$ the system can only discharge

$$x_{k+1} - x_k + \frac{u_k^-}{SCAP} = 0.$$

- For all hours where $CCAP > Q_k$ the power consumption can be formulated as

$$\begin{aligned} P_k - EIR_{(ice)} u_k^+ + EIR_{(ch)} u_k^- - \\ Q_k [EIR_{(ice)} - EIR_{(ch)}] y_k = Q_k EIR_{(ch)} \end{aligned}$$

which becomes

$$P_k - EIR_{(ice)} u_k^+ = Q_k EIR_{(ice)}$$

if the system decides to charge ($y_k = 1, u_k^- = 0$), and if the system decides to discharge ($y_k = 0, u_k^+ = 0$),

$$P_k + EIR_{(ch)} u_k^- = Q_k EIR_{(ch)}$$

as in the case when $CCAP \leq Q_k$.

- Charging and discharging limits when $CCAP > Q_k$ are enforced by

$$u_k^+ - (CCAP - Q_k) y_k < 0$$

and

$$u_k^- + Q_k y_k < Q_k.$$

- The power consumption may not exceed the limits D_0 or D_1 , which means that

$$P_k - D_{\gamma(k)} < 0.$$

- The state-of-charge has to remain within

$$0 \leq x_k \leq 1.$$

If $CCAP \leq Q_k$ the system has to discharge *at least* the difference required to aid the chiller in meeting the load and *at most* the full load, that is

$$Q_k - CCAP < u_k^- < Q_k$$

- Finally, if $CCAP > Q_k$, then

$$y_k \in \{0, 1\}.$$

5. Validation

In order to validate the accuracy of the DP-based search, the simplified plant model was implemented in the MIP environment. A commercial optimization software package (CPLEX (1994)) was used to find the optimal solution. The horizon was two weeks (336 hours) and an adiabatic ice storage system, i.e. one without heat transmission losses, was investigated. Figure 4 shows the state-of-charge of the ice storage system during the first five weekdays of the horizon versus time under optimal control as found by the DP-based heuristic and the optimal control determined by the MIP model. One can notice a significant similarity in the charging and discharging behavior of the ice storage system under the control proposed by the DP-based heuristic and the optimal control found by the MIP model.

The MIP model, after making the binary decision of whether to charge or discharge, will choose control variables from a *continuous range*. If both approaches are based on the same plant model, the MIP solution can be termed the true optimum. The DP embedded in the grid search is a *discrete representation* of the problem, since only a finite number of controls can be chosen from the set of admissible controls U . Consequently, DP approximates the true solution, MIP on the other hand, finds an optimal control sequence for the given model. The differences, however, between the optimal

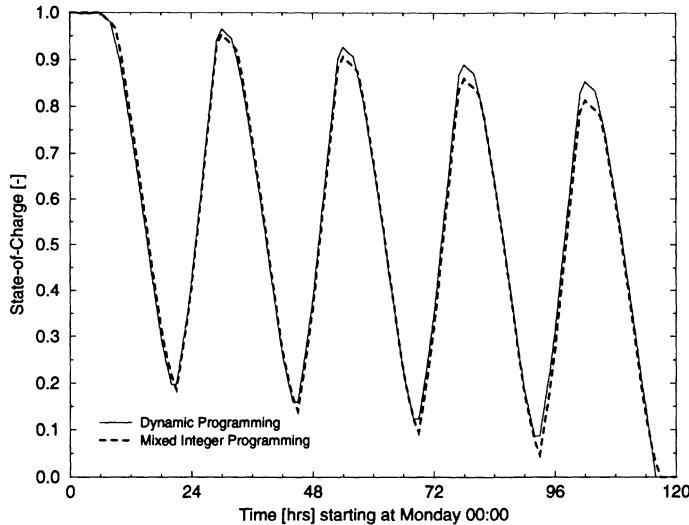


Figure 4: State-of-charge versus time under the control sequence found by the DP-based heuristic and the optimal control determined by mixed integer programming.

and approximate control sequences are quite small (as indicated in Table 1).

Additional validation runs were performed over a horizon of $K = 200$ hours with five different utility rate structures and four different cooling load profiles resulting in twenty combinations. Using a grid in the DP-based search of $N = 40$ intervals in the state-of-charge and $M = 40$ intervals in the control variable, leads on average to a difference of +3.47% (standard deviation 3.19%) in the total cost

Table 1: Comparison of approximate and optimal controls.

CONDITION	OPTIMAL CONTROL APPROACH		
	DP	MIP	Difference
Demand Limit $D_{max,0}$	179.9 kW	179.6 kW	+0.17%
Demand Limit $D_{max,1}$	105.9 kW	105.1 kW	+0.76%
Total Cost C	\$2626	\$2618	+0.31%

C , -1.37% (standard deviation 9.08%) in the off-peak demand limit, and +8.96% (standard deviation 8.84%) in the on-peak demand limit.

6. A Faster Heuristic

While the solution of control problems with a two week horizon requires hours of run time (2–3 hrs) for the DP-based approach even on the fastest available workstations, the MIP solution (either optimal or near-optimal) can be found in only a few minutes (4–5 min). In this paper, a new heuristic is proposed. In this approach, the grid search is reduced by narrowing the D_ν ranges. Although the MIP approach cannot utilize realistic plant models, a simplified model generally captures enough of the load-shifting characteristics of TES. Then, power consumption limits can be found that are sufficiently close to the limits found by the DP-based approach employing a realistic plant model.

The procedure is to solve the MIP model, then take the power consumption limits $D_{0,MIP}$ and $D_{1,MIP}$ as a starting point for the DP-based search method, and explore the domain defined by the radius r around the starting point. The new lower and upper limits of the DP-based search domain are therefore

$$D_{\nu,min} = (1 - r)D_{\nu,MIP}$$

and

$$D_{\nu,max} = (1 + r)D_{\nu,MIP},$$

respectively. In the limiting case $r = 0$, the DP algorithm will only determine the total cost once for the power consumption limits ($D_{0,MIP}, D_{1,MIP}$). Conversely, if $r = 1$ the lower boundaries of the search domain will be $D_{\nu,min} = 0$ and the upper bounds are $D_{\nu,max} = 2D_{\nu,MIP}$. This implies that when

$$2D_{0,MIP} < P_{max,0}^{x=0,u=0} + P_{max,1}^{x=0,u=0}$$

and

$$2D_{1,MIP} < P_{max,1}^{x=0,u=0},$$

the heuristic may not match the solution found by the full grid DP-based search. By running a full DP search as described above and comparing the peak consumption limits with those found by MIP, it can be assessed whether this event is likely or not.

It is expected from the validation above, that a radius of $r = 0$ should theoretically suffice to find the true limits if the same plant model is used. Using eight different rate structures and five different load profiles, i.e. forty combinations, the objective function value determined by the DP-based search was on average 1.57% above (standard deviation 1.92%) the value determined by mixed integer programming. Given the quantization of the DP algorithm, these findings support the assumption. The more simplifications the MIP plant model contains relative to the DP plant model, the larger r has to be in order to define a search region that includes the best (possibly optimal) peak power consumption limits.

7. Results

An analysis is conducted in which five different cooling load profiles and five different utility rate structures are investigated resulting in 25 parametric runs. The utility rate structures are taken from a report published by the Gas Research Institute (White *et al.* (1993)) and are representative of utilities across the US which are strongly involved in thermal energy storage.

The cooling load profiles are part of a commercial cooling load library established by the Wisconsin Center for Demand-Side Research WCDSR (1994). The five buildings are located in the state of Wisconsin and span a wide range of cooling load profile characteristics: A retail store, a grocery store, a medium-size three-story office building, a large 20-story office building, and a two-story medical facility.

A rather complex plant model requiring the solution of numerous nonlinear equations was used to model an indirect ice storage system with a reciprocating chiller and a water-cooled cooling tower. Either one or two variables (temperature setpoints) have to be adjusted to optimize the plant operation. Before the MIP algorithm was used to estimate the set of power consumption limits, the DP-based approach was run for a conventional control strategy over the time horizon of 200 hours. For each hour, the total power consumption was divided by the chiller load to yield the electric input ratio EIR . The average value in the case of charging was found to be $EIR_{ice} = 0.329$ and in the case of discharging $EIR_{chw} = 0.271$. These values essentially lump all influences on the plant operation together into two numbers and are subsequently used in the MIP runs.

Both algorithms are run for the 25 combinations. In 100% of the cases (25/25) the MIP estimate of the off-peak maximum power consumption was below the value found by the full DP-based search and in 52% of the cases (13/25) for the on-peak maximum power consumption. On average, the MIP off-peak estimate was 23% below the DP-based value with a standard deviation of 10%, while the MIP on-peak estimate was 12% above the DP-based value with a standard deviation of 39%. This result indicates that the MIP procedure models the off-peak process better than the on-peak process with respect to the plant model used in the DP procedure. An optimality gap of 1% was used in all MIP runs.

Defining the demand radius Γ ,

$$\Gamma = \sqrt{D_0^2 + D_1^2},$$

the MIP estimates can be found $\Delta\Gamma$ away from Γ , where

$$\Delta\Gamma = \sqrt{(D_0 - D_{0,MIP})^2 + (D_1 - D_{1,MIP})^2}.$$

The average relative distance $\bar{\gamma}$, where

$$\gamma = \frac{\Delta\Gamma}{\Gamma} = \frac{\sqrt{(D_0 - D_{0,MIP})^2 + (D_1 - D_{1,MIP})^2}}{\sqrt{D_0^2 + D_1^2}},$$

was found to be 5.71% with a standard deviation of 7.41% for the 25 investigated cases.

Since the MIP peak power consumption values lead to inadmissible solutions with the realistic plant model, in the second step the DP-based search is initiated with a given search radius r . For each of the cases, r is increased in 5%-steps starting from $r = 0.05$ and until a minimum radius r_{min} for which a feasible solution was found. Figure 5 is a histogram of the number of occurrences of the corresponding minimum search radius r_{min} in the range $0.05 \leq r_{min} \leq 1.0$. In 80% of the investigated cases, a search radius of $r_{min} \leq 0.30$ is sufficient to find a feasible solution. Since the number of explorations and therefore the CPU time roughly increase with the square of the search radius, a significant reduction in execution time is achieved. On average, the DP-based search with the MIP suggestion of peak power consumption at the search radius r_{min} is 21.7 times (standard deviation 8.3%) faster than the full DP-based search. The value of

the cost function that is found at r_{min} is on average 3.8% (standard deviation 6.0%) above the value generated by the full DP search. The substantial acceleration of the optimization procedure using the heuristic appears to warrant the loss in accuracy.

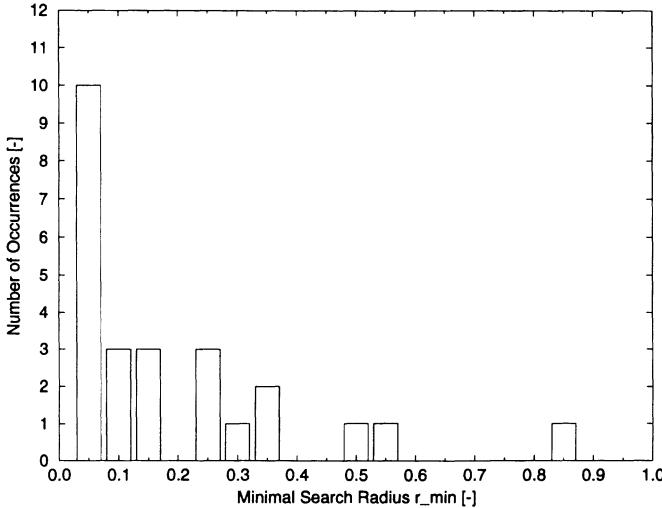


Figure 5: Histogram of the number of occurrences of the corresponding minimum search radius r_{min} .

8. Summary

A DP-based heuristic was developed that estimates the minimum total cost and optimal peak power consumption limits for the off- and on-peak period of an ice storage system by conducting an exhaustive search over a defined grid. This novel approach allows the minimization of the actual operating cost combining energy cost and peak power consumption cost. Previous work only addressed one of the cost terms at a time. An MIP model was generated to validate the findings of the DP-based search for a simplified plant model. The results indicate that the search algorithm indeed finds a near-optimal solution.

The DP-based search is prohibitive in its computational requirements when horizons longer than one week are considered. A faster heuristic is developed: The plant model used in the MIP algorithm captures the main aspects of the load-shifting behavior of the TES

system. The optimal limits of the peak power consumption as found by MIP are used to lock into the proximity of the limits found by a full DP-based search and the realistic plant model. Significant savings in run time are achieved by searching around the MIP optimal limits with a particular search radius r . The closer the plant models used in the MIP and DP routines are, the smaller the value of r can be. This second heuristic will be even more beneficial when long time horizons (months) have to be investigated since only a relatively short horizon is necessary to find good choices of the limits.

9. Acknowledgement

This work is partially funded by the American Society of Heating, Refrigerating, and Air-Conditioning Engineers (ASHRAE) under contract number 809-RP.

10. References

- H. Akbari and O. Sezgen (1992) *Case Studies of Thermal Energy Storage (TES) Systems: Evaluation and Verification of System Performance*. LBL-30852. Lawrence Berkeley Laboratory, University of California, Energy and Environment Division.
- CPLEX 3.0, (c) 1994 Cplex Optimization Inc.
- M. Krarti, M.J. Brandemuehl and G.P. Henze (1995) "Second Project Report for ASHRAE 809-TRP: Evaluation of Optimal Control for Ice Storage Systems." *ASHRAE Final Report*
- L.J. White, C.M. McVicker and E. Stiles (1993) *Electric and Gas Rates for the Residential, Commercial and Industrial Sectors: 1993* 1 and 2, Gas Research Institute.
- WCDSR (1994) *The Commercial Cooling Load Library*. Wisconsin Center for Demand-Side Research. Madison, WI.

Exploiting Block Structure to Improve Resource-Constrained Project Schedules

Helmut E. Mausser

Stephen R. Lawrence

Graduate School of Business Administration

University of Colorado

Boulder, CO

80309-0419

E-mail: mausserh@ucsu.colorado.edu

stephen.lawrence@colorado.edu

Abstract:

We identify a block structure, typically found in resource-constrained project schedules, that yields a natural decomposition of the original problem into a series of smaller subproblems. Reducing the makespan of any subproblem has the effect of reducing the total makespan by an equivalent amount. Since these subproblems can be independently reoptimized, this block structure provides a way of improving schedules generated by heuristic methods. We report computational results of applying this technique to problems containing up to five resources and on the order of 200 activities, and speculate as to its applicability to more powerful local search based heuristics.

Key Words: Heuristics, resource-constrained project scheduling, block decomposition.

1. Introduction

In practice, resource-constrained project scheduling is typically addressed using heuristic methods, which seek to find good solutions in a reasonable amount of time. Since such methods are unlikely to provide an optimal schedule, techniques for improving heuristic solutions can have a significant impact on practical scheduling applications. In this paper, we show that resource-constrained project schedules often contain blocks of activities that decompose the original problem into a series of smaller subproblems. These subproblems can be independently rescheduled in an effort to improve the overall project schedule. Furthermore, this block structure may facilitate the use of local search heuristics for resource-constrained problems.

The management of projects subject to the availability of a finite set of resources is important in the operation of many organizations. For example, in completing a construction project, a contractor must allocate limited labor and equipment among the various tasks, or activities, to be performed. While techniques such as the Critical Path Method (CPM) and the Program Evaluation and Review Technique (PERT) can account for precedence relationships among activities, they do not consider finite resources that may prevent some activities from being performed concurrently. The resource-constrained project scheduling problem (RCPSP) generalizes the job-shop scheduling problem and is thus NP-hard (Blazewicz *et al.* 1983).

Resource-constrained project scheduling can take a variety of forms, depending on resource and activity characteristics, and choice of objective function. In this paper, we consider multi-project problems in which resources are renewable in each period, activities are nonpreemptive and use resources at a constant rate, and the objective is to minimize makespan. As we discuss in Section 5, reducing project makespan will also improve any other regular measure of schedule performance. Minimizing makespan also permits us to treat single-project and multi-project problems equivalently.

RCPSP has received much attention from researchers (see surveys by Davis 1973, and Özdamar and Ulusoy 1993). While exact methods can optimally solve small instances of RCPSP (see, for example, Patterson 1984), the large size of practical problems has motivated the development of heuristic procedures. Some common priority-based dispatching rules are described in Lawrence (1985) and Alvarez-Vald  ez and Tamarit (1989). Serial heuristics assign priorities to all activities before constructing the schedule, while parallel heuristics determine priorities during scheduling (Kelly and Walker 1963). An additional distinction relates to whether priorities are assigned based on a measure of time or resource usage.

Previous studies have found parallel, time-based heuristics to be most effective in general (Cooper 1976, Davis and Patterson 1975, Davis 1973) and we exclusively use the parallel method of dispatch scheduling in this paper. The fact that no one single heuristic consistently yields the best results has motivated further research to improve the robustness of the heuristic approach. One strategy attempts to select the most promising heuristic in advance based on the problem characteristics (Patterson 1976, Ulusoy and Özdamar 1989). Other alternatives include applying a set of heuristics to the problem and selecting the best solution (Boctor 1990), or using a multi-pass procedure, as in the iterative forward/backward algorithm of Li and Willis (1992) and lead time iteration (Lawrence and Morton 1993).

We are not aware of any previous attempts to exploit the underlying structure of a schedule in order to obtain an improved solution. In particular, this paper identifies a block structure that allows a large problem to be easily decomposed into a series of smaller subproblems that can then be reoptimized in an effort to reduce the makespan. While we consider only priority-based dispatch rules in this study, the block structure may facilitate the implementation of more powerful local search heuristics, such as those typically used within a tabu search framework.

The paper is organized as follows. In Section 2, we identify the block structure and explain its appeal for improving existing schedules. Section 3 describes our implementation of the improvement procedure, and the results of a preliminary study involving dispatch rules are reported in Section 4. Finally, Section 5 suggests future avenues for research, with a particular emphasis on the applicability of block structure to metaheuristic methods such as tabu search.

2. Block Structure

Let the scheduling horizon consist of the set of time periods $\{1, 2, \dots, T\}$ and represent the start and finish time of activity i by s_i and f_i , respectively. We define a *block* to be a contiguous set of time periods that completely contains all activities occurring within it (no activity straddles the boundary between two blocks). Formally, periods t_1 through t_2 ($t_2 \geq t_1$) constitute a block if, for each activity i processed between period t_1 and t_2 , $s_i \geq t_1$ and $f_i \leq t_2$. For example, Figure 1 shows a schedule containing three blocks: the first extends from period 1 to 8 and contains activities $\{1, 2, 3, 4, 5\}$, the second extends from period 9 to 16 and contains activities $\{7, 9, 10, 11, 12, 14\}$, and the third extends from period 17 to 20 and contains activities $\{6, 8, 13\}$.

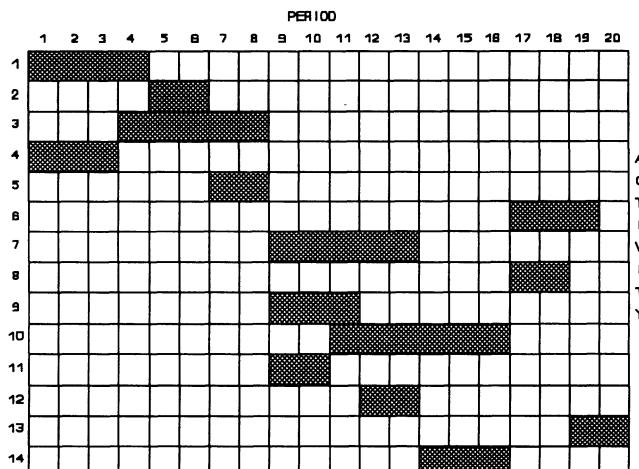


Figure 1: Project schedule containing 3 blocks.

The fact that this structure occurs naturally in project schedules may be somewhat surprising, but we have found that approximately 97% of the over 200 schedules examined, including both randomly generated instances and examples from the literature, contain multiple blocks (20 or more blocks per schedule is quite common for 200-activity problems). Thus, we expect that any procedure able to exploit this structure would have widespread applicability. We also mention that it remains to determine how block structure is affected by various problem characteristics, such as the average level of resource usage by activities. For instance, one might expect that as activities use smaller portions of the available resources, more of them can be processed concurrently, leading to more overlap among activities and hence decreasing the number of blocks. This is an area we are actively investigating.

An attractive feature of this structure is that each block can be considered independently of all others for scheduling purposes. To see this, note that all precedence constraints between activities within the block (*internal* activities) and those outside the block (*external* activities) remain satisfied regardless of the sequence assigned to the internal activities. Furthermore, since the project makespan is simply the sum of the individual block lengths, reducing the makespan of any one block effectively shortens the entire schedule by an equivalent amount. These facts suggest that it is reasonable to try to improve existing schedules by rescheduling individual blocks, a procedure we will call *block improvement*. Several related observations tend to support this claim.

Firstly, the block structure provides a natural decomposition of the schedule into smaller, more manageable pieces. One would expect that in many cases, the blocks would be of a size (in terms of number of activities) that would permit them to be reoptimized by exact algorithms. Thus, it may be possible to obtain a type of "local optimality" relative to the existing block structure.

Alternatively, heuristic scheduling procedures can be applied to the block subproblems. This approach holds particular promise since the aforementioned studies indicate a connection between problem characteristics and heuristic performance; a particular block, produced by a certain heuristic, could well exhibit characteristics that are better suited to another. This effectively provides a means of combining several different heuristics in the production of a single schedule, something that has received relatively little attention in the past. Also, heuristics that assign priorities based on all remaining unscheduled activities may in fact make locally suboptimal decisions. Thus, applying the same heuristic to a subproblem, and thereby using only local information, may result in an improved solution.

Attention need not be restricted to the smallest blocks in the schedule, which we will refer to as *minimal* blocks. In fact, adjacent minimal blocks can be combined into *compound* blocks to produce larger subproblems that can still be rescheduled independently. We will use B_i to refer to the i^{th} minimal block and $[B_i, B_j]$ to refer to the compound block consisting of B_i, B_{i+1}, \dots, B_j . For example, in Figure 1, $[B_1, B_2]$ extends from period 1 through period 16. By utilizing this type of "dynamic" restructuring together with successive rescheduling of the redefined blocks, it is possible to obtain significant movement of activities throughout the schedule. Thus, this approach may in fact yield solutions that differ significantly from the original schedule. Since blocks are independent for scheduling purposes, multiple blocks can be reoptimized concurrently. This suggests opportunities for making use of parallel processing.

Finally, this approach can be applied to any schedule exhibiting the appropriate block structure, regardless of how the schedule was obtained. Furthermore, the above observations suggest that it may well have a greater impact as problem size increases. Thus, block improvement shows potential as a type of "post processing" step that can be used in conjunction with other methods.

3. Implementation of Block Improvement

To implement the block improvement procedure, we first obtain an initial schedule by applying a set of heuristics to the problem and selecting the solution having minimum makespan (currently, we arbitrarily select one such solution if more than one heuristic yields an identical minimum length schedule; we are investigating ways to identify good candidate schedules). The initial solution is then decomposed into a set of minimal blocks. If the entire schedule comprises a single minimal block, then no improvement is attempted. Otherwise, we make several passes through the schedule, increasing the number of blocks per subproblem in each pass.

The first pass reschedules each minimal block in succession (i.e. each subproblem consists of a single block). Blocks containing only one activity are skipped, since no improvement is possible in this case. If rescheduling does not reduce the makespan, then the current project schedule is left unchanged. Otherwise, the new solution is decomposed into blocks, resulting in an updated block structure, and the improvement procedure continues with the first newly-created block. The first pass ends when all minimal blocks have been rescheduled without improvement.

As noted previously, it is possible to combine adjacent blocks in order to create larger subproblems. Thus, subsequent passes reschedule combinations of two, three, and four adjacent minimal blocks. While it is possible to continue in this manner, many of the resulting subproblems quickly become rather large. Of particular interest, however, is a long contiguous set of blocks, each of which contains a small number of activities. Thus, we make one final pass that combines the maximum number of adjacent minimal blocks possible to yield a subproblem of at most 30 activities. If the resulting compound block contains more than four minimal blocks, the subproblem is rescheduled. For example, given the block structure shown in Figure 2, the final pass would reschedule compound blocks [4,9] and [5,11]

(note that [6,11] is skipped since it is contained in [5,11]). The choice of a maximum of four contiguous blocks and 30 activities was based on the results of preliminary experiments. We are currently investigating the robustness of this result.

1 (17)	2 (21)	3 (12)	4 (8)	5 (4)	6 (6)	7 (3)	8 (7)	9 (1)	10 (5)	11 (2)	12 (14)
-----------	-----------	-----------	----------	----------	----------	----------	----------	----------	-----------	-----------	------------

Figure 2: Sample block structure showing block number (and number of activities).

As in the first pass, when reoptimization results in an improved solution (and a modified block structure), the improvement procedure is applied recursively to that portion of the schedule affected by the new block structure. This process is most easily described with the help of a specific example (Figure 3). Suppose that the current schedule consists of blocks 1 through 8 and that we have improved the makespan of the subproblem corresponding to compound block [3,5] (i.e. we are on the third pass through the schedule), resulting in the new block structure $[1^*, 4^*]$ for the subproblem. The improvement procedure then continues with the following sequence of subproblems: $[1^*, 1^*]$, $[2^*, 2^*]$, $[3^*, 3^*]$, $[4^*, 4^*]$, $[2, 1^*]$, $[1^*, 2^*]$, $[2^*, 3^*]$, $[3^*, 4^*]$, $[4^*, 6]$, $[1, 1^*]$, $[2, 2^*]$, $[1^*, 3^*]$, $[2^*, 4^*]$, $[3^*, 6]$, $[4^*, 7]$, $[6, 8]$, $[1, 2^*]$, etc.

Initial Block Structure

1	2	3	4	5	6	7	8
---	---	---	---	---	---	---	---

Modified Block Structure

1	2	1*	2*	3*	4*	6	7	8
---	---	----	----	----	----	---	---	---

Figure 3: Modified block structure due to improved solution.

4. Computational Experiment

As an initial test of the block improvement procedure, we applied it to four sets of randomly generated test problems. We restricted our attention to the use of heuristic procedures for generating the initial schedule and subsequently searching for improved solutions.

4.1 Experimental Design

We conducted preliminary tests to identify a suitable set of dispatch heuristics, listed in Table 1. At each step, the rules consider only those activities eligible for dispatching (i.e. precedence and resource feasible). In all cases, ties are broken by a first come first served rule. Lead time iteration updating (Lawrence and Morton 1993) is used whenever applicable, with smoothing parameter $\alpha = 0.1$ and a stopping criterion of 10 successive iterations without improvement.

We generated four sets of 20 problems with 1, 2, 3, and 5 resources respectively. Each problem consisted of 5 projects (again, the number of projects is irrelevant for a makespan objective), with the number of activities per project uniformly distributed on the interval [25,50]. Activity processing times were uniform [1,10] integers. Each resource had an availability of 100 units per period and resource utilization was uniformly distributed on [10,100] for each activity. The number of resources required by each activity was uniform [1, K], where K was the total number of resources, except for $K = 5$, in which case the number of resources required was uniform [1,4].

4.2 Experimental Results

The results for each set of problems appear in Table 2. In the table, the improvement percentages are calculated as follows:

$$\% \text{ improvement} = 100 * \frac{\text{old makespan} - \text{new makespan}}{\text{old makespan}}$$

Table 1: Priority-based dispatch rules.

Dispatch Rule		Activity Selection
BIN	Bin Packing ¹	$\max \{ r_{ik} \mid 1 \leq k \leq K \}$
CR	Critical Ratio	$\min \left\{ \frac{D - t}{D - l_i} \right\}$
GRD	Greatest Resource Demand	$\max \left\{ p_i \sum_k r_{ik} \right\}$
GRU	Greatest Resource Utilization ²	$\max \left\{ \sum_k r_{ik} \right\}$
LFT	Late Finish Time	$\min \{ l_i + p_i \}$
MINSLK	Minimum Slack	$\min \{ l_i - t \}$
ODD	Operation Due Date	$\min \{ l_i^* + p_i \}$
RM	Rachamadugu & Morton ³	$\max \left\{ \frac{1}{p_i} \exp \left(- \frac{(l_i - t)^+}{\beta \bar{p}} \right) \right\}$
RMP	Rachamadugu & Morton Pricing ⁴	$\max \left\{ \frac{1}{\pi_i} \exp \left(- \frac{(l_i - t)^+}{\beta \bar{p}} \right) \right\}$

D - due date (set equal to block length in all cases)

K - number of resources

l_i - late start time for activity i (working backward from $\max(D, L)$)

l_i^* - late start time for activity i (working backward from D)

L - critical path length

p_i - processing time of activity i

\bar{p} - average processing time of all activities

r_{ik} - amount of resource k required by activity i

t - current time period

β - lookahead parameter (see Lawrence and Morton 1993)

π_i - imputed resource cost of activity i (see Lawrence and Morton 1993)

¹ This is the first fit decreasing heuristic of Garey et al (1976).

² This rule implements a knapsack heuristic to maximize resource utilization.

^{3,4} See Lawrence and Morton (1993). We use bottleneck resource pricing and set $\beta = 3.5$.

Table 2 allows us to propose two conjectures. First, as noted previously, the block structure is common; multiple blocks occurred in 78 out of the 80 test problems. Secondly, as the number of resources increases, it becomes more difficult to improve an initial solution but the resulting improvement becomes more significant, both in terms of the average and maximum case. While the average improvement may appear to be small, it is important to remember that the initial solutions can be expected to be quite good, particularly since we select the best solution from among a set of heuristics. As a matter of reference, previous studies (Davis and Patterson 1975, Alvarez-Vald  z and Tamarit 1989,   zdamar and Ulusoy 1993) found that, on average, MINSLK and LFT gave results that were within 3.41% to 6.7% of the optimal solution. Furthermore, we anticipate that block improvement will provide larger percentage benefits with other scheduling objectives, such as weighted tardiness.

Table 2: Computational results.

Resources	Avg. No. Activities	Avg. Initial Makespan	Problems with Multiple Blocks	Problems Improved	Avg. (%) Improvement ¹	Max. (%) Improvement
1	185.1	583.8	19	19	0.82	2.12
2	186.3	567.6	19	15	0.90	2.48
3	182.7	578.9	20	14	0.93	3.03
5	187.3	390.5	20	11	1.56	3.97

¹ Averaged over the problems that were improved.

We have also used an exact approach, the branch-and-bound algorithm of Demeulemeester and Herroelen (1992), to solve a selected number of subproblems. The results to date have been somewhat mixed; the subproblems contain relatively few precedence constraints, thus assuming more of a bin packing flavor, and making it difficult to fathom partial solutions. We are currently further examining both heuristic and exact techniques, specifically designed for this type of problem, with the goal of incorporating them into future improvement procedures.

We do not report computational time for the improvement procedure since our current implementation makes extensive use of files in order to transfer data and consumes a disproportionate amount of time performing input/output operations.

5. Extensions and Future Research

Our experimental results indicate that the block structure present in project schedules can be exploited by heuristic methods. We now speculate as to its relevance for other versions of RCPSP, possible enhancements to the improvement procedure, and its potential for facilitating more complex solution approaches.

We have considered only the makespan criterion in this paper. Although a simple measure, reducing makespan will improve any regular measure of scheduling performance (Baker 1974). Consequently, our focus on makespan is a logical first step in investigating the benefits of block improvement. Makespan improvement will not be appropriate for non-regular measures such as early-tardy or net present value objectives, for which delays in activity start times can often improve schedule performance. The application of block improvement concepts to non-regular measures of performance is a potentially productive area for future research.

An interesting opportunity is to extend the scope of the rescheduling process beyond the particular subproblem under consideration. Specifically, one could keep track of resource slack in each block and then, while rescheduling, identify activities that could "fit" into another block (subject to precedence constraints). Moving such activities would increase the likelihood of improving the makespan of the current subproblem. This has particular relevance for the latter blocks in the schedule, which typically have low resource utilization rates for schedules constructed in a forward manner (Li and Willis 1992).

Block structure holds promise for more powerful metaheuristics such as tabu search. A "move" typical of many local search methods would change the position of a selected activity in the schedule. Unlike problems in which feasible solutions consist of permutations (such as the TSP), making it is easy to retain feasibility, in RCPSP moving an activity requires considerable adjustment in terms of:

- 1) creating a "gap" in the resource profile in which to insert the activity
- 2) filling the "gap" left by the activity that has been moved

In both cases, a substantial number of activities may need to be rescheduled before the move can be evaluated.

The block structure provides a natural way to limit the scope of such adjustments. For example, suppose that we want to remove activity i from block B_m . If we record, for each activity, the block in which it is contained, then the precedence requirements identify a "block window" in which i can be placed with minimal adjustment to the schedule. Figure 4 gives an example in which activity 81 is to be removed from block 8. Since its closest predecessor (63) and successor (117) are in blocks 6 and 12, respectively, we can move 81 to any block B_n ($6 \leq n \leq 12$) by simply rescheduling blocks 8 and B_n to recover feasibility. Of course, this approach only allows an activity to be placed in a particular block; one cannot specify the period in which the activity is to begin.

Act.	...	45	...	63	...	81	...	117	...
Block	...	3	...	6	...	8	...	12	...

Activity 81 has predecessors {45, 63} and successor {117}.

Figure 4: Local search example.

Alternatively, as part of the rescheduling process, we could look for "influential" activities whose movement to another part of the

schedule could drastically alter the solution structure. Possible characteristics of influential activities include large resource requirements, delaying many successors, or having a large block window. Such moves may help to "break apart" existing structure and move us to other areas of the search space.

Acknowledgements

We thank Erik Demeulemeester for providing the branch-and-bound code and Peter Brucker for helpful discussion. The first author was supported by a Gerald Hart Doctoral Research Fellowship.

6. References

- R. Alvarez-Vald  z and J.M. Tamarit, Heuristic Algorithms for Resource-Constrained Project Scheduling: A Review and an Empirical Analysis, in: *Advances in Project Scheduling*, eds. R. S  wi  ski and J. W  glarz (Elsevier Science Publishers, Amsterdam, 1989) p. 113.
- K.R. Baker, *Introduction to Sequencing and Scheduling*, (Wiley, New York, 1974).
- J. Blazewicz, J.K. Lenstra and A.H.G. Rinnooy Kan, Scheduling Subject to Resource Constraints: Classification and Complexity, *Discrete Applied Mathematics*, 5 (1983) 11.
- F.F. Boctor, Some Efficient Multi-Heuristic Procedures for Resource-Constrained Project Scheduling, *European Journal of Operational Research* 49 (1990) 3.
- D.F. Cooper, Heuristics for Scheduling Resource-Constrained Projects: An Experimental Investigation, *Management Science* 22 (1976) 1186.
- E.W. Davis, Project Scheduling under Resource Constraints - Historical Review and Categorization of Procedures, *AIEE Transactions* 5 (1973) 297.
- E.W. Davis and J.H. Patterson, A Comparison of Heuristic and Optimum Solutions in Resource-Constrained Project Scheduling, *Management Science* 21 (1975) 944.

- E. Demeulemeester and W. Herroelen, A Branch-and-Bound Procedure for the Multiple Resource-Constrained Project Scheduling Problem, *Management Science* 38 (1992) 1803.
- M.R. Garey, R.L. Graham, D.S. Johnson and A.C.C. Yao, Resource Constrained Scheduling as Generalized Bin Packing, *Journal of Combinatorial Theory (A)* 21 (1976) 257.
- J. Kelly and M. Walker, Scheduling Activities to Satisfy Resource Constraints, in: *Industrial Scheduling*, eds. J. Muth and G. Thompson (Prentice Hall, Englewood Cliffs, NJ, 1963).
- S.R. Lawrence and T.E. Morton, Resource-Constrained Multi-Project Scheduling with Tardy Costs: Comparing Myopic, Bottleneck, and Resource Pricing Heuristics, *European Journal of Operational Research* 64 (1993) 168.
- S.R. Lawrence, A Computational Comparison of Heuristic Scheduling Techniques, Working paper, Graduate School of Industrial Administration, Carnegie-Mellon University (1985).
- K.Y. Li and R.J. Willis, An Iterative Scheduling Technique for Resource-Constrained Project Scheduling, *European Journal of Operational Research* 56 (1992) 370.
- L. Özdamar and G. Ulusoy, A Survey on the Resource-Constrained Project Scheduling Problem, *IEE Transactions* (to appear).
- J.H. Patterson, A Comparison of Exact Approaches for Solving the Multiple Constrained Resource, Project Scheduling Problem, *Management Science* 30 (1984) 854.
- J.H. Patterson, Project Scheduling: The Effects of Problem Structure on Heuristic Performance, *Naval Research Logistics Quarterly* 23 (1976)
- G. Ulusoy and L. Özdamar, Heuristic Performance and Network/Resource Characteristics in Resource-Constrained Project Scheduling, *Journal of the Operational Research Society* 40 (1989) 1145.

Combining the Large-Step Optimization with Tabu-Search: Application to The Job-Shop Scheduling Problem

Helena Ramalhinho Lourenço

Departamento de Estatística e Investigação Operacional

Faculdade de Ciências

Universidade de Lisboa

Campo Grande C/2

1700 Lisboa, Portugal

e-mail: helena@gist.deio.fc.ul.pt

Michiel Zwijnenburg

Department of Econometrics

Faculty of Economy

Erasmus University

Rotterdam, The Netherlands

Abstract:

We apply the combined technique of tabu-search and large-step optimization to the job-shop scheduling problem. The job-shop scheduling problem can be defined as follows: given a set of machines and a set of jobs, the objective is to construct a schedule which minimizes the time necessary to complete all the jobs. We also present some diversification strategies for the tabu-search methods and relate them with the large-step optimization method. Relevant computational results and respective conclusions are also presented.

Key Words: Local Search, Tabu Search, Large-Step Optimization, Job-Shop Scheduling.

1. Introduction

We apply the combined technique of tabu-search and large-step optimization to the job-shop scheduling problem. In the job-shop scheduling problem we are given a set of m machines $\{M_1, \dots, M_m\}$ and a set of n jobs $\{J_1, \dots, J_n\}$. Each job J_j , $j=1, \dots, n$, consists of a sequence of m_j operations $O_{1j}, \dots, O_{mj,j}$, where O_{ij} is an operation of job J_j to be processed on machine M_i for a given uninterrupted processing time p_{ij} , where $\mu_{ij} \neq \mu_{i+1,j}$, for $i=1, \dots, m_j$, $j=1, \dots, n$.

The operations of each job must be processed in the given sequence. Each machine M_i , $i=1, \dots, m$, can process at most one operation at a time, and at most one operation of each job J_j , $j=1, \dots, n$, can be processed at a time. Let C_{ij} be the completion time of operation O_{ij} . The objective is to get a schedule that minimizes the maximum completion time $C_{\max} = \max_{i,j} C_{ij}$. A schedule is an allocation of a single time interval for each operation. A more useful representation is provided by the disjunctive graph model that can be represented as follows: $G=(O, A, E)$, where O is the vertex set corresponding to the operation set, A is the arc set corresponding to the job precedence constraints and E is the edge set corresponding to the machine capacity constraints.

In the disjunctive graph, the basic scheduling decision corresponds to orienting each edge in one direction or the other. A schedule is obtained by orienting all edges. The orientation is feasible if the resulting directed graph is acyclic, and its length is equal to the weight of a maximum weight path from s to t in this acyclic graph.

The job-shop scheduling problem is NP -hard in the strong sense. Furthermore, most of the special cases of this problem are NP -hard or strongly NP -hard.

In Section 2, we review local optimization methods, as local improvement and simulated annealing, and a two-phase optimization method, known as large-step optimization, which has recently been introduced for the traveling salesman problem. The first phase of this new method consists of a large optimized transition in the current solution, while the second phase is basically a local search method. The

main advantage of the large-step optimization methods is the use of optimization and tailored procedures in combination with local search methods.

So far, the methods used in this second phase, also called small-steps phase, are the local improvement and the simulated annealing methods. In this work we combine the large-step optimization with a tabu-search approach and apply it to the job-shop scheduling problem, because both methods separately applied to the problem gave good results. We present this combined method and related computational results in Section 3.

In Section 4, we present some diversification strategies for the tabu-search methods and relate them with the large-step optimization method. Relevant computational results are also presented. In Section 5, we present some conclusions and possible future research ideas.

2. Local and large-step optimization methods

Local improvement methods are iterative methods where initially a feasible solution of the problem in question is obtained, and at each step we make one local modification, or transition, of a prespecified type in the current solution. If an improvement in the cost function is obtained, then we accept the new solution as the current one, and, otherwise, we reject it. The algorithm terminates when we cannot improve the current solution by performing one transition, i.e., when we have a local optimal solution.

Consider now the job-shop scheduling problem. A solution corresponds to a schedule and the cost function is the maximum completion time of the schedule. The method used to obtain the initial schedule is the priority rule with priority function most work remaining. An extensive study considering different methods to obtain initial schedule has been done by Lourenço (1995).

Simulated annealing accepts solutions with increased value for the cost function, called uphill moves, with small and decreasing probability, in an attempt to get away from a local optimal solution and keep exploring the region of the feasible solutions. The probability is controlled by a parameter known as the temperature, which is

gradually reduced from a high value, at which most uphill moves are accepted, to a low one at which few, if any, such moves are accepted.

In our implementation of simulated annealing, we use the same method to generate the initial schedule as for the local improvement method, and the neighborhood structure of van Laarhoven, Aarts and Lenstra (1992) (neighborhood VLA&L), but we use a different cooling schedule. Our cooling schedule is very similar to the geometric cooling schedule presented in Johnson et al.(1989). Later on, we also consider a version of the simulated annealing method using the neighborhood structure developed by Dell'Amico & Trubian (1993), (neighborhood D&T).

Martin, Otto and Felten (1992) introduced a large-step optimization method for the traveling salesman problem, which was applied to the job-shop scheduling problem by Lourenço (1993). The large-step optimization method consists of three main routines: a large-step phase, a small-steps phase and an accept/reject test, which all three are consecutively executed for a certain number of large-step iterations.

Local improvement methods proceed downhill for a while, making good progress, but they tend to get trapped in local optimal solutions, usually far away from the global optimal solution. Simulated annealing methods try to improve this by accepting uphill moves depending on a decreasing probability controlled by the temperature parameter. But, at small temperatures, they also tend to get stuck in valleys of the cost function. Large-step optimization methods allow to leave these valleys even at small temperatures, and only then a local search optimization method is applied, returning to a local optimal solution. At this point, an accept/reject test is performed.

The three main routines of a large-step optimization method are: the method to perform a large step, the one for the small steps and the accept/reject test (see Figure 1).

An important part of the large-step optimization methods is the large-step phase. The large step phase should be constructed so that valleys are easily climbed over, and they should be specially tailored to the problem in consideration. As mentioned, this procedure should

make a large modification in the current solution to drive the search to a new region and, at the same time, to perform some kind of optimization to obtain a solution not too far away from a local (or global) optimal schedule. These two factors of the large-step method are the main differences from usual local search methods.

1. Get an initial schedule s_0 .
2. Perform the following loop $numiter$ times:
 - 2.1. Large step phase: change s_i to s'_i by performing a big perturbation.
 - 2.2. Small steps phase: run a local search optimization method with initial solution s'_i , and obtain s''_i .
 - 2.3. Perform an accept/reject test comparing s_i with s''_i , if s''_i is accepted, let $s_{i+1}=s''_i$.
3. Return the best solution found s_{best} .

Figure 1: Large-step optimization method.

So far, the methods that are usually associated with the small steps phase are local optimization algorithms like the local improvement method or the simulated annealing method. In the next section, we will consider the tabu-search approach as the small-steps procedure.

The accept/reject test can accept only downhill moves. This looks like the local improvement methods presented before, but where only local optimal solutions (with respect to the small steps) are considered. On the other hand, the accept/reject test can accept all moves or it can also look like the test done in a simulated annealing method, i.e. all downhill moves are accepted and a uphill move is accepted with a small and decreasing probability.

Different applications of the large-step optimization methods have appeared in the literature. Martin, Otto and Felten (1992) developed a large-step optimization methods for the traveling salesman problem, where they used a 4-opt move as the large step phase, and, as the small steps, a local improvement method based on 3-opt moves. Johnson (1990) also applies a large-step local optimization method, that he called iterated Lin-Kernighan, to the traveling salesman problem, using

a 4-opt move as the large-step and the Lin and Kernighan heuristic for the small steps. Lucks (1992) applied large-step optimization to the graph partitioning problem, where the large step is done by interchanging a randomly selected set of vertices of a given size, and, as small steps, the Kernighan and Lin approximation method and a simulated annealing approach were used. He concluded that for random graphs, large-step optimization outperformed all other tested algorithms, while for geometric graphs, the large-step method in some cases was outperformed. Brucker, Hurink & Werner (1993),(1994), presented a version of the large-step methods, designated as improving local search heuristics, for some scheduling problems.

The large step optimization is different from random restarts methods, Feo and Resende (1994), in which randomly constructing an initial schedule followed by a local optimization method is repeated for a number of iterations, returning the best solution found. As suggested by Johnson (1993), large-step optimization methods can be viewed as restart methods. But instead of starting with a different initial random solution at each iteration, in the large step a solution s' is obtained from a solution s , by performing a sufficient large optimized perturbation. In this way, all the achievements from previous runs are not totally lost in the next runs. After every large step a local optimization method is applied as a small steps procedure which leads the search process to a good local optimal solution. Also, these methods have the advantage of combining the ideas associated with local search methods and tailored heuristics.

Next, we apply the large-step optimization methods to the job-shop scheduling problem. For the small steps phase, we have considered the following methods: the local improvement method and the simulated annealing method. Lourenço (1995) proposed four methods to perform the large step procedure. The first three methods are based on choosing one or two machines and reordering the operations to be processed on these machines, using some prespecified method, and the last one is based on reversing the order of processing of several operations. We use one of the most successful which can be briefly described as follows: reschedules iteratively two randomly chosen machines to optimality. First choose two random machines and ignore the order of the operations to be processed in these machines. Next, for one of the

two machines, determine the release and delivery times of the operations to be processed by the respective machine, given the scheduled operations on the other machines. Solve the one-machine schedule associated with this machine using Carlier's algorithm, Carlier (1982). Then repeat this process to reschedule the second machine. For more information, see Lourenço (1995).

Next, we present computational results obtained from the application of these methods to several instances of the problem (see Tables 1 to 4). The methods are tested on set of known instances. Since the amount of testing is extensive, in preliminary tests we use just a subset of the instance. All programs were written in C. For instances of size 10 jobs, 10 machines, and smaller, the programs run in a Shine 486DX with a clock of 66 Mhz, and for the bigger instance the programs ran on a SPARC SUN 4M.

For the job-shop scheduling problem, Lourenço (1995) concluded that the local improvement method is clearly inferior to the simulated annealing, even when several trials are performed. Also the large-step method combined with the local improvement gave no better results than when we combined the large-step with the simulated annealing. Therefore we do not consider the local improvement in our computational experience. The large-step optimization methods outperformed the simulated annealing method, but the difference between these two became closer when the same amount of time was allowed for both. In many cases, the large-step optimization methods found an optimal schedule and in others the distance from the optimum or lower bound was small.

Given the results obtained by previous applications of the large-step optimization methods to combinatorial optimization methods, including the job-shop scheduling problem, and given the success of tabu-search to the same problem, it looks promising to apply the large-step optimization method using in the small steps phase a tabu-search method.

3. Combined techniques of large-step optimization and tabu-search

The tabu-search, in contrary of local optimization methods presented previously, makes use of historical information. The historical information is kept in three kinds of memory functions: the short-term, the intermediate and long term memory function. The last two will be considered in next section. Short term memory function has the task to memorize certain attributes of the search process of the recent past and it is incorporated in the search via one or more tabu list. For more information on tabu-search methods see, for example, Glover (1989) and Glover et al.(1993).

The application of tabu search to the job-shop scheduling problem by Dell'Amico and Trubian (1992) gave very good results, taking a small amount of time. Note that a big percentage of the running time for a large-step optimization method is spent by the simulated annealing method. Therefore, an improvement for large-step optimization methods should be achieved by using small-step methods that give good local optimal solutions and run in a shorter amount of time than the simulated annealing. By the above observations the tabu search methods looks like a good candidate to be used instead of the local improvement method, that are fast but gives poor local optimal solutions, and instead of simulated annealing that gives good local optimal solutions, but at very high computational cost.

The basis for our implementation of the tabu-search is derived from Dell'Amico and Trubian (1993), but there are some differences. Dell'Amico and Trubian used a bi-directional method to obtain an initial schedule, while our tabu-search, like in the local improvement method and simulated annealing approach, starts by constructing a initial schedule according to the priority rule most remaining work. In our implementation we did not apply their intensification strategy which let the process return to the best solution found so far in the process, if during a certain number of iterations the best solution did not improve. The reason for this is because we plan to use the tabu-search in the small-steps phase of the large-step optimization method. Also, different in our implementation is the use of exact methods for calculating the longest path in a graph belonging to a feasible solution and for checking if a graph contains a cycle. Similar to Dell'Amico and

Trubian tabu-search are the neighborhood structure, the tabu-list structure and the rules which define the length of the tabulist.

The parameters in all programs are set in such a way that for a certain instance, they use the same amount of computer time as our implementation of the Dell'Amico and Trubian tabu-search method needed for 12000 iterations for the same instance. For example, for instance MT10 this took more and less 1550 seconds on a Shine 486DX with clock 66 Mhz.

Table 1: Best results for the Local Optimization Methods with VLA&L neighborhood structure.

Neighborhood: VLA&L		Size	SA	TS	LSSA	LSTS
Instance	Opt	n*m	best	best	best	best
ABZ5	1234	10*10	1239	1236	1244	1238
CAR5	7702	10*6	7822	7732	8305	7702
LA04	590	10*5	590	590	590	590
LA21	1046	15*10	1058	1056	1069	1053
MT10	930	10*10	949	930	956	940
ORB3	1005	10*10	1040	1020	1051	1005
ORB4	1005	10*10	1026	1019	1026	1011

Table 2: Average results for the Local Optimization Methods with VLA&L neighborhood structure.

Neighborhood: VLA&L		Size	SA	TS	LSSA	LSTS
Instance	Opt	n*m	av.	av.	av.	av.
ABZ5	1234	10*10	1247	1237	1245	1239
CAR5	7702	10*6	8019	7943	8476	7924
LA04	590	10*5	591	590	591	590
LA21	1046	15*10	1080	1060	1077	1060
MT10	930	10*10	960	945	973	942
ORB3	1005	10*10	1060	1027	1064	1015
ORB4	1005	10*10	1035	1022	1030	1013

The results are presented in Table 1, 2, 3 and 4. We can conclude that the large-step optimization combined with the tabu-search outperforms the simulated annealing and the large-step optimization using simulated annealing. The average value out of 5 runs for the large-step optimization with tabu-search is always smaller than the average results for the large-step optimization with simulated annealing. In the beginning of the search in the simulated annealing approach there is no fast improve as in the tabu-search. Therefore the simulated annealing wastes time in the beginning of the process, meanwhile this initial period is effectively used by tabu-search. Even the tabu-search alone outperforms the large-step optimization method with simulated annealing small-steps procedure but it is slightly inferior to the large-step optimization with tabu-search. For example, for the famous instance of Muth and Thompson with 10 jobs and 10 machines (MT10) and considering the D&T neighborhood, the best value obtained by the large-step optimization method with tabu search (LSTS) was the optimal value of 930 and the average was 939 meanwhile if the simulated annealing (LSSA) was used as the small-steps procedure the respective values were 951 and 963. Using our implementation of the tabu-search (TS) alone we obtained the following values, 940 was the best obtained and 946 the average. For the instance propose by Lawrence (LA21) the best value obtained by LSTS was 1047, and the average was 1055. While for the LSSA the respective values were 1078 and 1096. Applying the TS method alone we obtained the following the values 1059 and 1065.

Table 3: Best results for the Local Optimization Methods with D&T neighborhood structure.

Neighborhood: D&T		Size	SA	TS	LSSA	LSTS
Instance	Opt/LB	n*m	best	best	best	best
ABZ5	1234	10*10	1245	1238	1238	1238
CAR5	7702	10*6	7835	7725	7725	7720
LA04	590	10*5	597	590	590	590
LA21	1046	15*10	1090	1059	1078	1047
MT10	930	10*10	944	940	951	930
ORB3	1005	10*10	1048	1023	1069	1024
ORB4	1005	10*10	1035	1013	1021	1013

Table 4: Average results for the Local Optimization Methods with D&T neighborhood structure.

Neighborhood: D&T		Size	SA	TS	LSSA	LSTS
Instance	Opt/LB	n*m	av.	av.	av.	av.
ABZ5	1234	10*10	1257	1240	1247	1239
CAR5	7702	10*6	7905	7790	8108	7940
LA04	590	10*5	605	591	598	591
LA21	1046	15*10	1100	1065	1096	1055
MT10	930	10*10	964	946	963	939
ORB3	1005	10*10	1073	1034	1077	1030
ORB4	1005	10*10	1056	1019	1027	1022

We have also compared several large-step optimization methods, which differ from each other by the number of the large-step iterations. The combination of large/small steps number of iterations are set in a special proportion to each other such that the complete run of the algorithms takes as much time as one run of the previous large-step optimization method with tabu-search, 12000 total iterations. In general the best results in terms of the best and the average value were obtained by large step optimization with the number of large step iterations between 5 and 20, with a little drop near 10 large-step iterations.

Viewing the results for both tabu-search with different neighborhoods, it seems that the simple neighborhood structure (VLA&L) with a large number of iterations can compete with a more complicated neighborhood structure (D&T), which need more time to pick the best move out of all possible ones, and therefore may be executed for fewer iterations. But more tests in this issue is need to be able to make stronger conclusions.

4. Diversification strategies

Long term memory is the basis for the diversifying strategy which has to lead the process to new regions of the solution space. The objective of a diversification strategy is to produce a new starting point, wherefore the local search can continue.

In this section we pay attention to a number of diversification strategies, which are closely related to the large-step phase. Both a diversification strategy as well as a large-step phase apply a big change to the current solution, which provides them the possibility to get the search process out of a local optima valley. The difference between the large-step procedure and the diversification strategies that we are proposing is that this latter ones make use of a long term memory, where, large-step optimization can apply a relevant big change to a given solution without the use of information directly from the past, but using optimization techniques.

In the approach to obtain a new starting point, two kind of diversifying strategies can be distinguished, strategies using a restart method and strategies using a method which lead the search to new regions by an iterative process. These last kinds of diversification methods usually provide advantages over restart methods, Glover (1993). For a number of iterations the random restart methods construct a random initial schedule, followed by the application of a local optimization procedure, returning the best solution found. The only difference with a more systematic restart method is that the initial solution is not randomly obtained, but constructed with information from previous periods in the search process. The storage of this information can be done with frequency counts. The idea is to count the number of times that moves are applied during the non-diversification phase of the search process, while during the diversification phase the moves are penalized according to their frequency counts, with the objective to oblige the process to search for moves not so often applied, resulting in never visited solutions.

Next, we present two diversification methods for the job-shop scheduling problem using frequency counts. The first method is a restart method which begins every diversification iteration by constructing a new starting schedule using frequency counts obtained in the preceding steps of the method, that we will call small steps in analogy with the large-step optimization methods. Every time the method finds a better solution in the small steps phase, best solution and best value are updated and the frequency memorizes the predecessor and successor of every job on every machine in the new

best schedule. After these period of small-steps is finished, the frequency counts for every job how many times a job was the predecessor and the successor of any another job at any machine in the improving schedules. We obtain a diversified schedule by constructing a schedule according to a priority rule, which can be overruled if it wants to schedule an operation after another operation resulting in an already existing neighbor combination in one of the memorized improving schedules. In this way, we try to obtain a schedule wherein the predecessor and successor of every job are different from the predecessors and successors of the same job in the same machine in the memorized improving schedules.

The second diversification we have implemented is an iterative method, using frequency counts, which leads on a path to new regions of the solution space. The frequency counts are obtained in the previous iterations by storing the number of times a certain arc has been reversed. In our implementation we memorized only the critical arcs, involved in a move. This is obvious for the VLA&L neighborhood. For the D&T neighborhood structure, counting the exact number of reversals would be very time consuming if we consider all single reverses. Therefore, we decided to count only the number of time the critical arcs are involved in a move. The diversification method consist of running the same tabu-search method, but now the moves are penalized to the critical arcs involved in the following way (see Laguna (1992)):

$\text{compare-length}(s') = \text{length}(s') + \text{penalty} * \text{frequency-of-critical-arc}$,
 where s' is the neighbor of s . If a critical arc of s is involved in the applied move, the number of times this arc was picked in the preceding steps is stored in the frequency-of-critical-arc. The penalty is the value which a move is punished per applied move involving the arc. In this way moves which were frequently applied in the preceding iterations are punished in the diversification phase to force the method to apply new moves, with the objective to drive the search process to new regions. The value of the variable penalty and the number of iterations in the diversification phase will be determined via trial and error method.

We obtained results with this method for four combinations of number of diversification steps/small steps iterations (see Tables 5, 6,

7 and 8). These combinations are similar to the ones used in testing the large-step optimization methods with tabu-search. With some exceptions the incorporation of our restart diversification strategy does not contribute to a better performance of the tabu-search method. We have observed that after the diversification iterations, a resulting schedule has a high length. In general, the best values, as well the average ones, are getting worse when the number of diversification steps increases. Therefore we can conclude that our restart diversification strategy changes the schedule too drastically. After the application of the diversification method, the tabu-search method needs a lot of iterations to get near to good solutions. For the method with 5 diversification step iterations, the good solutions are already difficult to reach. In general, the results are not better than the results of tabu-search without diversification.

Table 5: Best results obtained by the tabu-search method, large-step method with tabu-search and tabu-search with restart diversification strategy

Tabu-Search with restart diversification strategy						
best	Method		Number of diversification step iterations			
Instance	TS	LSTS	5	10	15	20
ABZ5	1238	1238	1238	1238	1238	1238
CAR5	7725	7720	7821	7843	7787	8039
LA04	590	590	590	590	590	590
LA21	1059	1047	1053	1068	1079	1082
MT10	940	930	940	940	940	940
ORB3	1023	1024	1020	1026	1035	1035
ORB4	1013	1013	1011	1011	1013	1017

Consider now the second diversification strategy, designated by iterative diversification strategy. When we compare the results obtained by this method (TSID) with tabu-search without diversification (TS) and large-step optimization with tabu-search (LSTS), we can conclude that on average this method can compete with LSTS and performs better than TS. As for example, for MT10 the best and average value obtained by TSID were 936 and 938 (penalty=10) or

930 and 941 (penalty=5), meanwhile for TS was 940 and 946, and for LSTS were 930 and 939. For LA21, the results were, for TSID and LSTS were 1047 and 1055, respectively.

Table 6: Average results obtained by the tabu-search method, large-step method with tabu-search and tabu-search with restart diversification strategy

Tabu-Search with restart diversification strategy						
average	Method		Number of diversification step iterations			
Instance	TS	LSTS	5	10	15	20
ABZ5	1240	1239	1243	1242	1242	1240
CAR5	7790	7940	7997	8067	8022	8180
LA04	591	591	591	590	590	590
LA21	1065	1055	1065	1073	1085	1088
MT10	946	939	952	950	954	956
ORB3	1034	1030	1040	1049	1055	1053
ORB4	1019	1022	1017	1015	1017	1024

Table 7: Best results obtained by the tabu-search method, large-step method with tabu-search and tabu-search with iterative diversification strategy

Tabu-Search with iterative diversification strategy					
best	Method		Penalty		
Instance	TS	LSTS	10	2	5
ABZ5	1238	1238	1238	1236	1236
CAR5	7725	7720	7731	7702	7787
LA04	590	590	590	590	590
LA21	1059	1047	1047	1059	1060
MT10	940	930	936	937	930
ORB3	1023	1024	1023	1023	1020
ORB4	1013	1013	1013	1013	1013

Table 8: Average results obtained by the tabu-search method, large-step method with tabu-search and tabu-search with iterative diversification strategy

Tabu-Search with iterative diversification strategy					
average	Method		Penalty		
Instance	TS	LSTS	10	2	5
ABZ5	1240	1239	1239	1240	1238
CAR5	7790	7940	7935	7923	8163
LA04	591	591	591	591	591
LA21	1065	1055	1055	1073	1066
MT10	946	939	938	939	941
ORB3	1034	1030	1027	1027	1025
ORB4	1019	1022	1017	1017	1017

The reason that the iterative diversification strategy performs better than the restart one is that, the iterative diversification with the chosen parameters changes the schedule enough to get out of an eventually bad valley, but not so radical that the resulting schedule has high length. The same happen when we apply the large-step procedure in the large-step optimization methods. In the iterative diversification strategy a kind of "punished optimization" is done, and with the right parameters, it does not take a lot of small-steps iterations to get near a good solution again, contrary to what happens in the restart diversification strategy.

5. Conclusions

The idea of combining large-step optimization method with a tabu-search approach seems interesting to solve combinatorial optimization problem, due the good results that we obtain from the application to the job-shop scheduling problem. Therefore, it will be interesting to develop similar methods to solve other problems and test the efficiency of these combined optimization methods. The combination of optimization and tailored methods with local search methods may lead to an improvement in the use of these last methods in solving problem. In our opinion, it is a very interesting area to develop.

Another issue that it will be relevant to explore is the connection between the large-step procedure in the large-step optimization methods and the diversification strategies, briefly mentioned in this work. The relation with the intensification strategies was not considered in the work, but we are planning to do it in the near future.

6. References:

- P. Brucker, J. Hurink and F. Werner, Improving Neighborhoods for Local Search Heuristics, Working paper, Osnabrücker Schriften zur Mathematik, Universität Osnabrück (1993).
- P. Brucker, J. Hurink and F. Werner, Improving Local Search Heuristics for Some Scheduling Problems, Working Paper, Osnabrücker Schriften zur Mathematik, Universität Osnabrück (1994).
- J. Carlier, The One-Machine Sequencing Problem, *European Journal of Operational Research*, 11 (1982) p. 42.
- M. Dell'Amico and M. Trubian, Applying Tabu-Search to the Job-Shop Scheduling Problem, *Annals of Operations Research*, 41 (1993) p. 231.
- T.A. Feo and M.G.C. Resende, Greedy Adaptive Search Procedures, to appear in *Annals of Operations Research*.
- F. Glover, Tabu Search - Part I, *ORSA Journal on Computing*, 1(3) (1989) p. 190.
- F. Glover, "Tabu Thresholding: Improved Search in Nonmonotonic Trajectories", to appear in *ORSA Journal on Computing*.
- F. Glover, M. Laguna, T. Taillard and D. de Werra, Tabu Search, *Annals of Operations Research*, 41 (1993).

- D.S. Johnson, Local Optimization and the Traveling Salesman Problem, in: *Proceedings of the 17th Annual Colloquim on Automata, Languages and Programming* (Springer-Verlag, 1990) p.446.
- D.S. Johnson, Random starts for local optimization, DIMACS Workshop on Randomized Algorithms for Combinatorial Optimization (1993).
- D.S. Johnson, C.R. Aragon, L.A. McGeoch, and C. Schevon, Optimization by Simulated Annealing: an experimental evaluation; part I, graph partitioning, *Operations Research*, 39(3) (1989) p.865.
- M. Laguna, A guide to implementing Tabu Search, *Investigacion Operativa*, (1994).
- H.R. Lourenço, Job-Shop Scheduling: Computational Study of Local Search and Large-Step Optimization Methods, *European Journal of Operational Research*, 83 (1995) p.347.
- V.B.F. Lucks, Large-Step Local Improvement Optimization for the Graph Partitioning Problem, Master's Dissertation, Cornell University, Ithaca, NY, USA (1992).
- O. Martin, S.W. Otto and E.W. Felten, Large-Step Markov Chains for the TSP Incorporating Local Search Heuristics, *Operations Research Letters*, 11 (1992) p. 219.
- P.J.M. van Laarhoven, E.H.L. Aarts and J.K. Lenstra, Job Shop Scheduling by Simulated Annealing, *Operations Research*, 40(1) (1992) p. 113.

Job-Shop Scheduling by Simulated Annealing Combined with Deterministic Local Search

Takeshi Yamada and Ryohei Nakano

NTT Communication Science Laboratories

2 Hikaridai Seika-cho Soraku-gun Kyoto 619-02 JAPAN

E-mail: {yamada,nakano}@cslab.kecl.ntt.jp

Abstract:

The Job-Shop Scheduling Problem (JSSP) is one of the most difficult \mathcal{NP} -hard combinatorial optimization problems. This paper proposes a new method for solving JSSPs based on simulated annealing (SA), a stochastic local search, enhanced by shifting bottleneck (SB), a problem specific deterministic local search. In our method new schedules are generated by a variant of Giffler and Thompson's active scheduler with operation permutations on the critical path. SA selects a new schedule and probabilistically accepts or rejects it. The modified SB is applied to repair the rejected schedule; the new schedule is accepted if an improvement is made. Experimental results showed the proposed method found near optimal schedules for the difficult benchmark problems and outperformed other existing local search algorithms.

Key Words: Simulated annealing, shifting bottleneck, job-shop scheduling, heuristics, local search

1. Background

Scheduling is allocating shared resources over time to competing activities. It has been the subject of a significant amount of literature in the operations research area. Emphasis has been on investigating *machine scheduling problems* where *jobs* represent activities and *machines* represent resources; each machine can process at most one job at a time.

The $n \times m$ *minimum-makespan* general job-shop scheduling problem, hereafter referred to as JSSP, can be described by a set of n jobs that is to be processed on a set of m machines. Each job has a technological se-

quence of machines to be processed. Each *operation* requires the exclusive use of each machine for an uninterrupted duration called *processing time*. The time required to complete all jobs is called *makespan*. The objective when solving or optimizing this general problem is to determine the processing order of all operations on each machine that minimizes the makespan. The JSSP is not only \mathcal{NP} -hard, but is extremely difficult to solve optimally. An indication of this is given by the fact that one 10×10 problem formulated by Muth and Thompson (1963) remained unsolved for over 20 years.

Besides exhaustive search algorithms based on branch and bound methods, several approximation algorithms have been developed. The most popular ones in practice are based on priority rules and active schedule generation. Adams, Balas and Zawack (1988) developed a more sophisticated method called *shifting bottleneck* (SB) which has been shown to be very successful. Stochastic approaches such as simulated annealing (SA), genetic algorithms and tabu search have been recently applied with good success.

This paper proposes a powerful method to solve the JSSP. The method is based on a combination of critical block simulated annealing (CBSA): a stochastic local search method proposed by Yamada, Rosen and Nakano (1994), and shifting bottleneck: a deterministic local search method. CBSA possessed the simplicity and flexibility of SA, worked well for difficult benchmark problems, and outperformed the existing SA approaches. However more recent approaches perform better than CBSA. In the present method, a modified SB is applied to repair the rejected schedule of CBSA and the new schedule is accepted if it is improved. By adding a long jump of SB, the new CBSA can omit many time-consuming transitions to make its search much more efficient.

2. Neighborhood Structure

2.1. Critical blocks

A JSSP is often described by a disjunctive graph $G = (V, C \cup D)$, where

- V is a set of nodes representing operations of the jobs together with two special nodes, a *source* (0) and a *sink* \star , representing the beginning and end of the schedule, respectively.
- C is a set of conjunctive arcs representing technological sequences of the operations.
- D is a set of disjunctive arcs representing pairs of operations that must be performed on the same machines.

The processing time for each operation is the weighted value attached to the corresponding nodes.

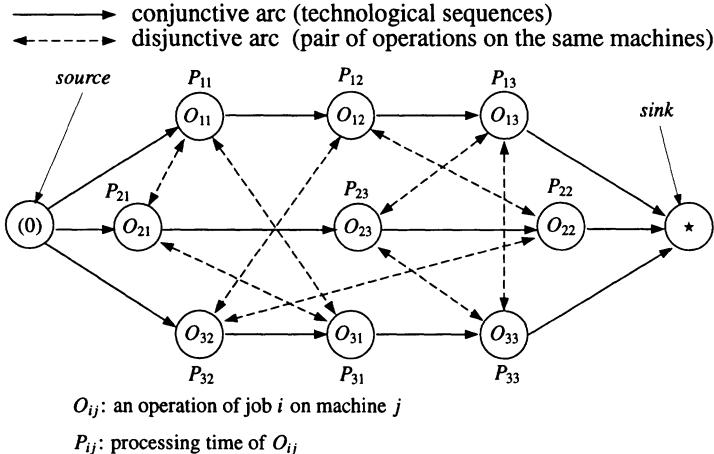


Figure 1: The disjunctive graph G of a 3×3 problem

Scheduling is to define ordering between all operations that must be processed on the same machine, i.e. to fix precedences between these operations. In the disjunctive graph model, this is done by turning all undirected (disjunctive) arcs into directed ones. The set of all directed arcs selected from disjunctive arcs is called *selection*. A selection S defines a feasible schedule if and only if the resulting directed graph is acyclic. For such a case, S is called a *complete selection*. A complete selection and the corresponding feasible schedule can be used interchangeably and represented by the same symbol S . *Makespan* is given by the length of the longest weighted path from source to sink in this graph. This path \mathcal{P} is called *critical path* and is composed of a sequence of *critical operations*. A sequence of consecutive critical operations on the same machine is called a *critical block*.

2.2. Critical block neighborhood

For a JSSP, a neighborhood $N(S)$ of a schedule S can be defined as the set of feasible schedules that can be reached from S by exactly one transition (a single perturbation of S). For example, a transition operator that exchanges a pair of consecutive operations in a critical block and forms a neighborhood has been used in Laarhoven, Aarts and Lenstra (1992) and in Taillard (1994). The transition operator was originally defined by

Balas (1969) in his branch and bound approach. Another very powerful transition operator was used in Brucker, Jurisch and Sievers (1994) and in Dell'Amico and Trubian (1993). The transition operator permutes the order of operations in a critical block by moving an operation to the beginning or end of the critical block, thus forming a *CB neighborhood*.

A schedule obtained from S by moving an operation within a block to the front of the block is called a *before candidate*, and a schedule moving an operation to the rear of the block is called an *after candidate*. A set of all before and after candidates $N'^C(S)$ may contain infeasible schedules. The CB neighborhood is given as:

$$N^C(S) = \{S' \in N'^C(S) \mid S' \text{ is a feasible schedule}\}.$$

It has been experimentally shown by Yamada, Rosen and Nakano (1994) that the CB neighborhood is more powerful than the former one.

A schedule's makespan may often be reduced by shifting an operation to left without delaying other jobs. When no such shifting can be applied to a schedule, it is called an *active schedule*. An optimal schedule is clearly active so it is safe and efficient to limit search space to the set of all active schedules. An active schedule is generated by the *GT algorithm* proposed by Giffler and Thompson (1960). The outline is described in Algorithm 2.1. In the algorithm, the *earliest completion time* $EC(O)$ of an operation O means its completion time when processed with highest priority. An active schedule is obtained by repeating the algorithm until all operations are processed. In step 3, if all possible choices are considered, all active schedules will be generated, but the total number will still be very large.

Algorithm 2.1 GT algorithm

1. Let O^* be an operation with the minimum among the earliest completion times of unscheduled operations and M^* be the machine which processes O^* : $EC(O^*) = \min\{EC(O) \mid O : \text{unscheduled}\}$.
 2. Assume $j - 1$ operations have been processed on M^* . A *conflict set* $C[M^*, j]$ means a set of unscheduled operations on M^* each of whose processing overlaps with O^* .
 3. Select an operation O from $C[M^*, j]$ and schedule it as the j -th operation on M^* with its completion time equals to $EC(O)$.
-

As explained above, a before or after candidate is not necessarily executable. In the following, we propose a new neighborhood similar to

CB neighborhood; its element is not only executable, but also active and close to the original. Let S be an active schedule and $B_{k,h,M}$ be a block of S on a machine M , where the front and the rear operations of $B_{k,h,M}$ are the k -th and the h -th operations on M respectively. Let $O_{p,M}$ be an operation in $B_{k,h,M}$ that is the p -th operation on M . Algorithm 2.2 generates an active schedule $S_{M,p,k}$ (or $S_{M,p,h}$) by modifying S such that $O_{p,M}$ at the position p is moved to the position as close to the front position k (or the rear position h) of $B_{k,h,M}$ as possible. Parts of the algorithm are due to Dell'Amico and Trubian (1993). The new neighborhood $AN^C(S)$ is now defined as a set of all $S_{M,p,k}$'s and $S_{M,p,h}$'s over all critical blocks:

$$AN^C(S) = \bigcup_{B_{k,h,M}} \{S' \in \{S_{M,p,k}\}_{k < p < h} \cup \{S_{M,p,h}\}_{k < p < h}, S' \neq S\}.$$

Algorithm 2.2 Modified GT algorithm generating $S_{M,p,k}$ or $S_{M,p,h}$

1. Same as step 1 of Algorithm 2.1.
 2. Same as step 2 of Algorithm 2.1.
 3. **CASE 1:** $S_{M,p,k}$ generation
 - If $k \leq j \leq p$ and $O_{p,M} \in C[M^*, j]$, then select and schedule $O_{p,M}$.
 - Otherwise, select and schedule from $C[M^*, j]$ the earliest operation in S .
 - CASE 2:** $S_{M,p,h}$ generation
 - If $p \leq j \leq h$ and $C[M^*, j]$ contains an operation other than $O_{p,M}$, then select and schedule from $C[M^*, j] \setminus \{O_{p,M}\}$ the earliest operation in S .
 - If $j = h$ or $C[M^*, j] = \{O_{p,M}\}$, then select and schedule $O_{p,M}$.
 - Otherwise, select and schedule from $C[M^*, j]$ the earliest operation in S .
-

3. Critical Block Simulated Annealing

Critical block simulated annealing (CBSA) proposed by Yamada, Rosen and Nakano (1994) is a method of searching job shop scheduling space by using simulated annealing with $N^C(S)$ as the neighborhood of a schedule S . CBSA consists of three parts: the main part is the iteration loop shown in Algorithm 3.1; the *warmup* is a reverse annealing process which adaptively determines initial and final temperatures after starting at a sufficiently low temperature; the *reintensification strategy* makes the system jump from the current state to the best one obtained so far if no better solution is found after a large number R of acceptances (R is called

the reintensification frequency). In this paper, CBSA is extended to use $AN^C(S)$ instead of $N^C(S)$.

Algorithm 3.1 CBSA main loop

1. Set $S = S_0$, a randomly generated initial schedule with makespan $L(S_0)$, the initial step counter $k = 0$, and $T_{k=0} = T_0$, the initial temperature.
 2. Repeat the following steps:
 - (a) Pick S' from $AN^C(S)$ randomly.
 - (b) Accept S' probabilistically according to the Metropolis Criterion, i.e. choose S' with probability one if $L(S') \leq L(S)$, and with $e^{-(L(S') - L(S))/T_k}$ otherwise.
 - (c) If S' is accepted, set $S = S'$.
 3. Set $k = k + 1$ and decrease the current temperature T_k according to the annealing schedule.
 4. If $T_k > T_f$, the final temperature, go to step 2; otherwise stop.
-

In Algorithm 3.1, if the acceptance probabilities are low for all members in $AN^C(S)$, the system will remain trapped in a local minimum S and it will take a long time to move to a new state. The algorithm may stay in S even after all members are selected in step 2a and evaluated in step 2b. To avoid this, relative acceptance probability $\bar{P}(S_i) = P(S_i)/\sum_{S_j \in AN^C(S)} P(S_j)$ for each $S_i \in AN^C(S)$ will be introduced after all members in $AN^C(S)$ are visited. The member S_i in $AN^C(S)$ is then randomly selected in proportion to $\bar{P}(S_i)$, and the system moves unconditionally to S_i . This modification is effective because the neighborhood size is limited. Therefore, Algorithm 3.1 is modified as in Algorithm 3.2.

4. CBSA enhanced by Shifting Bottleneck

Shifting bottleneck (SB) proposed by Adams, Balas and Zawack (1988) is a powerful method for solving a JSSP. In the method, a one machine scheduling problem (a relaxation of the original JSSP) is solved for each machine not yet sequenced, and the outcome is used to find a bottleneck machine, a machine of the longest makespan. Every time a new machine has been sequenced, the sequence of each previously sequenced machine is subject to reoptimization. SB consists of two subroutines: the first one (SBI) repeatedly solves one machine scheduling problems; the second one (SBII) builds a partial enumeration tree where each path from the root to a leaf is similar to an application of SBI.

SBI is a constructive method that generates a complete schedule from

Algorithm 3.2 Modified version of CBSA main loop

1. Set $S = S_0$, $k = 0$, and $T_{k=0} = T_0$. Set the number of $AN^C(S)$ members already visited $n = 0$.
 2. Repeat the following steps:
 - (a) Select S_i from $AN^C(S)$ randomly. Calculate $L(S_i)$ and set $n = n + 1$ if S_i is not yet visited.
 - (b) Accept S_i with probability one if $L(S_i) \leq L(S)$, and with $P(S_i) = e^{-(L(S_i) - L(S))/T_k}$ otherwise.
 - (c) If S_i is accepted, then set $S = S_i$, $n = 0$.
Or if $n = N$, select S' from $AN^C(S)$ in proportion to the probability
 $\bar{P}(S') = P(S') / \sum_{S_j \in AN^C(S)} P(S_j)$, and set $S = S'$, $n = 0$.
 3. Set $k = k + 1$ and decrease T_k according to the annealing schedule.
 4. If $T_k > T_f$, go to step 2; otherwise stop.
-

scratch. Modifying the method is necessary in order to refine a certain complete schedule for improvement. The *BottleRepair* shown in Algorithm 4.2 describes an iterative version of the basic SB. The reoptimization process used here is the same as used in Algorithm 4.1. The basic idea of *BottleRepair* comes from the original paper of SB (1988) where the last α noncritical machines are temporarily removed for the reoptimization.

Algorithm 4.1 SBI

1. Set $S = \emptyset$ and make all machines unsequenced.
 2. Solve a one machine scheduling problem for each unsequenced machine. Among the unsequenced machines, find the bottleneck machine and add its schedule to S . Make the machine sequenced.
 3. Reoptimize all sequenced machines in S .
 4. Go to step 2 unless S is completed; otherwise stop.
-

As shown in Algorithm 3.2, S_i is selected from $AN^C(S)$ and is probabilistically accepted. *BottleRepair* is applied to S_i only when S_i is rejected. The resulting schedule S_i^* is accepted if its makespan is shorter than that of S . To summarize, step 2b-1 as defined in Algorithm 4.3 is added to Algorithm 3.2 just after step 2b.

BottleRepair gives a systematic way to inspect the schedule's critical path and permutes operations again and again by repeatedly solving one machine problems in a deterministic manner. If it generates an improved schedule S' from S , the critical path of S' becomes different from

Algorithm 4.2 *BottleRepair*: Iterative SB

1. A certain complete schedule S is given. Reset all sequences of all the non-critical machines (machines which do not include any part of critical path in S) and make the machines unsequenced.
 2. Reoptimize all sequenced machines.
 3. Solve a one machine scheduling problem for each unsequenced machine. The machines are ranked by the makespans. The next two steps are applied in the descending order of rank.
 4. Solve a one machine scheduling problem and add its schedule to S . Make the machine sequenced.
 5. Reoptimize all the sequenced machines.
-

S and the difference is much greater than that between S and its CBSA neighbor. On the other hand, CBSA gives a stochastic more focused local search around the current critical path. The proposed integration of CBSA and SB is expected to have the synergistic effect as: SB gives a long jump to CBSA so that it can omit many time-consuming inferior transitions and CBSA adds stochastic perturbations to SB so that it can escape from the local minima.

Algorithm 4.3 SB enhancement for CBSA

- 2b-1** If S_i is rejected, apply *BottleRepair* to S_i and generate S_i^* . Accept S_i^* and set $S_i = S_i^*$ if $L(S_i^*) < L(S)$.
-

5. Experimental Results

5.1. Muth and Thompson's Benchmark

A 10×10 problem (mt10) and 20×5 problem (mt20) formulated by Muth and Thompson (1963) (*MT benchmarks*) are well known benchmark JSSPs. CBSA with and without SB modification was evaluated using these problems. Table 1 shows the results of 20 trials with different random number seeds on a SUN SPARC station 10. All programs are written in the C language.

Results for the mt10 problem using CBSA without SB show that the optimal solutions of $L = 930$ were found in 11 trials. The average cpu time was 3 min. 10 sec., and the fastest was 1 min. 21 sec. Although the solutions of $L = 930$ were found in only half of the trials, the cpu time in successful runs were satisfactorily short. If the temperature is more slowly lowered, though it takes longer, the rate of finding optimal

Table 1: Comparisons between CBSA and CBSA+SB using MT benchmarks

Prob	$n \times m$	CBSA ($R = 6,000$)				CBSA+SB			
		best	mean	std	BT	best	mean	std	BT
mt10	10×10	930	933.65	4.04	190	930	932.45	3.01	786
mt20	20×5	1178	1179.45	1.94	235	1165	1165.00	0.00	449

std: standard deviation

BT: average cpu time (sec.) to find the best solution

solutions will become higher as in Yamada, Rosen and Nakano (1994). CBSA without SB could not find any optimal solution for mt20 problem. In most cases, solutions of $L = 1178$ were found instead of the optimal $L = 1165$.

The results for the mt10 problem using CBSA with SB modification show that the number of trials finding optimum solutions increased slightly, but the average cpu time increased about four times. This fact indicates that CBSA without SB is powerful enough to solve mt10 problem. On the other hand, all 20 trials with SB modification for mt20 problem found the optimal solutions of $L = 1165$ in an average cpu time of 7 min. 29 sec., and 1 min. 22 sec. was the best time. The effect of SB enhancement is obvious from this problem. Reintensification did not work well because the optimal or near optimal solutions were obtained at an early stage of the search.

5.2. Other Benchmarks

Results in the previous section indicate that if CBSA without SB can solve a problem skillfully, applying CBSA+SB has no advantage. However, if CBSA fails to work well, CBSA+SB may improve solution quality and compensate for the extra cpu time needed with SB enhancement. A set of benchmark problems has been established to evaluate different algorithms for JSSPs. Table 2 shows the makespan performances of CBSA+SB and various other algorithms for the ten difficult benchmark JSSPs. All experiments of CBSA+SB runs were done on a HP 730 (HP 730 is about 1.5 times faster than SUN SPARC station 10).

The LB column indicates the theoretical lower bound of the problem if the optimal makespan is unknown. The CBSA+SB column indicates the best makespans found from ten trials. Each trial used a reintensification frequency of $R = 1,000$ and ran for three hours or until the known optimal makespan or lower bound was found. The column head-

ings Aart, Matt, Appl and Tail indicate the best performances of those in Aarts *et al.* (1994), Mattfeld, Kopfer and Bierwirth (1994), Applegate and Cook (1991) and Taillard (1994) respectively.

Table 2: Results of 10 tough JSSPs

Prob	$n \times m$	LB	CBSA+SB				Aart	Matt	Appl	Tail
			best	mean	std.	BT				
abz7	20×15	654	665	671.0	3.92	7814	668	672	668	665
abz8	20×15	635	675	680.0	3.13	8775	670	683	687	676
abz9	20×15	656	686	698.6	7.42	8749	691	703	707	691
la21	15×10	1040	1046	1049.3	3.32	361	1053	1053	1053	1047
la24	15×10	–	935	939.2	1.99	6226	935	938	935	935
la25	20×10	–	977	979.3	1.62	4117	983	977	977	977
la27	20×10	1235	1235	1242.4	6.15	7805	1249	1236	1269	1240
la29	20×10	1120	1154	1162.4	7.10	5434	1185	1184	1195	1170
la38	15×15	1184	1198	1206.8	4.53	3479	1208	1201	1209	1202
la40	15×15	–	1228	1230.2	2.32	3331	1225	1228	1222	1222

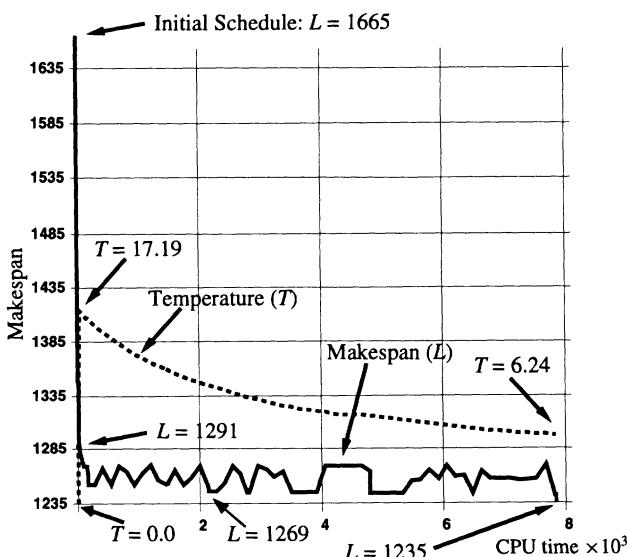


Figure 2: The time evolution of CBSA+SB trial for the la27 problem

Figure 2 shows the time evolution of the makespan (L) and temperature (T) of the best trial of CBSA+SB for the la27 problem. The abscissa

Table 3: An optimal solution of la27 problem

<i>m</i>	Job sequences on each machine																			
1	18	9	4	2	11	3	12	8	10	7	13	15	20	1	6	19	5	14	17	16
2	14	9	8	12	3	17	15	4	10	13	6	7	11	19	1	2	18	16	20	5
3	11	14	3	19	10	4	9	5	6	2	16	17	13	15	7	20	18	12	8	1
4	6	11	10	1	19	3	8	13	15	18	2	12	14	17	4	7	9	16	20	5
5	12	3	5	6	20	13	11	8	1	17	9	19	7	14	2	16	18	15	10	4
6	19	18	11	8	14	7	3	16	2	17	1	10	9	5	13	6	20	4	12	15
7	17	2	13	15	9	8	19	7	6	20	10	18	14	16	5	4	3	1	11	12
8	2	12	16	14	11	10	3	5	19	20	6	7	13	17	8	18	15	1	4	9
9	18	12	3	14	11	10	16	8	19	13	4	6	15	2	17	20	1	9	7	5
10	13	20	18	9	4	5	11	6	3	8	2	19	10	14	1	17	7	15	16	12

m: machine

shows the cpu time in seconds, and the solid and dotted lines show the makespan and the temperature respectively. Starting from the makespan value $L = 1665$, it rapidly decreases to $L = 1291$ during the first warmup interval. After twelve times of reintensification, it finally reached the optimal value $L = 1235$. In this experiment, 56059 schedules were generated and 8850 schedules accepted. About 25% of the accepted schedules were accepted by *BottleRepair* to add CBSA long jumps and the rest served as stochastic perturbations to *BottleRepair*. The oscillatory behavior is due to reintensification.

Although CBSA+SB outperformed other methods most of the cases in Table 2, the required computational time is much longer. For example, Vaessens, Aarts and Lenstra (1994) reported that Applegate's method in the Appl column found a schedule of $L = 1269$ in 604.2 sec. on SUN SPARC station ELC which is about 10 times slower than HP 730. This is because each CBSA+SB experiment includes a lot of unsuccessful *BottleRepair* trials. But this gap can be filled to some extent by the fact that in the same experiment, CBSA+SB passed a point $L = 1269$ in 164 sec.

6. Conclusion

The proposed method CBSA+SB is an improved CBSA enhanced by integrating with a problem specific method called shifting bottleneck. The performance of CBSA+SB was evaluated using difficult benchmark problems. The results show that for eight problems of ten difficult benchmark problems, CBSA+SB could find schedules better than or equal to

the best schedules published so far in the literature, when enough computational time is given. A new solution of $L = 1235$ was found for the la27 problem; it is optimal because the value equals the theoretical lower bound. Further research is necessary to reduce the computational time.

7. References

- E.H.L. Aarts, P.J.M. van Laarhoven, J.K. Lenstra and N.L.J. Ulder,
 A computational study of local search algorithms for job shop
 scheduling, *ORSA J. on Comput.*, 6 (1994) 118.
- J. Adams, E. Balas and D. Zawack, The Shifting Bottleneck Procedure
 for Job Shop Scheduling, *Mgmt. Sci.*, 34 (1988) 391.
- D. Applegate and W. Cook, A Computational Study of the Job-Shop
 Scheduling Problem, *ORSA J. on Comput.*, 3 (1991) 149.
- E. Balas, Machine sequencing via disjunctive graphs: an implicit enu-
 meration algorithm, *Oper. Res.*, 17 (1969) 941.
- P. Brucker, B. Jurisch and B. Sievers, A Branch & Bound Algorithm for
 the Job-Shop Scheduling Problem, *Discrete Applied Mathematics*,
 49 (1994) 107.
- D.C. Mattfeld, H. Kopfer, and C. Bierwirth, Control of Parallel popula-
 tion dynamics by social-like behavior of ga-individuals, In: *Proc.
 of PPSN 3rd* (Springer-Verlag, 1994) p. 16.
- M. Dell'Amico and M. Trubian, Applying tabu search to the job-shop
 scheduling problem, *Annals of Oper. Res.*, 41 (1993) 231.
- B. Giffler and G.L. Thompson, Algorithms for solving production
 scheduling problems. *Oper. Res.*, 8 (1960) 487.
- P.J.M. van Laarhoven, E.H.L. Aarts and J.K. Lenstra, Job Shop Schedul-
 ing by Simulated Annealing, *Oper. Res.*, 40 (1992) 113.
- J.F. Muth and G.L. Thompson, *Industrial Scheduling*, (Prentice-Hall,
 New Jersey, 1963).
- E.D. Taillard, Parallel taboo search techniques for the job-shop schedul-
 ing problem, *ORSA J. on Comput.*, 6 (1994) 108.
- R.J.M. Vaessens, E.H.L. Aarts, and J.K. Lenstra, Job shop scheduling
 by local search, Tech. rept., COSOR 94-05, Eindhoven University
 of Technology, Dpt. of Math. and CS, NL (1994).
- T. Yamada, B.E. Rosen and R. Nakano, A Simulated Annealing Ap-
 proach to Job Shop Scheduling using Critical Block Transition Op-
 erators, In: *Proc. of IEEE ICNN* (IEEE, Florida, 1994).

**Cybernetic Optimization by Simulated Annealing:
An Implementation of Parallel Processing
Using Probabilistic Feedback Control**

Mark A. Fleischer

Department of Engineering Management

Old Dominion University

Norfolk, Virginia 23529-0248

E-mail: maf100f@oduvm.cc.odu.edu

Sheldon H. Jacobson

Department of Industrial and Systems Engineering

Virginia Polytechnic Institute and State University

Blacksburg, Virginia 24061-0118

E-mail: jacobson@vtvm1.cc.vt.edu

Abstract:

The convergence of the simulated annealing (SA) algorithm is accelerated by a *probabilistic feedback control* (*PFC*) scheme. This scheme uses two or more parallel processors to solve the same or related combinatorial optimization problems and are coupled by a *Probabilistic Measure of Quality* (*PMQ*). The *PMQ* is used to generate an error signal for use in feedback control. Control over the search process is achieved by using the error signal to modulate the temperature parameter. Other aspects of control theory, such as the system gain and its effects on system performance, are described. Theoretical and experimental results which demonstrate the merits of a *PFC* system are also presented.

Key Words: Control theory, cybernetics, parallel processing, simulated annealing.

1. Introduction

This paper shows how certain properties of the simulated annealing (SA) algorithm and concepts from cybernetics (Wiener 1961) can be merged to provide a new paradigm for probabilistic optimization techniques. This approach, cybernetic optimization by simulated annealing (COSA), provides intelligence to the SA algorithm that can be used to improve its performance by incorporating a negative feedback control scheme in the algorithm using parallel processing.

SA has been a very popular algorithm for finding global optima for many different types of problems (Eglese 1990). Its attractiveness stems from the ease with which it can be implemented, its convergence properties, and the fact that it can escape local optima by virtue of its hill-climbing capability. In some ways, however, it is a "dumb" optimization scheme. In its generic or traditional implementation, it is blind to the global optimum; it can find it and then leave it, searching the entire configuration space in total ignorance of the quality of the solution it left.

Enhancements to the generic implementation are, however, often obvious and easy enough to implement. The most popular method is to keep track of the best solution obtained up to the current iteration. On the other hand, in *iterated descent* one can perform repeated local searches with random starting solutions often with similar results (Reeves 1993). It seems therefore that the convergence properties of SA are attractive only from a scientific and mathematical point of view, and that its practical use as an optimization scheme is often overstated. Indeed, this may explain why SA has so many variations, promoters, and detractors (Eglese 1990). We believe, however, that the convergence properties of SA have hitherto not been fully utilized and that one way to do so is by employing concepts from cybernetics.

Cybernetics, coined by Wiener (1961), comes from the Greek word for *steersman* and unites concepts of information, communication, and control theory. Its central message is that control over some process can be achieved by the generation and communication of information often in the form of some *feedback* mechanism. With appropriate processing, this information can be used to bring a process more on target and under control. Can such concepts be used in solving optimization problems?

Clearly, *information*, how it is generated, measured and transmitted is a cornerstone of cybernetics. But it also plays an important role in optimization. Afterall, optimization algorithms are simply methods of taking raw data (*information*), processing it, and ultimately reporting it. Optimizing such optimization techniques continues to be an active area of research the goal of which is to overcome the limitations posed by current technologies and methods. Current attempts to overcome these limitations involve the use of parallel processing (Aarts and Korst 1989, Azencott 1992).

1.1 Parallelization Issues

Parallelization of algorithms often requires that problems be broken up into numerous subproblems that can be solved simultaneously. For some types of problems, there are natural ways of breaking up a problem into *independent* subproblems. For others, however, it is difficult to conceive of ways of doing this. Often, a subproblem's solution depends on another subproblem's solution and must be solved *before* the other. These problems, therefore, cannot be solved in parallel or simultaneously.

In the realm of combinatorial optimization problems (*COPs*), natural ways of breaking up problems often seem lacking. This is because such problems have as their solution some *combination* of attributes. One cannot simply divide up these attributes into separate and distinct subsets, solve the problem for each subset, and then connect all the solutions. This is because the objective function value depends on *all* of these attributes.

A number of search algorithms and heuristics for solving such *COPs* exist, but usually take the solution space as a whole (Manber 1989). How can such algorithms be run in parallel? Intuitively, if two processors executing a search algorithm are applied to a given problem, they ought to find the global optimum sooner than only one processor. Roughly speaking, each processor would have to search only half the solution space in a given amount of time. Adding more processors therefore has the potential of speeding things up. This is the basis for most so-called *SPMD* designs (Azencott 1992, Reeves 1993). To make such an approach effective, however, requires that each processor be given some *information* on the progress of the other. Otherwise, all we can expect is two processors working independently and doubling our electric bill (Azencott 1992)!

1.2 Communication Issues

Azencott (1992) describes several effective parallelization techniques, such as *parallel annealing by periodically interacting multiple searches*. After several independent processors solving the same problem are run for a fixed number of iterations, the best solution among them is *communicated* to all processors. Annealing then proceeds for each processor from that point on at a slightly lower temperature. This raises yet another fundamental issue in parallel computing: how best to utilize the information generated by each processor. This brings us back to the fundamental issues involved in cybernetics—information, communication, and control.

This paper addresses these issues directly by showing at once how the SA algorithm can be parallelized *and* how information can effectively be communicated among parallel processors through the use of a *feedback loop*. Such feedback systems however generally require the generation of an error signal that measures how far off target a system is. In *negative feedback*, this signal is *fed back* to some actuation or control device to modify its output in such a way that the error signal is *reduced*.

Although a number of researchers have described so-called *adaptive* methods, some of which incorporate parallelization techniques, these methods are not based on the negative feedback of error information. Instead, they modify the gross structure of the SA algorithm (such as the constants in the cooling schedule or the type of cooling schedule) based on aspects of a particular problem (Ingber 1995, Frost 1993). None of the documented parallel computing schemes employ the concepts of *negative feedback control*. Nor do these methods truly take advantage of one of SA's most attractive elements—convergence in probability to the global optimum.

Such feedback must be based on information produced by the system. In this context, the information comes from another processor running SA on the same *COP* as part of a *probabilistic feedback control (PFC)* scheme. The next few sections describe the elements of PFC, its application, and its performance.

2. Probabilistic Feedback Control

There are two fundamental aspects to a feedback control system: the generation of an error signal, and the use of that signal for control purposes. In the context of parallel SA, such an error signal must necessarily be based on *probabilistic* information regarding the quality of the current solutions. This quality measure will therefore be referred to as a **Probabilistic Measure of Quality (PMQ)**. Thus, all we can hope to achieve in terms of control is to affect the degree of randomness that the search mechanism entails. The SA algorithm has a perfect mechanism for doing this: the temperature *control* parameter. Changing the value of this parameter changes the transition probabilities and, hence, the entropy generated by the associated Markov chain. Thus, rather than altering some direction, rate, or acceleration by feedback, as in many feedback control systems, we can alter the *probabilities and associated uncertainty levels* by feedback.

The utility of *COSA* and the *PMQ* is founded on SA's convergence property: if two or more processors generate solutions that are close to the optimum an increasing proportion of time (*i.e.*, convergence in probability), then the two solutions should be *close to each other* an increasing proportion of time. Conversely, and *assuming convergence in probability*, if two processors generate solutions that are in some sense close to each other, then it is *more likely* that one or both processors is close to the optimum. Thus, the proximity of two solutions forms the basis for the generation of an error signal (see Glover 1995 for a description of his *proximate optimality principle*). Control over the search process can be effected by either raising or lowering the temperature depending upon the magnitude of the *PMQ*. This affects the information rate of the search process all the while allowing two (or more) processors to work on a given problem concurrently.

To summarize, the idea behind *PFC* is this: when it is more likely that at least one processor is producing a relatively good solution, then the algorithm should lower the information rate and uncertainty (Fleischer 1993) associated with the next state. This makes it more likely that the algorithm will explore nearby solutions. Conversely, if the solutions are likely to be poor, then increasing the information rate and uncertainty levels allows SA to explore more remote regions of the solution space. Put another way, if one is searching for gold (an inherently probabilistic endeavor), one is more likely to succeed if one spends more time looking

where all the other prospectors are looking.

2.1 Mathematical Foundations: Error, Control, Gain

The following definitions pertain to the underlying graph $G(V, E)$ (Mitra, *et al.* 1986) which models the configuration space of a COP solved by SA.

Definitions:

i_x the index of a solution associated with processor Π_x in a PFC system.

$d_{i_x i_y}$ the length of the shortest path in $G(V, E)$ between solutions i_x and i_y .

$X^{[k]}$ the random minimum distance from the globally optimal solution obtained by processor Π_x at time k . $Y^{[k]}$ is defined similarly.

$Z^{[k]}$ the random minimum distance between the current solutions of processors Π_x and Π_y at time k .

$\alpha_x^{[k]}, \alpha_y^{[k]}, \beta_y^{[k]}$ the sets of attributes corresponding to the current solutions at time k for processor Π_x and Π_y , respectively where $\beta_y^{[k]}$ corresponds to the sets of attributes of a complementary problem being solved by processor Π_y .

$\pi_{i_x i_y}(t)$ the stationary probability of solution i_x, i_y of the system at temperature t where the state space is expanded to encompass the solutions of both processors Π_x and Π_y .

$G_{i_x j_x}$ the probability of generating candidate solution j_x given the current solution i_x .

f_{i_x} the objective function value of solution i_x .

$$\Delta f_{j_x i_x}^+ \equiv \max\{0, f_{j_x} - f_{i_x}\}.$$

PMQs can be based on any value that converges in probability to zero. For instance, a PMQ can be based on the differences in the objective function values between two parallel processors with similar theoretical and experimental results (Fleischer 1995). This paper explores the use of PMQs based on the attributes of a COP as depicted in Figure 1. As each processor proceeds, the sets of attributes corresponding to the current solutions of each processor will tend to overlap to a greater degree and more frequently reflecting the convergence in probability of SA. This is depicted in the progression from picture (a) to (b) to (c).

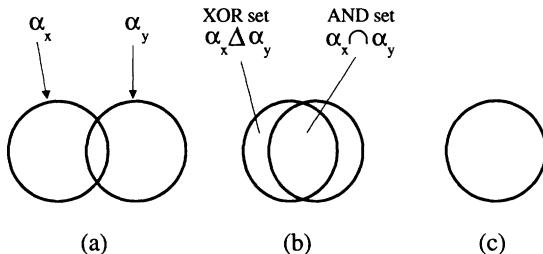


Figure 1: XOR and AND sets

Note that the degree of coincidence is proportional to the size of the set of attributes that are in the intersection set, $\alpha_x \cap \alpha_y$, and inversely proportional to the size of the symmetric difference of the two sets, $\alpha_x \Delta \alpha_y$. The *probabilistic* descriptor in *PMQ* emphasizes that this coincidence may be due to chance and merely measures an *increased probability* that a set is optimal or near optimal. These sets of attributes will be referred to, for obvious reasons, as the *XOR* set and the *AND* sets.

A useful *PMQ* is one that will modify the temperature in a manner consistent with the notion of *negative feedback control*—increasing the temperature when the quality of solutions is low, decreasing it when quality of solutions is high—an inverse relationship. Two types of *PMQs* may be used depending on the types of problems being solved. For two processors solving the *same COP*, the quality of solutions *increases* as the *XOR* set *decreases* and converges in probability to zero. The size of the *XOR* set can therefore be utilized as a coefficient of the temperature modifying it in a manner consistent with negative feedback. For complementary problems such as the *Maximum Independent Set (IS)* and *Minimum Vertex Cover (VC)*, the *AND* is useful for similar reasons—the sets of attributes *migrate apart* as in the progression from picture (c) to (b) to (a) in Figure 1 and the intersection set converges in probability to zero (see Bondy and Murty 1976 for a complete description of the relationship between IS and VC). These two problems will be the basis of our empirical study in Section 3.

Mathematically, the size of these *PMQs* corresponds to the value of $Z^{[k]}$ in $G(V, E)$. This value measures how far apart the solutions are in the configuration space and therefore can be used as an error signal for in a negative feedback control scheme.

By employing the concept of negative feedback control we are implicitly suggesting that there is some benefit from maintaining a situation in which the error is zero. In other contexts, this may mean a path of improvement, such as in steepest descent, that moves along a gradient. Here, we justify using $Z^{[k]}$ as an error signal because a zero error is associated with an *increased state probability* of the globally optimal solution as Theorem 1 shows.

Theorem 1 *Let $Z^{[k]}$ serve as an error signal in a PFC system. Then for time index k sufficiently high i.e., temperature t_k sufficiently low, $\Pr\{X^{[k]} = 0 \mid Z^{[k]} = 0\} > \Pr\{X^{[k]} = 0\}$.*

Proof: See Fleischer 1995.

The following theorem shows how control is effected by raising or lowering the temperature to modulate the entropy $h_i(t_k)$ of solution i .

Theorem 2 *For temperature t_k sufficiently small, then (i): for all i , $h_i(t_k + \Delta t) \geq h_i(t_k)$ and (ii): there exists an i , such that $h_i(t_k + \Delta t) > h_i(t_k)$.*

Proof: See Fleischer 1995.

In negative feedback control systems, the error signal must be processed and often amplified so as to have the proper effect on the actuation device so that control over the system is possible. If the error signal is amplified too much, however, then the actuation will be too great and will *overcorrect* the system causing it to *overshoot* the optimal state, often by a degree greater than the original error. This will cause the error signal to reverse its effect on the actuator causing the system to overshoot again, but in the opposite direction and to an even greater degree. Eventually, the system starts to wildly oscillate and control is impossible. This behavior, referred to as *hunting* (Wiener 1961), occurs when the *system gain* is too large and generally results in a breakdown of control over the system. This same type of behavior should therefore be observable in *PFC* when the temperature parameter is modulated too much. Under such circumstances we should expect a degradation in system performance.

To test this idea, the system gain can be set by modifying the temperature by PMQ_k/PMQ_0 , the ratio of the PMQs at time k and time 0, the initial value of the PMQ. This can be used as a coefficient of the cooling schedule to modulate the temperature parameter and at the same time preserve the limiting properties of the cooling schedule (see (1)), a condition necessary for Theorem 1. Setting PMQ_0 directly affects the system gain. For example, if $PMQ_0 = 2$, corresponding to two attributes, a change of one attribute, e.g., $PMQ_k = 3$, would increase the temperature by 50% whereas if $PMQ_0 = 100$, such a change in the PMQ would only increase the temperature by a mere 1%. The following cooling schedule incorporates these ideas

$$t'_k = \left(\frac{PMQ_k}{PMQ_0} + \theta \right) t_k = \left(\frac{d_{i_x i_y}}{d^*} + \theta \right) t_k \quad (1)$$

where $\theta > 0$ is the damping parameter and t_k is any cooling schedule guaranteeing convergence in probability. (If $\theta = 0$, then for any $t_k > 0$ fixed, there is a limiting distribution where all local minima, including the optima, become absorbing states. See Fleischer 1993 for a further description of what happens when the PMQ becomes zero.) This leads to the following theorem.

Theorem 3 *Given a two-processor PFC system with a symmetric neighborhood generation mechanism using (1), then the stationary distribution of the system at temperature t is given by*

$$\pi_{i_x i_y}(t) = \frac{1}{B_{xy}(t)} \exp \left(\frac{(-f_{i_x} - f_{i_y}) d^*}{t(d_{i_x i_y} + \theta)} \right) \quad (2)$$

where $B_{xy}(t) = \sum_{i_x} \sum_{i_y} \exp \left(\frac{(-f_{i_x} - f_{i_y}) d^*}{t(d_{i_x i_y} + \theta)} \right)$ is a normalization constant and $d^* > 0$ is a constant that serves as the gain setting. The transition probability of the system at temperature t_k where $j_x \neq i_x, j_y \neq i_y$ is

$$p_{i_x i_y, j_x j_y}^{[k]} = G_{i_x j_x} G_{i_y j_y} \exp \left(\frac{(-\Delta f_{j_x i_x} - \Delta f_{j_y i_y}) d^*}{t_k(d_{i_x i_y} + \theta)} \right) \quad (3)$$

Proof: See Fleischer 1995.

Equations (2) and (3) are very similar to the well-known stationary and transition probabilities of generic SA and retain the global and detailed balance properties of Markov chains. Thus, the conditions

necessary for a weakly ergodic system are preserved (Aarts and Korst 1989).

The advantages of the simple incorporation of the information embodied in $d_{i_x i_y}$ in the exponents in (2) and (3), *in light of Theorem 1*, are readily apparent. For smaller values of $d_{i_x i_y}$, the probability of moving from the current solution is lower and the stationary distribution of the current solution is higher than it would be without inclusion of $d_{i_x i_y}$. Conversely, the higher the value of $d_{i_x i_y}$, the greater the probability of moving from the current solution and the lower the stationary probability. The net effect is to shift the weight of the probability to those states where $d_{i_x i_y}$ is small thus increasing the stationary probability of optimal states (per Theorem 1). We believe this more fully takes advantage of the convergence properties of SA in an inherently fundamental way.

2.2 Implementation Techniques & Issues

A PFC scheme can be implemented by running two or more Acceptance Processors (APs), Π_x and Π_y , in parallel, each storing its version of the current solution in local memory. Each AP evaluates the Metropolis Acceptance Criterion (Metropolis *et al.* 1953) for a given candidate attribute *given its own version of the current solution*. The candidate attributes are selected by a candidate Generating Processor (GP) which couples the two (or more) APs. The two APs run concurrently and synchronously on the same or related problem. Figure 2, along with the following pseudo-code, illustrates how these processors are all connected and how information is routed in this PFC network.

```

GP   1: init_system();
      2: PMQ = get_PMQ();
      3: temp = calc_temp(PMQ);
      4: cand = select_candidate_attribute();
AP   5: Metropolis_Accept(cand, temp);
      6: goto GP;

```

Step 1 initializes the system by creating initial solutions which also influences the value of PMQ_0 . This is done by a value in a parameter file which controls a random number generator which, in turn, determines whether any particular attribute is to be included in the initial set or not. The larger the likelihood that an attribute is in the current sets for processors Π_x and Π_y , the larger the AND set and the smaller the XOR set will tend to be. This provides a way to experiment with the system gain

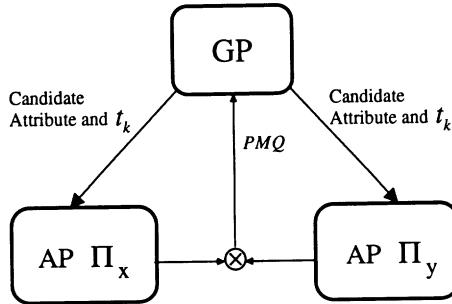


Figure 2: A Parallel Probabilistic Feedback Control Network

as described in Section 2.1. Step 2 begins the actual annealing process by first obtaining the PMQ and then using it to calculate the temperature parameter in Step 3. Step 4 involves the generation of a candidate attribute which, along with the temperature, are passed to the APs , in parallel, for processing of the Metropolis Acceptance Criterion. Note that these steps, the initialization procedure, the candidate generation procedure, and the Metropolis Acceptance procedure, are all basic to any generic SA implementation. The only difference here is that the Metropolis Acceptance procedure can be done in parallel on numerous processors and the calculation of the temperature is modified by the PMQ .

2.3 Candidate Selection

The selection of a candidate attribute by the GP can be done in several ways depending upon the type of problem being solved. Here we describe the *common candidate method* in which a single candidate attribute is selected by the GP and communicated to each AP during each iteration.

2.4 Computation of the PMQ Based on Attributes

The symbol \otimes in Figure 2 indicates that the size of the PMQ is obtained from the current sets of attributes and is readily computed using boolean logic in similar fashion as Δf_{ji}^+ is for many problems (Aarts and Korst 1989). After the determination of PMQ_0 from the initial solutions, the value of ΔPMQ is calculated thereafter. Thus, instead of rescanning the

entire graph in each processor's memory during each iteration, only the value of ΔPMQ is necessary and can be computed using the following approach. Define the following indicator variable for candidate attribute v :

$$I_{\alpha_x}^{[k]}(v) = \begin{cases} 1 & \text{if } v \in \alpha_x \text{ at time index } k \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

Then,

$$PMQ_{k+1} = PMQ_k + \Delta PMQ_{k+1} \quad (5)$$

and the PMQ can be computed using the following:

$$\Delta PMQ_{k+1} = I_{\alpha_x}^{[k+1]}(v) \times I_{\alpha_y}^{[k+1]}(v) - I_{\alpha_x}^{[k]}(v) \times I_{\alpha_y}^{[k]}(v) \quad (6)$$

$$\Delta PMQ_{k+1} = I_{\alpha_x}^{[k+1]}(v) \oplus I_{\alpha_y}^{[k+1]}(v) - I_{\alpha_x}^{[k]}(v) \oplus I_{\alpha_y}^{[k]}(v) \quad (7)$$

where (6) and (7) are used to calculate the *AND* and *XOR* sets, respectively. This permits calculation of the PMQ in $\mathcal{O}(1)$ time.

3. An Example: The IS & VC Problems

To illustrate a *PFC* system, consider two *COPs*, the *VC* problem and the *IS* problem described earlier. The *attributes* for these *COPs* correspond to *nodes* in a graph, combinations of which yield certain objective function values. Because these two problems for the same graph are complementary (Bondy and Murty 1976), the PMQ will be based on the *AND* set, the intersection of the current independent set α_k , and the current vertex cover β_k . The cooling schedule used here is based on Mitra, *et al.* (1986) but modified by the PMQ :

$$t_k = \left(\frac{|\alpha_k \cap \beta_k|}{|\alpha_0 \cap \beta_0|} + \theta \right) \frac{\gamma}{\log(c+k)} \quad (8)$$

where the *AND* set is indicated by $|\alpha_k \cap \beta_k|$.

3.1 Experimental Methodology & Results

To test whether the notion of *PFC* has any merit at all, we implement such a system and compare the convergence rate of it to the convergence rate of generic *SA*. The following hypothesis test is used: $H_0 : \langle f_{PFC} \rangle = \langle f \rangle$ where $\langle f_{PFC} \rangle$ is the expected value of the final state *with* feedback control, $\langle f \rangle$ is the expected value of the final state *without* feedback control. These values are estimated using the statistics $\overline{f_{PFC}}$ and \overline{f} , the average values of the final states with and without *PFC*, respectively. The expected value of the final state is a measure on the convergence of *SA*

because SA converges in distribution (Mitra, *et al.* 1986). The estimated variances s_{PFC}^2, s^2 of the final state were also calculated. Finally, the z and p -values associated with these statistical tests are also computed (the z values were calculated using the following test statistic for unknown variances with $n = 100$, *i.e.*, 100 replications of SA experiments, each 250 iterations long):

$$z = \frac{\overline{f}_{\text{PFC}} - \overline{f}}{\sqrt{\frac{s_{\text{PFC}}^2}{n} + \frac{s^2}{n}}} \quad (9)$$

The test bed used was a graph with 30 nodes and randomly generated arcs. The initial temperature, $t_1 = 70$, the final temperature $t_{250} = 1$ using (8) with $\theta = 0$. The system gain is based on the value of $|\alpha_0 \cap \beta_0|$. The following gain descriptors are associated with the *expected size* of the initial AND sets: *low gain*–24.3 nodes, *medium gain*–19.2 nodes, *high gain*–7.5 nodes. Table 1 summarizes the results for these gain settings where each pair of rows corresponds to a IS-VC pair and indicates the statistical results for both coupled and uncoupled SA processors using the same random number stream.

Table 1: Statistical Results

Processor II	Gain	w/PFC		w/o PFC		z -value	p -value
		$\overline{f}_{\text{PFC}}$	s^2	\overline{f}	s^2		
IS	Low	9.49	2.65	3.90	7.93	17.18	$< 10^{-3}$
VC	Low	20.75	1.87	25.77	4.18	-20.41	$< 10^{-3}$
IS	Med	9.50	6.67	4.49	7.69	13.22	$< 10^{-3}$
VC	Med	21.17	3.07	25.94	3.67	-18.37	$< 10^{-3}$
IS	High	8.71	8.57	5.94	7.76	6.85	$< 10^{-3}$
VC	High	24.08	12.04	27.02	8.04	-6.56	$< 10^{-3}$

These numbers show a very apparent benefit when a feedback mechanism is in place. For example, the values in the first two rows indicate the mean value of the final state and its estimated variance for IS and VC using low gain PFC. For this gain setting, the mean values for the IS and VC sets were 9.49 and 20.75, respectively. These values were tested against the uncoupled (no feedback mechanism used) IS and VC values of 3.90 and 25.77, respectively, resulting in z values of 17.18 and

–20.41. From these data, the IS solutions *with PFC* were higher than the corresponding solutions *without PFC*. For the VC problem, the solutions were correspondingly *lower* with PFC than without PFC (bear in mind that IS is a *maximization* problem and VC is a *minimization* problem). These results therefore indicate that using PFC leads to superior performance on these problems.

3.2 Non-Monotonic Cooling Schedules

Glover (1995) suggests that non-monotonic trajectories for various search techniques may provide superior performance than monotonic trajectories. In the context of SA, such trajectories would naturally be associated with the cooling schedule (Osman 1993). The use of PFC in SA leads to such non-monotonic cooling. Figure 3 illustrates how the temperature for a typical SA run responds to perturbations in the error signal in its search for the optimal solution. Note that the temperature curve using a high system gain tends to fluctuate more wildly about the uncoupled, monotonic cooling schedule while the low gain curve is much smoother and exhibits superior performance (as evidenced by comparing the z values obtained using low gain and high gain in Table 1).

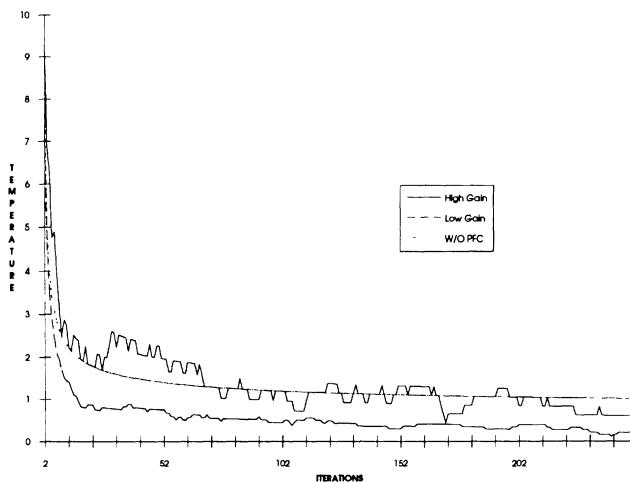


Figure 3: Non-Monotonic Cooling Using PFC

4. Conclusion

This paper has introduced the concept of cybernetic optimization using probabilistic feedback control in simulated annealing. By showing a connection between the basic elements of control theory and how they can be applied in a probabilistic optimization scheme, a new way of implementing parallel processing concepts has been described. By taking advantage of the convergence properties of SA, several parallel processors can generate a probabilistic error signal. This error signal is used to modulate that which causes SA to converge in probability—the temperature control parameter. This directly affects the entropy of the search mechanism and increases the steady-state probabilities of the globally optimal solutions.

These ideas were tested experimentally using an implementation of SA to solve the independent set and vertex cover problems. The experiments show that *PFC* is a viable method for implementing parallel SA in a probabilistic feedback control network.

5. Acknowledgements

This research was partially supported by the National Aeronautics and Space Administration, Contract NAS1-19858-13 (first author), and the National Science Foundation, Contract DMI-94-09266 and the Air Force Office of Scientific Research, Contract F49620-95-1-0124 (second author).

6. References

- Aarts, E. and J. Korst (1989), *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*, New York, N.Y. John Wiley & Sons.
- Azencott, R. ed. (1992), *Simulated Annealing: Parallelization Techniques*, New York, N.Y., John Wiley & Sons, Inc.
- Bondy, J.A. and U.S.R. Murty (1976), *Graph Theory with Applications*, New York, N.Y. North-Holland, Elsevier Science Publishing Co., Inc.

- Eglese, R.W. (1990), "Simulated Annealing: A Tool for Operational Research", *European Journal of Operational Research*, Vol. 46, No. 3. June 15, 1990. pp.271-281.
- Fleischer, M. (1993), "Assessing the Performance of the Simulated Annealing Algorithm Using Information Theory", Doctoral dissertation, Case Western Reserve University.
- Fleischer, M. (1995), "Cybernetic Optimization By Simulated Annealing: Accelerating Convergence By Parallel Processing and Probabilistic Feedback Control", Technical Report. Department of Engineering Management, Old Dominion University.
- Frost, R., N. Priebe, P. Salamon (1993) "An Ensemble Based Simulated Annealing Library For Multiprocessors". Working paper.
- Glover, F. (1995), "Tabu Thresholding: Improved Search By Non-monotonic Trajectories", to appear in *INFORMS Journal on Computing*.
- Ingber, L. (1995) "Adaptive Simulated Annealing: Lessons Learned", *Journal of Control and Cybernetics*. To appear.
- Manber, U. (1989), *Introduction to Algorithms: A Creative Approach*, New York, N.Y. Addison-Wesley Publishing Company.
- Metropolis, N., A. Rosenbluth, M. Rosenbluth, A. Teller and E. Teller (1953), "Equation of State Calculations by Fast Computing Machines", *Journal of Chemical Physics*, Vol. 21, pp. 1087 - 1092.
- Mitra, D., F. Romeo and A. Sangiovanni-Vincentelli (1986), "Convergence and Finite-Time Behavior of Simulated Annealing", *Advances in Applied Probability*, Vol. 18, pp. 747-771.
- Osman, I. (1993), "Metastrategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem", *Annals of Operations Research*, Vol. 41, pp.421.
- Reeves, C. ed. (1993) *Modern Heuristic Techniques for Combinatorial Problems*. New York, John Wiley & Sons.
- Wiener, N. (1961), *Cybernetics: or Control and Communication in the Animal and the Machine*, 2nd ed. Cambridge, Massachussetts. The M.I.T. Press.

A simulated annealing algorithm for the computation of marginal costs of telecommunication links

J.L. Lutton and E. Philippart
France Telecom CNET PAA/ATR
38-40 rue du général Leclerc
92131 Issy-les-Moulineaux France
E-mail : jean-luc.lutton@issy.cnet.fr
emmanuelle.philippart@issy.cnet.fr

July 5,1995

Abstract

The aim of this study is the calculation of marginal costs in a transmission network. A classical network model enables the marginal costs of links to be defined for any given topology. The limits of a computation linked to a particular optimal network configuration led us to consider statistical mechanics methods originally used in combinatorial optimization. Probabilistic properties such as thermostatistical persistency ensure the convergence of a Metropolis based algorithm adapted from network design and dimensioning to the computation of marginal costs.

Key Words : Marginal costs, Simulated annealing algorithms, Thermostatistical persistency properties

1 Introduction

Within the general trend of the telecommunication field seeking economical efficiency, France will be entirely open to competition by 1998. For such a competitive environment, pricing will be a strategical issue. Historically prices have been fixed according to political goals, see Encaoua (1987). In the early phase, the telecommunication supplier had to increase the number of subscribers, so prices were at first very low, far from the real costs involved. Later, in order to satisfy universal service constraints, local networks had to be interconnected together. In the current more incentive phase, the number of subscribers remains stable but traffic demands are growing. Consequently, the present objective is to optimize the use of networks with respect to both ressource allocation and pricing policy. Furthermore, the evolution of telecommunication techniques keeps on generating important economies of scale which strengthens the argument that a uniform pricing is not satisfactory : in a competitive environment, suppliers seek cost oriented pricing in order to limit opportunities for competition.

A regulatory agency is also needed to respect the maximization of the general welfare that is consumers and producers surplus. Specifically when regulators seek guidance on pricing rules for a regulated firm, the well known pricing principle that characterizes competitive equilibrium tells them to compare price to marginal costs, see Baumol (1994). The marginal cost of a product may be interpreted as the true

cost a consumer imposes upon an economy in buying an additional unit of product. In mathematical terms, it is the derivative of total cost with respect to the output.

Dealing with a telecommunication network, marginal costs are defined for a given type of service between a set of customers, in relation to the number of traffic units to be routed in the network, see Morgan (1976). A global network marginal cost computation could not take into account the local costs involved in the network between customers. So, marginal costs have to be described as a symmetric matrix mc , where $mc(i, j)$ represents the incremental cost of a traffic unit between any pair of customers i and j . The value $mc(i, j)$ is also called the marginal cost of the link (i, j) .

The complexity of marginal cost computation in a telecommunication network comes from two different points. First, considering a real size network –say about twenty nodes– the routing cost of one given traffic demand in the network is marginal compared to the network total cost, and more often than not inferior to the accuracy of this type of network computing tools. Thus, one can not compute marginal costs as the incremental cost generated with an incremental demand. Secondly, marginal costs computed on the traffic routing paths remain approximate since the network configuration may change for any additional unit of traffic. These costs are tightly linked to the network topology. Out of practice, previous studies concerning transmission network optimization showed us that there are a lot of networks of equivalent total cost which topologies are quite different. This point is developed in the following section. When the computation of marginal costs is independent from a specific network topology, it is a useful estimation for the regulator in a competitive environment as a lower bound for a per unit traffic pricing.

This study is restricted to a transmission system network, which optimization has already been the aim of previous studies, see Sutter (1992). Here we present a simulated annealing based algorithm for optimal network calculation adapted for the computation of marginal costs. A classical network model is first presented so as to define marginal costs of links for a given graph design. The Metropolis procedure provides an interesting tool for their computation for a fair pricing guidance. Then, the convergence properties from statistical mechanics methods are described in the third section. Finally some results are set out.

2 The marginal costs definition

The network marginal costs are defined according to a classical model in which the network is connected to an undirected graph $G = (X, E)$, where X is the set of nodes and E the set of edges –routing abilities in between nodes. Typically, traffic demands defined as a deterministic symmetric matrix have to be routed along the graph edges between each endpoints. Endpoints are the nodes defined as origin or destination of at least one traffic demand. We assume here that each edge has a binomial cost : a fixed cost when opening the edge, and a linear cost which is a per unit traffic cost.

Given a network design, the optimal solution corresponds to the shortest path traffic routing between each origin and destination node. A shortest path is the set of edges enabling a path between two nodes which minimizes the sum of each corresponding linear cost. Thus, given a network topology $G = (X, E)$, the marginal

costs of the network links $mc_G(i, j)$ are defined by the following expression :

$$\forall (i, j) \in X^2, mc_G(i, j) = \begin{cases} lc(i, j) & \text{if } (i, j) \in E, \\ \sum_{(p, q) \in A_{i,j}} lc(p, q) & \text{otherwise.} \end{cases}$$

where :

- $A_{i,j} = \{ (p, q) \in E / (p, q) \text{ belongs to the shortest path between } i \text{ and } j \}$
- $lc(i, j)$ linear cost of facility (i, j) .

This definition induces :

$$\forall (i, j) \in X^2, lc(i, j) \leq mc_G(i, j)$$

This matrix can be computed for any given topology, so it would be possible to consider the optimal network design previously obtained with an optimization heuristic. A simple example can show the limits of such a strategy. Let us consider for instance a traffic demand between two French interconnection nodes Clermont and Lyon (Figure 1). Various optimization processes led us to several equivalent network configurations like the two following cases :

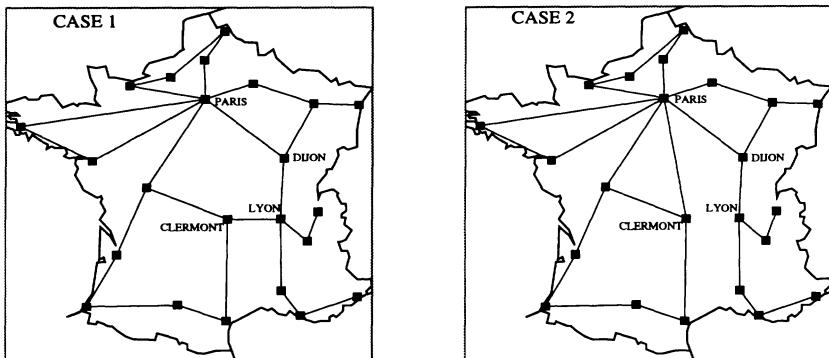


Figure 1: Influence of optimal network design on links marginal costs calculation

- case 1 : a direct link exists between the two nodes (Figure 1, case 1)

The traffic routed on the shortest path will certainly be carried across the direct link, with a marginal cost equal the link linear cost :
 $lc(Clermont, Lyon)$.

- case 2 : a direct link doesn't exist (Figure 1, case2)

The traffic has to be routed through other nodes belonging to the shortest path, so for instance via Paris and then Dijon. So the marginal cost of the link $(Clermont, Paris)$ is the sum of the linear costs of the following edges :
 $lc(Clermont, Paris)$, $lc(Paris, Dijon)$ and $lc(Dijon, Lyon)$.

Since linear costs are tightly linked to distance between nodes, it can be assumed that :

$$lc(Clermont, Lyon) \ll lc(Clermont, Paris) + lc(Paris, Dijon) + lc(Dijon, Lyon)$$

So a customer offering traffic on this link is penalized if the optimal network design chosen by the service supplier is close to case 2. Computed this way, marginal costs depend much on the optimal network topology chosen, and it should not be imposed on customers. In fact, in a competitive environment, any rival supplier building the direct link would be able, considering the same pricing method, to provide a cheaper link pricing.

Thus the marginal cost computing based on the optimal network design is not satisfactory.

3 Statistical mechanics properties and the Metropolis procedure

The Metropolis procedure, see Metropolis (1953), is at the outset of the simulated annealing techniques used for combinatorial optimization, see Bonomi (1984) and Chardaire (1993). This method has been originally developed to solve problems of energy minimization in statistical mechanics. Some of its probabilistic properties are interesting tools for optimization problems, see Lutton (1985).

In statistical mechanics, a physical system submitted to interacting components is completely defined by the set of internal configurations –or states– and a function giving the energy associated to each configuration. The dynamic of the system internal evolution is assumed to generate a flow such that the invariant measure described during the evolution is proportional to the Boltzmann factor. So :

$$p(x) \simeq \exp\left\{\frac{-H(x)}{T}\right\}$$

where :

- $p(x)$: system probability to be in the state x
- $H(x)$: energy of state x
- T : temperature of the surrounding heatbath.

$p(x)$ enables the measure of the internal states distribution in terms of energy level of the current system. As temperature decreases, this distribution is sharpened around to the ground state, namely the lowest energy state. When T drops to 0, the system reaches the ground state minimizing the internal energy.

Considering combinatorial optimization, each admissible solution is a potential state of the system and a cost function describes the energy of the system. Thus, the Metropolis procedure generates a sequence of solutions with a visiting probability $p(x)$ which describes a stationary process .

The running state of the system corresponds to a configuration c of the network where :

- $c = (c_1, c_2, \dots, c_N)$
- N maximum number of edges belonging to c
- each edge is described as a boolean variable equals to 1 when the edge belongs to the current network configuration, that is :

$$c_i = \begin{cases} 1 & \text{if } i^{\text{th}} \text{ edge belongs to } c \\ 0 & \text{otherwise} \end{cases}$$

- $f(c)$ is the total cost corresponding to configuration c

A simulated annealing algorithm can be easily implemented with basic transitions like removing, adding or exchanging edges, see Sutter (1992). The stationary distribution of the generated Markov chain, built from the successive states reached and described in the Metropolis algorithm, tends to the Boltzmann distribution $Q_T(c)$:

$$Q_T(c) = \frac{e^{-\frac{f(c)}{T}}}{\sum_{c \in C} e^{-\frac{f(c)}{T}}}$$

The expectation of the variable c_i called the thermostatistical persistency, see Chardaire 1992, of the variable c_i is according to the Boltzmann distribution :

$$E_T(c_i) = \sum_{c \in C} c_i Q_T(c)$$

From the Boltzmann properties, the generated random walk converges to an optimal solution when temperature decreases, so if C^* is the set of optimal configurations, then :

$$\lim_{T \rightarrow 0} Q_T(c) = \begin{cases} \frac{1}{|C^*|} & \text{if } c \in C^* \\ 0 & \text{otherwise} \end{cases}$$

Initialy, physicists have used Monte-Carlo procedures like Metropolis algorithm to sample the states of a system according to the Boltzmann probability distribution.

At a given temperature, any average value of the system can be computed. Thus, let us call u any variable representing the system.

For a given temperature T , the mean value $\langle u \rangle_T$ can be easily performed :

$$\langle u \rangle_T = \sum_{c \in C} \frac{u(c) e^{-\frac{f(c)}{T}}}{\sum_{c \in C} e^{-\frac{f(c)}{T}}}$$

where :

- c is a system configuration
- C is the set of all possible configurations of the system
- f is the cost function

From the properties previously recalled, during the optimization procedure :

$$\lim_{T \rightarrow 0} \langle u \rangle_T = \frac{1}{|C^*|} \sum_{c \in C^*} u(c)$$

Applied to the marginal cost computation, if $mc_{(T,k)}(i,j)$ is the value computed for the temperature T and iteration number k , this property ensures the convergence of average values $mct(i,j)$ performed for each temperature step, upon an equivalent total cost configurations set :

$$\langle mc \rangle_T(i,j) = \frac{\sum_{k=1}^{kmax} mc_{(T,k)}(i,j)}{kmax}$$

where $kmax$ is equal to the number of iterations performed for each temperature step; in practice $kmax$ increases as temperature decreases.

Consequently :

$$\lim_{T \rightarrow 0} \langle mc \rangle_T(i,j) = mc^*(i,j)$$

Finally, the computation of the limit marginal costs of links $mc^*(i,j)$, computed with mean value upon a set of equivalent optimal configurations, contribute to limit the influence of the optimal network design.

4 Simulated annealing based algorithm for marginal cost computation

A network design and dimensioning algorithm based on Metropolis procedure has been developed to compute average marginal costs of links. Beginning from the complete graph, the algorithm leads to manage some basic transitions, three of them are considered here : to add, remove or exchange edges. The network total cost is computed for each transition proposed. If a transition decreases the total cost, then it is accepted as a new graph configuration, otherwise its acceptance is limited with a temperature parameter. As temperature tends towards zero during the optimization procedure, the acceptance probability decreases. For each temperature set, a large number of iterations is performed in which each type of transition is proposed, in order to reach thermic equilibrium. Given any graph design G , that is for each (temperature step T , iteration number k) one can compute the marginal costs matrix of the current system $mct_{(T,k)}(i,j)$ ($= mc_G(i,j)$), then it still remains to average marginal costs for each link.

The algorithm performance depends tightly on the computation time needed for the estimation of the new configuration cost. This task is divided into two main parts : the first one consists in recomputing the new shortest paths, and the second one in rerouting the traffic on the new paths defined. An enhanced method has been developed in order to limit computation time specifically for the first part. Thus, after any transition a Moore-Dijkstra algorithm is only applied to a subgraph, built recursively with adjacent nodes to initial edge endpoints. When transition consists in removing an edge only nodes which shortest paths are involved are inserted in

the subgraph, and when adding an edge the only nodes considered are those which shortest paths could be reduced. Exchanging edges is almost the same problem since it can be considered like adding and removing two different edges.

5 Results

The problem considered has 36 nodes, quite representative of the french interconnection network. About 200 flows to be routed from node to node, set the demand matrix. The cost function involves only transmission costs. The optimal network defined with the design and dimensioning heuristic has 48 edges (Figure 2).

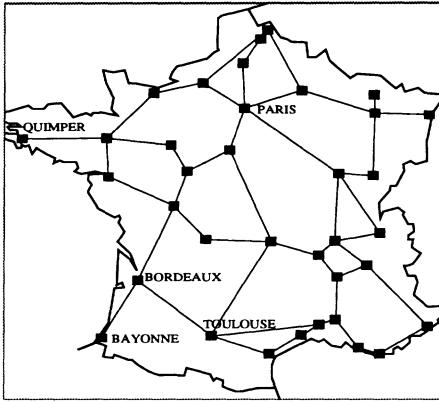


Figure 2: Optimal network design

The evolution of mean marginal costs $mct(i, j)$ for various temperature steps becomes quickly stable during the optimization although theoretically the real value is only reached when temperature tends to zero (Figure 3).

As a matter of fact, for medium temperatures ($T \simeq 0.1$) which means that the random walk generated among the solution set by the simulated annealing algorithm is far to be only made of optimal solutions, almost all the marginal costs have reached their limit values. Therefore, even if the current network design is very different from the optimal one, a reliable approximation of marginal costs is already obtained. This significantly shows that the values performed do truely depend on the circuits demand necessary for the network flow routing, but are quite independent from the optimal network topology. Let us highlight also that all types of evolutions may occur relatively to the type of link considered.

As marginal costs over-estimate the corresponding linear costs, we looked at this gap for a few randomly chosen links (Figure 4).

Initally, the marginal costs computed at high temperatures are very far from the corresponding direct link linear costs. When temperature decreases, two kinds of situations may occur. Some marginal costs quickly converge to corresponding linear costs (Figure 4, paths between Bordeaux and Bayonne and between Bordeaux and Toulouse). This means that those two edges or paths belong to most of the optimal

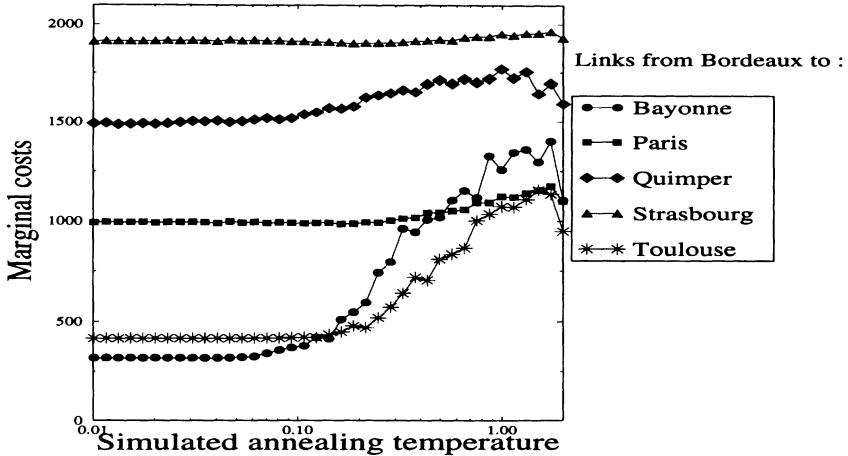


Figure 3: Evolution of average marginal costs during the Metropolis procedure

solutions. On the contrary, some marginal costs may remain independent values and this shows how the current local network configuration corresponds in fact to an arbitrary choice among many equivalent optimal configurations (Figure 4, paths between Bordeaux and Strasbourg and between Bordeaux and Paris).

But the peculiar nature of this convergence consists in its swiftness. The stability of these marginal costs induces the final existence of the direct links in the optimal network very early in the optimization process. Reciprocally, the probability to have a direct link is very low when the third part of the optimization begins, the gap is still important.

Comparing the limit marginal costs to the marginal costs defined by the optimal network design is a means to confirm the efficiency of the Metropolis procedure as an independent computation from the final optimal network configuration chosen (Figure 5). Thus, values obtained for the path between Bordeaux and Quimper show that a significant gap can be measured.

Such type of graphs showing the marginal cost evolution within a wide range of temperature needs a long simulation time. Nevertheless, the properties and swift convergence of the method enables us to obtain reliable approximations of marginal costs for medium temperatures with few iterations, so the method is very few CPU time consuming.

6 Conclusion

A simulated annealing based network design and dimensioning algorithm has been adapted to compute marginal costs of transmission links. The Metropolis procedure allows to determine limit marginal costs independently from the network optimal topology chosen by the telecommunication supplier. The values performed become quickly stable during the optimization process which proves the tight relationship between marginal costs and the need in circuits for the demand routing. This

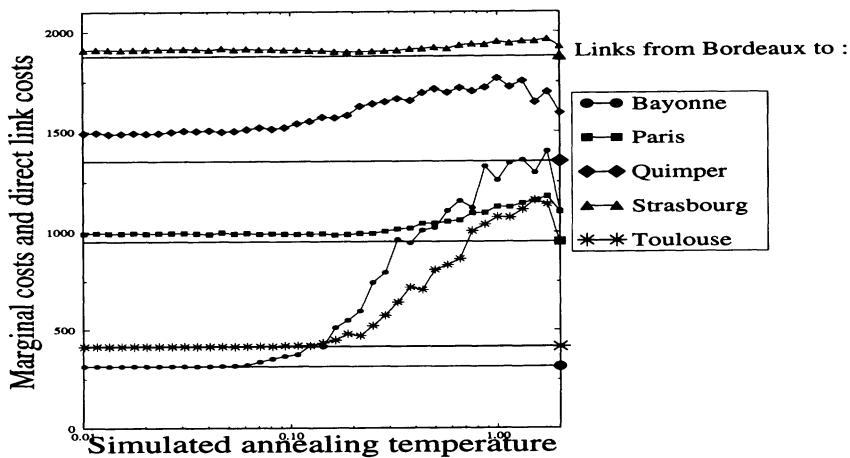


Figure 4: Comparison between marginal costs and linear costs

method applied to real data could be an interesting tool for practical network cost sharing analysis.

Marginal costs do not take into account fixed costs involved in the network total cost function. Therefore in order to define a sustainable pricing, the remaining fixed costs have to be shared between the various demands. This problem will be dealt in a further study.

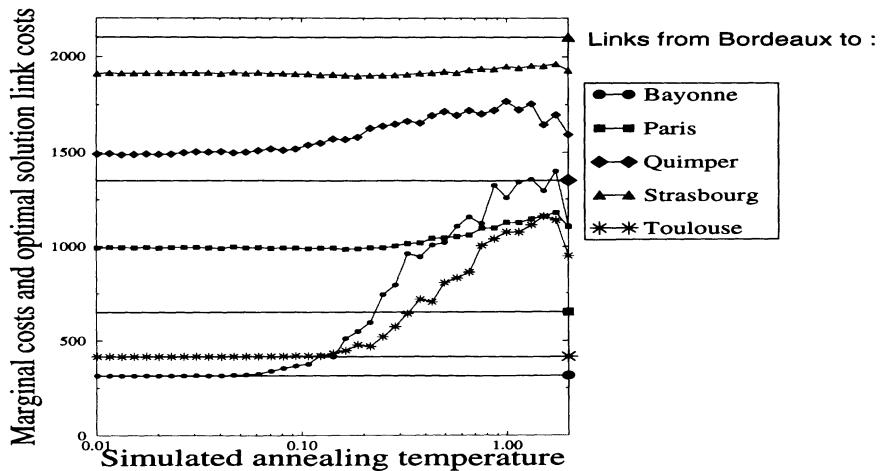


Figure 5: Comparison between marginal costs and optimal network design marginal costs

REFERENCES

- W.J. Baumol and J.G. Sidak, *Toward Competition in Local Telephony*, (The MIT Press, Chapter 3, 1994).
- E. Bonomi and J.L. Lutton, The N-City Travelling Salesman Problem : Statistical Mechanics and the Metropolis algorithm, *SIAM Review 26/4*, (1984), pages 551–568.
- P. Chardaire and J.L. Lutton, Using Simulated Annealing to Solve Concentrator Location Problems in Telecommunication Networks, *Applied Simulated Annealing, Lecture notes in Economics and Mathematical System, Springer Verlag, 1993*.
- P. Chardaire and J.L. Lutton and A. Sutter, Thermostatistical Persistency : a Powerful Improving Concept for Simulated Algorithms, to appear in *EJOR*, 1995.
- N. Curien and M. Gensollen, *Les Théories de la Demande de Raccordement Téléphonique*, (Revue Economique : Economie des Télécommunications, ENSPTT Economica, Presses de la Fondation Nationale des Sciences Politiques, 1987).
- D. Encaoua and M. Moreaux, *L'Analyse Théorique des Problèmes de Tarification et d'Allocation de Coûts dans les Télécommunications*, (Revue Economique : Economie des Télécommunications, ENSPTT Economica, Presses de la Fondation Nationale des Sciences Politiques, 1987).

S.C. Littlechild, *Elements of Telecommunication Economics*, (Peter Peregrinus LTD, The University of Birmingham, England, 1979).

J.L. Lutton, Utilisation de Méthodes de Mécanique Statistique pour Étudier des Systèmes de Télécommunication et Traiter des Problèmes de Recherche Opérationnelle, France Telecom CNET, NT CNET/PAA/ATR/SST/1675, 1985.

N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller and E. Teller, Equation of State Calculations by Fast Computing Machines, *J.Chem.Phys.* 21, 1953.

T.J. Morgan CGIA CEng. FIEE, *Telecommunications Economics*, (Technicopy Limited, chapter 4, 1976).

A. Sutter, P. Chardaire and J.L. Lutton, A Simulated Annealing Algorithm for the Minimum Cost Multicommodity Flow Problem, France Telecom CNET, NT CNET/ PAA/ ATR/ ORI/ 3043, 1992.

Learning to Recognize (Un)Promising Simulated Annealing Runs: Efficient Search Procedures for Job Shop Scheduling and Vehicle Routing

Norman M. Sadeh

Carnegie Mellon University

Yoichiro Nakakuki

NEC

Sam R. Thangiah

Slippery Rock University

Abstract:

Simulated Annealing (SA) procedures can potentially yield near-optimal solutions to many difficult combinatorial optimization problems, though often at the expense of intensive computational efforts. The single most significant source of inefficiency in SA search is the inherent stochasticity of the procedure, typically requiring that the procedure be rerun a large number of times before a near-optimal solution is found. This paper describes a mechanism that attempts to learn the structure of the search space over multiple SA runs on a given problem. Specifically, probability distributions are dynamically updated over multiple runs to estimate at different checkpoints how promising a SA run appears to be. Based on this mechanism, two types of criteria are developed that aim at increasing search efficiency: (1) a *cutoff criterion* used to determine when to abandon unpromising runs and (2) *restart criteria* used to determine whether to start a fresh SA run or restart search in the middle of an earlier run. Experimental results obtained on a class of complex job shop scheduling problems show (1) that SA can produce high quality solutions for this class of problems, if run a large number of times, and (2) that our learning mechanism can significantly reduce the computation time required to find high quality solutions to these problems. The results also indicate that, the closer one wants to be to the optimum, the larger the speedups. Similar results obtained on a smaller set of benchmark Vehicle Routing Problems with Time Windows (VRPTW) suggest that our learning mechanisms could help improve the efficiency of SA in a number of different domains.

Key Words: Simulated annealing, learning, scheduling, vehicle routing.

1 Introduction

Simulated Annealing (SA) is a general-purpose search procedure that generalizes iterative improvement approaches to combinatorial optimization by sometimes accepting transitions to lower quality solutions to avoid getting trapped in local minima [12, 3]. SA procedures have been successfully applied to a variety of combinatorial optimization problems, including Traveling Salesman Problems [3], Graph Partitioning Problems [10], Graph Coloring Problems [11], Vehicle Routing Problems [19], Design of Integrated Circuits, Minimum Makespan Scheduling Problems [14, 18, 31] as well as other complex scheduling problems [35], often producing near-optimal solutions, though at the expense of intensive computational efforts.

The single most significant source of inefficiency in SA search is the inherent stochasticity of the procedure, typically requiring that the procedure be rerun a large number of times before a near-optimal solution is found. Glover et al. developed a set of "Tabu" mechanisms that can increase the efficiency of SA and other neighborhood search procedures by maintaining a selective history of search states encountered earlier during the *same run* [7]. This history is then used to dynamically derive "tabu restrictions" or "aspirations", that guide search, preventing it, for instance, from revisiting areas of the search space it just explored. This paper describes a complementary mechanism that attempts to learn the structure of the search space *over multiple runs* of SA on a given problem. Specifically, we introduce a mechanism that attempts to predict how (un)promising a SA run is likely to be, based on probability distributions that are refined ("learned") over multiple runs. The distributions, which are built at different checkpoints, each corresponding to a different value of the temperature parameter used in the procedure, approximate the cost reductions that one can expect if the SA run is continued below these temperatures. Two types of criteria are developed that aim at increasing search efficiency by exploiting these distributions:

- *A Cutoff Criterion:* This criterion is used to detect runs that are unlikely to result in an improvement of the best solution found so far and, hence, should be abandoned;
- *Restart Criteria:* When completing a run or abandoning an unpromising one, these criteria help determine whether to

start a fresh SA run or restart search in the middle of an earlier promising run.

The techniques presented in this paper have been applied to a class of complex job shop scheduling problems first described in [23]. Problems in this class require scheduling a set of jobs that each need to be completed by a possibly different due date. The objective is to minimize the sum of tardiness and inventory costs incurred by all the jobs. This class of problems is known to be NP-complete and is representative of a large number of actual scheduling problems, including Just-In-Time factory scheduling problems [23, 24]. Experimental results indicate (1) that SA can produce high quality solutions for this class of problems, if run a large number of times, and (2) that our learning mechanism can yield significant reductions in computation time. The results further indicate that, the closer one wants to be to the optimum, the larger the speedups. To test the generality of our learning mechanism, we also ran similar experiments with a SA procedure developed for the Vehicle Routing Problem with Time Windows (VRPTW). While preliminary, results on this second class of problems suggest that, indeed, our learning mechanism could help improve the efficiency of SA search in a number of different domains.

The balance of this paper is organized as follows. Section 2 quickly reviews fundamentals of SA search. Section 3 analyzes the behavior of typical SA runs and introduces a mechanism that aims at learning to recognize (un)promising runs on a given problem, using the concept of *Expected Cost Improvement Distributions (ECID)*. In Section 4, we use *ECID* distributions to develop a *cutoff criterion* to determine when to abandon unpromising runs. Section 5 presents three *restart criteria* based *ECID* distributions. Experiments obtained on a set of benchmark job shop scheduling problems with tardiness and inventory costs are reported in Section 6. A summary is provided in Section 7.

2 Simulated Annealing Search

Figure 1 outlines the main steps of a SA procedure designed to find a solution $x \in S$ that minimizes a real-valued function, $\text{cost}(x)$. The procedure starts from an initial solution x_0 (randomly drawn from S) and iteratively moves to other neighboring solutions, as deter-

mined by a neighborhood function, $neighbor(x)$, while remembering the best solution found so far (denoted by s). Typically, the procedure only moves to neighboring solutions that are better than the current one. However, the probability of moving from a solution x to an inferior solution x' is greater than zero, thereby allowing the procedure to escape from local minima. $rand()$ is a function that randomly draws a number from a uniform distribution on the interval $[0, 1]$. The so-called temperature, T , of the procedure is a parameter controlling the probability of accepting a transition to a lower quality solution. It is initially set at a high value, T_0 , thereby frequently allowing such transitions. If, after N iterations, the best solution found by the procedure has not improved, the temperature parameter T is decremented by a factor α ($0 < \alpha < 1$). One motivation for progressively lowering the temperature is to obtain convergence. Additionally, as the procedure slowly moves towards globally better solutions, accepting transitions to lower quality solutions becomes increasingly less attractive. When the temperature drops below a preset level T_1 , the procedure stops and s is returned (not shown in Figure 1).

```

 $T = T_0; x = x_0 (\in S); min = \infty;$ 
while ( $T > T_1$ ) {
    for  $i = 1, N$  {
         $x' = neighbor(x);$ 
        if ( $cost(x') < cost(x)$ )  $x = x';$ 
        else if ( $rand() < exp\{(cost(x) - cost(x'))/T\}$ )  $x = x';$ 
        if ( $cost(x) < min$ )  $min = cost(x), s = x;$ 
    }
    if ( $Min$  was not modified in the above loop)  $T = T * \alpha;$ 
}

```

Fig. 1 Basic Simulated Annealing Procedure.

Fig. 2 depicts the cost distribution of the best solutions returned by 300 SA runs on a typical combinatorial optimization problem – a job shop scheduling problem from a set of benchmarks to be

described in Section 6.

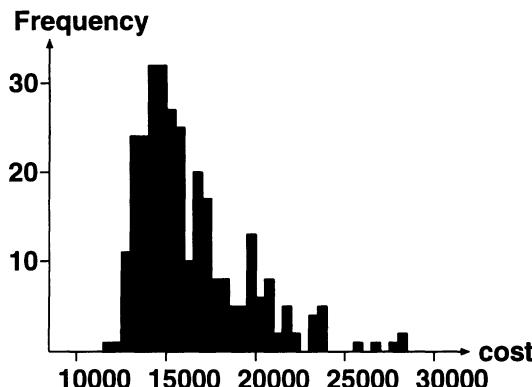


Fig. 2 Cost Distribution of the Best Solutions Found by 300 SA Runs.

The optimal solution for this problem is believed to have a cost around 11,500 – the value in itself is of no importance here. Figure 2 indicates that, if run a large number of times, SA is likely to eventually find an optimal solution to this problem. It also shows that, in many runs, SA gets trapped in local minima with costs much higher than the global minimum. For instance, 60% of the runs produce solutions with a cost at least 30% above the global minimum. This suggests that, if rather than completing all these unsuccessful runs, one could somehow predict when a run is likely to lead to a highly sub-optimal solution and abandon it, the efficiency of SA could be greatly enhanced. The following section further analyzes the behavior of typical SA runs and proposes a mechanism which, given a problem, aims at learning to recognize (un)promising SA runs.

3 Learning To Recognize (Un)promising SA Runs

Figure 3 depicts the behavior of a SA procedure on two different scheduling problems (from the set of benchmarks used in Section 6). For each problem, the figure depicts five SA runs, plotting the cost of the best solution, s , as the temperature of the procedure is progressively lowered – temperatures are shown in log scale, which is almost equivalent to computation time in linear scale. SA behaves

very differently on these two problems. For instance, in Problem #1, the range of final solutions is relatively narrow, while in Problem #2 it is much wider. Another differentiating factor is the behavior of the procedure at low temperatures. It seems that for Problem #1, the quality of a run can already be estimated quite accurately at $T = 50$ (e.g. the best run at $T = 50$ remains best at lower temperatures), while this is less so for Problem #2.

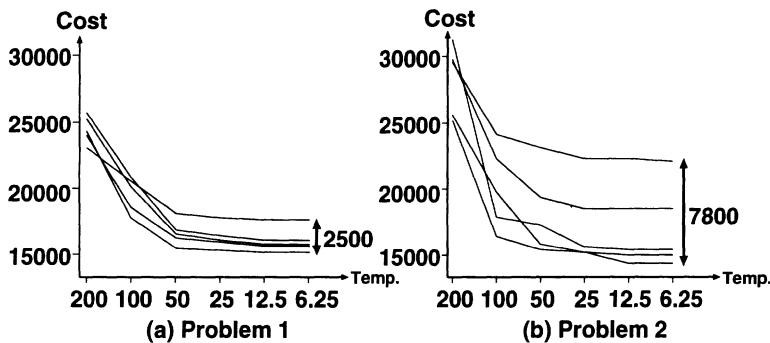


Fig. 3 Cost reductions in five SA runs on two different problems.

Clearly, such properties are not intrinsic to a problem itself. They could change if a different neighborhood structure or a different cooling profile was selected, as these parameters can affect the types of local optima encountered by the procedure and the chance that the procedure extricates itself from these local optima below a given temperature. While, in general, it may be impossible to find a SA procedure that reliably converges to near-optimal solutions on a wide class of problems, we can try to design adaptive SA procedures which, given a problem, can learn to recognize (un)promising runs and improve their performance over time. Below, we present a mechanism, which, given a problem, attempts to "learn" at different checkpoint temperatures the distribution of cost improvements that one can hope to achieve by continuing search below these temperatures.

Specifically, we postulate that, given a problem and a checkpoint temperature $T = t$, the distribution of the cost improvement that

is likely to be achieved by continuing a run below t can be approximated by a normal distribution. Using performance data gathered over earlier runs on a same problem, it is possible to approximate these *Expected Cost Improvement Distributions* (*ECID*) for a set C of checkpoint temperatures and use these distributions to identify (un)promising runs.

Formally, given a combinatorial optimization problem and a SA procedure for that problem, we define c_i^t as the cost of the best solution, s , at check point t in the i -th run and c_i^0 as the cost of the best solution obtained at temperature $T = T_1$ in the i -th execution. When the $(n + 1)$ -st run reaches a checkpoint temperature t , the *ECID* below t is approximated as a normal distribution $N[\mu_n^t, \sigma_n^t]$, whose average, μ_n^t , and standard deviation, σ_n^t , are given by:

$$\mu_n^t = \frac{\sum_{i=1}^n (c_i^t - c_i^0)}{n}, \quad \sigma_n^t = \sqrt{\frac{\sum_{i=1}^n \{(c_i^t - c_i^0) - \mu_n^t\}^2}{n-1}}$$

By incrementally refining these estimators over multiple runs, this mechanism can in essence "learn" to recognize (un)promising SA runs. The following sections successively describe a cutoff criterion and three restart criteria based on *ECID* distributions.

4 A Cutoff Criterion

Suppose that, in a sixth run on Problem #1, the best solution obtained at checkpoint $T = 100$ is solution **A** – Figure 4(a). At this checkpoint, the distribution of c_6^0 – the cost of the best solution that will have been found if the run is completed – can be approximated by the normal distribution $N[c_6^{100} - \mu_5^{100}, \sigma_5^{100}]$. This distribution, represented in Fig. 4(a), suggests that, if continued, the current run has a good chance of improving the current best solution, \mathbf{x} . Suppose that based on this analysis, the run continues until the next checkpoint, $T = 50$, and that the best solution found by the run when it reaches that temperature is **A'**. At this point, a new distribution of c_6^0 can be computed to check how the run is doing. This distribution, $N[c_6^{50} - \mu_5^{50}, \sigma_5^{50}]$ is shown in Figure 4(b). It appears much less promising than the one at $T = 100$. Now, the chances of improving the current best solution, \mathbf{x} , appear remote:

it probably does not make sense to continue this run.

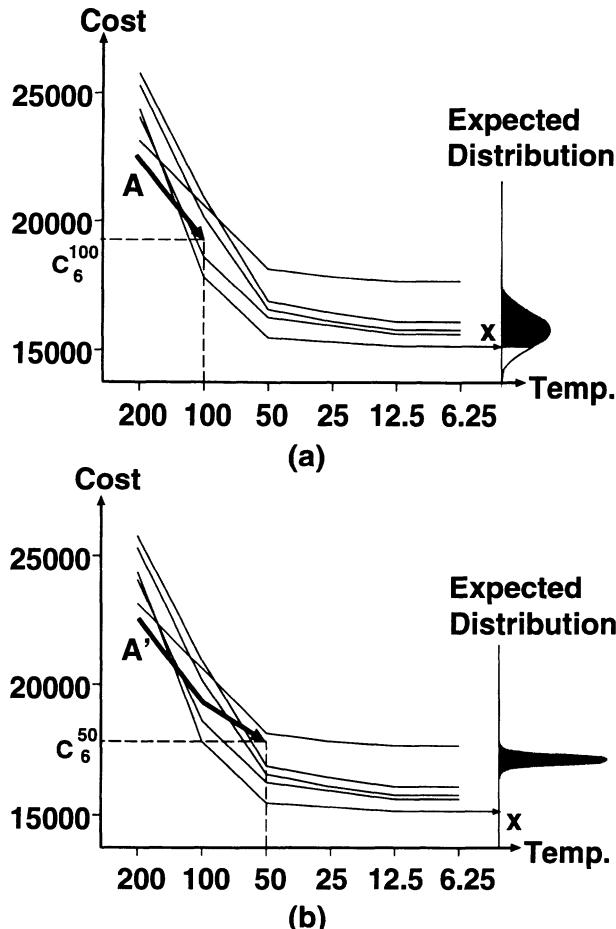


Fig. 4 Expected Cost Improvement Distributions at $T=100$ and $T=50$.

Formally, when the $(n + 1)$ -st run reaches a checkpoint temperature t , a cutoff criterion is used to determine whether or not to continue this run. In the study reported in Section 6, we use a cutoff criterion of the form:

$$\frac{(c_{n+1}^t - \mu_n^t) - x_n}{\sigma_n^t} > \text{threshold}$$

where x_n is the cost of the best solution found during the previous n runs and *threshold* is a threshold value. If the inequality holds,

the current run is abandoned. For example, if $threshold = 3$ (the value used in our experiments) and the cutoff inequality holds at a given checkpoint temperature t , the probability of improving x_n by continuing the run below t is expected to be less than 1% [1].

5 Three Restart Criteria

Whenever a run is completed or abandoned, two options are available: either start a fresh new annealing run *or*, instead, restart an earlier (promising) run, using a different sequence of random numbers ("reannealing"). In total, if reannealing is constrained to start from one of the checkpoint temperatures, there are up to $n \cdot |C| + 1$ possible options, where n is the number of earlier runs and $|C|$ the number of checkpoints in set C . Below, we describe three "restart criteria" that aim at selecting among these options so as to maximize the chances of quickly converging to a near-optimal solution.

5.1 Maximum Cost Reduction Rate Criterion

When considering several points from which to restart search, two factors need to be taken into account: (1) the likelihood that restarting search from a given point will lead to an improvement of the current best solution and (2) the time that it will take to complete a run from that point. Restarting from a low temperature will generally bring about moderate solution improvements, if any, while requiring little CPU time. Starting fresh new runs or restarting from higher temperatures can lead to more significant improvements, though less consistently and at the cost of more CPU time. In general, the cost improvements that can be expected from different temperatures will vary from one problem to another, as illustrated in Figure 3 (and as formalized by *ECID* distributions).

A natural restart criterion is one that picks the restart point expected to maximize the rate at which the cost of the current best solution will improve. For each restart candidate O_k (fresh annealing or reannealing), this can be approximated as the expected cost reduction (in the best solution), if search is restarted from O_k , divided by the expected CPU time required to complete a run from that restart point. Below, we use $R(O_k)$ to denote this approximation of the expected cost reduction rate, if search is restarted from

O_k :

$$R(O_k) = \frac{\text{expected-reduction}(O_k)}{\text{expected-CPU}(O_k)}$$

where $\text{expected-reduction}(O_k)$ is the expected cost reduction at the end of a run starting from O_k and $\text{expected-CPU}(O_k)$ is the CPU time that this run is expected to require. $\text{expected-CPU}(O_k)$ can be approximated as the average time required to complete earlier runs from O_k 's temperature. $\text{expected-reduction}(O_k)$ can be evaluated using $ECID$ distributions, as detailed below.

Given a reannealing point O_k at checkpoint temperature t and n earlier SA runs completed from t or above, $\text{expected-reduction}(O_k)$ can be approximated as:

$$\text{expected-reduction}(O_k) = \int_{LB}^{x_n} \{P_{nk}^t(x) \cdot (x_n - x)\} dx$$

where $P_{nk}^t(x)$ is the density function of the normal distribution $N[c_k - \mu_n^t, \sigma_n^t]$, c_k is the cost of O_k 's best solution¹, x_n is the cost of the best solution obtained over the first n runs, and LB is a lower-bound on the optimal solution²

Similarly, if O_k is a fresh SA run, $\text{expected-reduction}(O_k)$ can be approximated as:

$$\text{expected-reduction}(O_k) = \int_{LB}^{x_n} \{P_n(x) \cdot (x_n - x)\} dx$$

where $P_n(x)$ is the density function of the normal distribution $N[\mu_n^0, \sigma_n^0]$, with

$$\mu_n^0 = \frac{\sum_{i=1}^n c_i^0}{n}, \quad \sigma_n^0 = \sqrt{\frac{\sum_{i=1}^n (c_i^0 - \mu_n^0)^2}{n-1}}.$$

5.2 Randomized Criterion

One possible problem with the above criterion is its high sensitivity to possible inaccuracies in approximations of $ECID$ distributions. This can be a problem when the number of earlier runs is still small. When inaccurate $ECID$ distributions lead the criterion to choose a poor restart point, the procedure may take a long time before it improves the quality of the current best solution. In the

¹To be consistent, if O_k corresponds to the i -th SA run, $c_k = c_i^t$, as defined in Section 3.

²In the experiments reported in this paper, LB was simply set to 0.

meantime, it may keep on coming back to the same poor restart point. For this reason, it is tempting to use a randomized version of the criterion. One such variation involves randomly picking from a set of promising restart points, $H = \{O_l | R(O_l) > \beta \cdot \text{Max}\{R(O_k)\}\}$, while assuming that each element in H has the same probability, $1/|H|$, of being selected. β is a constant whose value is between 0 and 1.

5.3 Hybrid Criterion

A third alternative involves keeping some level of stochasticity in the restart criterion, while ensuring that more promising restart points have a higher chance of being selected. This is done by selecting restart points in H according to a Boltzmann distribution that assigns to each element $O_l \in H$ a probability

$$p(O_l) = \frac{\exp(R(O_l)/\tau)}{\sum_{O_k \in H} \exp(R(O_k)/\tau)}$$

Here, τ is a positive constant. If τ is very large, this method becomes equivalent to the randomized criterion described in subsection 5.2. If $\tau \approx 0$, this criterion becomes similar to the criterion of subsection 5.1. A similar distribution is used in the Q-learning algorithm described in [33].

6 Performance Evaluation

To evaluate performance of our cutoff and restart criteria, we consider a class of complex Job Shop Scheduling Problems first introduced in [23]. Similar, though more preliminary, results obtained for the Vehicle Routing Problem with Time Windows have also been reported in [26].

6.1 The Job Shop Scheduling Problem with Tardiness and Inventory Costs

The Just-In-Time Job Shop Scheduling Problem or Job Shop Scheduling Problem with Tardiness and Inventory Costs was first introduced in [23]. The problem assumes a factory, in which a set of jobs, $J = \{j_1, j_2, \dots, j_n\}$, has to be scheduled on a set of resources, $RES = \{R_1, R_2, \dots, R_m\}$. Each job requires performing a set of

operations $O^l = \{O_1^l, O_2^l, \dots, O_{n_l}^l\}$ and, ideally, should be completed by a given due date, dd_l , for delivery to a customer. Precedence constraints specify a complete order in which operations in each job have to be performed. By convention, it is assumed that operation O_i^l has to be completed before operation O_{i+1}^l can start ($i = 1, 2, \dots, n_l - 1$). Each operation O_i^l has a deterministic duration du_i^l and requires a resource $R_i^l \in RES$. Resources cannot be assigned to more than one operation at a time. The problem is to find a feasible schedule that minimizes the sum of tardiness and inventory costs of all the jobs ("Just-In-Time" objective). This problem is known to be NP-complete [23] and is representative of a large number of actual factory scheduling problems where the objective is to meet customer demand in a timely yet cost effective manner. Additional details on this model can be found in [23].

Experimental results reported below suggest that a good neighborhood function for this problem can be obtained by randomly applying one of the following three operators to the current schedule³:

- SHIFT-RIGHT: randomly select a "right-shiftable" operation and increase its start time by one time unit⁴.
- SHIFT-LEFT (mirror image of SHIFT-RIGHT): randomly select a "left-shiftable" operation and decrease its start time by one time unit.
- EXCHANGE: randomly select a pair of adjacent operations on a given resource and permute the order in which they are processed by that resource. Specifically, given two consecutive operations, A and B on a resource R , with A preceding B in the current solution, the exchange operator sets the new start time of B to the old start time of A and the new end time of A to the old end time of B ⁵.

³In the scheduling jargon, the Just-In-Time objective considered in this study is known to be irregular[15]. Prior applications of SA to job shop scheduling have only considered regular objectives such as Minimum Makespan. It can be shown that the neighborhoods used in these earlier studies are not adequate to deal with irregular objectives such as the one considered here [25].

⁴An operation is said to be "right(left)-shiftable" if its start time can be increased (decreased) by one time unit without overlapping with another operation.

⁵In our implementation, exchanging two operations is allowed even if a precedence constraint is violated in the process. Precedence constraint violations are

In our experiments, the probability of picking the EXCHANGE operator was empirically set to 3/7 while the probabilities of picking SHIFT-RIGHT or SHIFT-LEFT were each set to 2/7. Additionally, the values of parameters in the SA procedure (see Figure 1) were set as follows: $T_0 = 700$, $T_1 = 6.25$, $N = 200,000$ and $\alpha = 0.85$.

The performance of this SA procedure has been evaluated in a comparison against 39 combinations of well-regarded dispatch rules and release policies previously used to assess the performance of the Sched-Star [16] and Micro-Boss [23, 24] systems on a set of 40 benchmark problems similar to the ones described in [23]. The 40 benchmarks consisted of 8 problem sets obtained by adjusting three parameters to cover a wide range of scheduling condition: an average due date parameter (tight versus loose average due date), a due date range parameter (narrow versus wide range of due dates), and a parameter controlling the number of major bottlenecks (in this case one or two). For each parameter combination, a set of 5 scheduling problems was randomly generated, thereby resulting in a total of 40 problems. Each problem involved 20 jobs and 5 resources for a total of 100 operations. On average, when compared against the best solution found on each problem by the 39 combinations of dispatch rules and release policies, SA reduced schedule cost by 15% (average over 10 SA runs). When comparing the best solution obtained in 10 SA runs against the best solution obtained on each problem by the 39 combinations of dispatch rules and release policies, SA produced schedules that were 34% better. However, while running all 39 combinations of dispatch rules and release policies takes a few CPU seconds on a problem, a single SA run takes about 3 minutes on a DECstation 5000/200 running C. Additional details on these experiments can be found in [25].

6.1.1 Empirical Evaluation of Cutoff and Restart Criteria

We now turn to the evaluation of the cutoff and restart criteria presented in this paper and compare the performance of five variations of the SA procedure presented in 6.1:

- **N-SA:** regular SA, as described in 6.1 (no learning).
- **P-SA:** SA with cutoff criterion.

handled using large artificial costs that force the SA procedure to quickly get rid of them [25].

- **B-SA:** SA with cutoff and Maximum Cost Reduction Rate restart criteria.
- **R-SA:** SA with cutoff and randomized restart criteria ($\beta = 0.5$).
- **H-SA:** SA with cutoff and hybrid restart criteria ($\beta = 0.5$ and $\tau = 1$).

When running P-SA, B-SA, R-SA, and H-SA, the cutoff and/or restart criteria were only activated after 5 complete SA runs to allow for the construction of meaningful *ECID* distributions. All four of these procedures used the same set of checkpoints, $C = \{200, 100, 50, 25, 12.5\}$.

The five procedures were compared on the same 40 benchmark problems described in subsection 6.1⁶. Each SA procedure was run for 2 hours on each benchmark problem. Furthermore, to eliminate possible noise effects, each two-hour experiment was repeated a total of 15 times. The results presented here were obtained by averaging performance of these 15 runs of each procedure on each problem.

Fig. 5 depicts the performance of the five SA procedures on a typical benchmark problem. The first 15 minutes are not represented, as they correspond to the first 5 runs when the cutoff and restart criteria have not yet been activated.

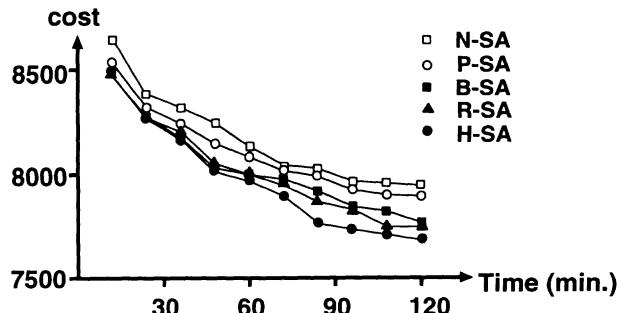


Fig. 5 Improvement of the best solution over time.

The figure shows that throughout its run, N-SA was dominated by the other four procedures. It also indicates that both the cutoff

⁶At the present time, only a subset of the problems in each of the 8 problem sets have been completed. Complete results will be presented in the final version of the paper.

criterion and the restart criteria contributed to this performance improvement. Among the three restart criteria, H-SA appears to perform best. Figure 5 further suggests that the restart criterion in H-SA improves performance through the entire run, as the gap between H-SA and N-SA widens over time. These observations are confirmed by results obtained on the 8 problem sets of the study, as depicted in Figure 6. Fig. 6(a) shows the average cost reductions yielded by P-SA, B-SA, R-SA and H-SA over N-SA at the end of the two-hour runs. Figure 6(b) gives the average reduction in the CPU time required by each of these four procedures to find a solution of equal or better quality than the best solution found by N-SA in two hours. It can be seen that H-SA requires between 30% and 70% less CPU time than N-SA.

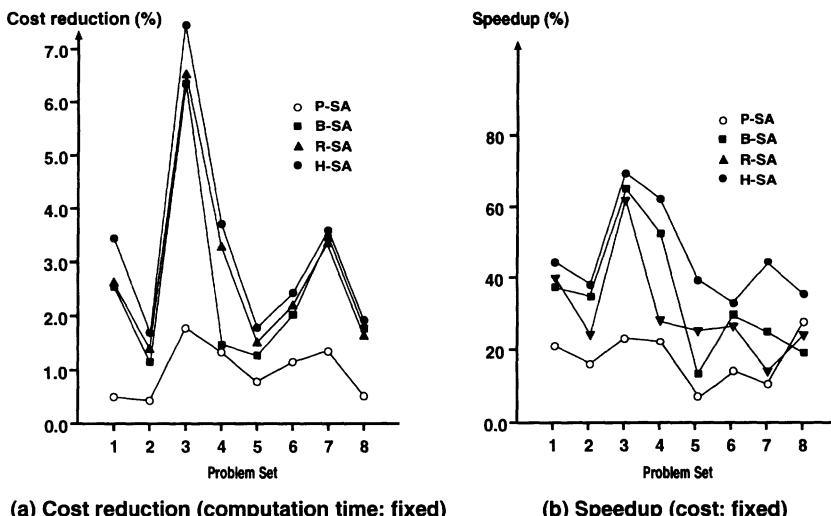


Fig. 6 Empirical comparison.

A finer analysis indicates that performance improvements produced by our cutoff and restart criteria increase as one requires higher quality solutions. Figure 7, compares the average CPU time of each of the five procedures as the required quality of solutions is increased. While all five procedures take about as long to find a solution with cost below 9000 or 8800, the time required to find a solution below 8500 varies significantly (e.g. H-SA can find such a solution in 3500 seconds while N-SA requires close to 10,000 sec-

onds).

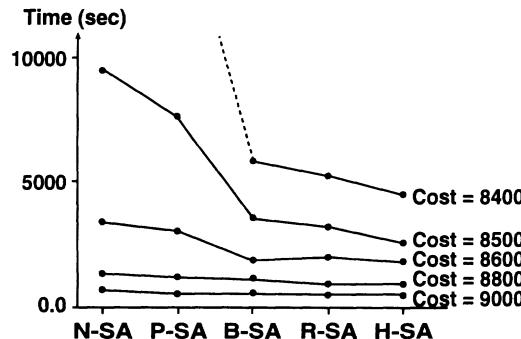


Fig. 7 Speedups as a function of required solution quality.

As already indicated in Section 5, the difference in performance between B-SA, R-SA and H-SA suggests that a deterministic use of *ECID* distributions to decide where to restart search can be tricky, as these distributions may not be accurate, especially when only a small number of runs has been completed. By injecting non-determinism in the restart criterion, R-SA and H-SA ensure that the procedure will not always restart from the same point. The procedure is forced to sample a wider area and in the process gets a chance to refine *ECID* distributions. From this point of view, B-SA is a procedure that places more emphasis on using existing knowledge of the search space than acquiring new one, while R-SA places more emphasis on learning and less on exploiting already acquired information. H-SA appears to provide the best balance between these two requirements.

Finally, it should be obvious that the CPU time and memory overheads of our cutoff and restart criteria are very moderate. All in all, in our experiments, the CPU time required to learn *ECID* distributions and apply the cutoff and restart criteria was well under 1% of total CPU time.

7 Summary

In summary, we have developed a mechanism that learns to recognize (un)promising SA runs by refining "Expected Cost Improvement Distributions" (*ECIDs*) over multiple SA runs, and have developed search cutoff and restart criteria that exploit these distributions. These mechanisms are domain independent and have been

validated on two domains: the just-in-time job shop scheduling domain with tardiness and inventory costs and the vehicle routing problem with time windows. Experimental results show that in both domains these mechanisms can dramatically reduce the computational requirements of competitive SA procedures. Experiments presented in this paper further indicate that the closer one seeks to be to the optimum, the larger the speedups.

8 Acknowledgements

This research was supported, in part, by the Advanced Research Projects Agency under contract F30602-91-C-0016 and, in part, by an industrial grant from NEC. The second author would like to thank Masahiro Yamamoto, Takeshi Yoshimura and Yoshiyuki Koseki of NEC Corporation for giving him the opportunity to spend a year at Carnegie Mellon University. The authors would also like to thank Sue Sun for her help with the VRPTW implementation.

References

- [1] Beyer, W.H. "CRC Standard Mathematical Tables, 28th Edition," CRC Press, Inc., Boca Raton, Florida, 1987.
- [2] Bodin, L., B. L. Golden, A. A. Assad and M. Ball "The State of the Art in the Routing and Scheduling of Vehicles and Crews" *Computers and Operations Research* Vol. 10 no. 2, 1983.
- [3] Cerny, V., "Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm," *J. Opt. Theory Appl.*, Vol. 45, pp. 41–51, 1985.
- [4] Chiang, W. C. and R. Russell "Simulated Annealing Metaheuristics for the Vehicle Routing Problem with Time Windows" Working Paper, Department of Quantitative Methods University of Tulsa, Tulsa, OK 74104. 1993.
- [5] Desrochers, M., J. Desrosiers and M. M. Solomon "A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows" newblock *Operations Research* Vol. 40 no. 2, 1992.
- [6] Garey, M. R. and Johnson, D. S. "Computers and Intractability: A Guide to the Theory of NP-Completeness," Freeman and Co., 1979
- [7] Glover, F. and Laguna M. "Tabu Search," Chapter in *Modern Heuristic Techniques for Combinatorial Problems*, June 1992.
- [8] Golden B. L. and A. A. Assad (eds.) "Vehicle Routing: Methods and Studies," North Holland, Amsterdam 1988.
- [9] Graves, S. C. "A Review of Production Scheduling," *Operations Research* Vol. 29 no. 4, pp. 646–675, 1981.
- [10] Johnson, D. S., Aragon, C. R., McGeoch, L. A. and Schevon, C. "Optimization by Simulated Annealing: Experimental Evaluation; Part I, Graph Partitioning" *Operations Research* Vol. 37 no. 6, pp. 865–892, 1989.

- [11] Johnson, D. S., Aragon, C. R., McGeoch, L. A. and Schevon, C. "Optimization by Simulated Annealing: Experimental Evaluation; Part II, Graph Coloring and Number Partitioning" *Operations Research* Vol. 39 no. 3, pp. 378–406, 1991.
- [12] Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P., "Optimization by Simulated Annealing," *Science*, Vol. 220, pp. 671–680, 1983.
- [13] Laporte, G. "The Vehicle Routing Problem: An Overview of Exact and Approximate Algorithms," *European Journal of Operational Research*, Vol. 59, pp. 345–358, 1992.
- [14] Matsuo H., C.J. Suh, and R.S. Sullivan "A Controlled Search Simulated Annealing Method for the General Jobshop Scheduling Problem" Technical Report, "Department of Management, The University of Texas at Austin Austin, TX, Working Paper, 1988. 1992.
- [15] Morton, T.E. and Pentico, D.W. "Heuristic Scheduling Systems," *Wiley Series in Engineering and Technology Management*, 1993
- [16] Morton, T.E. "SCED-STAR: A Price-Based Shop Scheduling Module," *Journal of Manufacturing and Operations Management*, pp. 131–181, 1988.
- [17] Nakakuki,Y. and Sadeh,N. "Increasing the Efficiency of Simulated Annealing Search by Learning to Recognize (Un)Promising Runs," *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pp. 1316–1322, 1994.
- [18] Osman,I.H., and Potts, C.N., "Simulated Annealing for Permutation Flow-Shop Scheduling," *OMEGA Int. J. of Mgmt Sci.*, Vol. 17, pp. 551–557, 1989.
- [19] Osman, I.H. "Meta-Strategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem," *Annals of Operations Research*, Vol. 41, pp. 421–451, 1993.
- [20] Palay, A. "Searching With Probabilities" PhD thesis, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, July 1983.

- [21] Potvin, J. Y. and S. Bengio. "A Genetic Approach to the Vehicle Routing Problem with Time Windows" Technical Report, CRT-953, Centre de Recherche sur les Transports, Universite de Montreal, Canada, 1993.
- [22] Potvin, J. Y. and J. M. Rousseau "A Parallel Route Building Algorithm for the Vehicle Routing and Scheduling Problem with Time Windows" *European Journal of Operational Research*, Vol. 66, pp. 331-340, 1993.
- [23] Sadeh, N. "Look-ahead Techniques for Micro-opportunistic Job Shop Scheduling" PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, March 1991.
- [24] Sadeh, N. "Micro-Opportunistic Scheduling: The Micro-Boss Factory scheduler" Chapter in *Intelligent Scheduling*, Zweben and Fox (eds), Morgan Kaufmann Publishers, 1994.
- [25] Sadeh, N. and Nakakuki Y. "Focused Simulated Annealing Search: An Application to Job Shop Scheduling" To appear in *Annals of Operations Research*, Special issue on Metaheuristics in Combinatorial Optimization, Also available as technical report CMU-RI-TR-94-30 at the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, 1994.
- [26] Sadeh, N. Nakakuki Y. and Thangiah S.R. "Learning to Recognize (Un)Promising Simulated Annealing Runs: Applications to Job Shop Scheduling and Vehicle Routing" Working paper. The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, 1995.
- [27] Savelsbergh M. W. P. "Local Search for Routing Problems with Time Windows," *Annals of Operations Research*, Vol. 4, pp. 285–305, 1985.
- [28] Solomon, M. M. "Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints." *Operations Research*, Vol. 35, no. 2 pp. 254–265, 1987.
- [29] Thangiah, S. R. "Vehicle Routing with Time Windows using Genetic Algorithms." Chapter in *Application Handbook of Genetic Algorithms: New Frontiers*, CRC Press, 1994.

- [30] Thangiah, S. R. , I. H. Osman, T. Sun "Metaheuristics for Vehicle Routing Problems with Time Windows." Technical Report *Artificial Intelligence and Robotics Laboratory*, Computer Science Department, Slippery Rock University, Slippery Rock, PA 16057, 1994.
- [31] Van Laarhoven, P.J., Aarts, E.H.L., and J.K. Lenstra "Job Shop Scheduling by Simulated Annealing," *Operations Research*, Vol. 40, No. 1, pp. 113–125, 1992.
- [32] Vepsäläinen,A.P.J. and Morton, T.E. "Priority Rules for Job Shops with Weighted Tardiness Costs," *Management Science*, Vol. 33, No. 8, pp. 1035–1047, 1987.
- [33] Watkins, C. J. C. M., "Learning with Delayed Rewards" PhD thesis, Cambridge University, Psychology Department, 1989.
- [34] Wefald, E.H. and Russel, S.J. "Adaptive Learning of Decision-Theoretic Search Control Knowledge" *Sixth International Workshop on Machine Learning*, Ithaca, NY, 1989.
- [35] Zweben, M., Davis E., Daun B., Deale M., "Rescheduling with Iterative Repair" Technical Report FIA-92-15, NASA Ames Research Center, Artificial Intelligence Research Branch, Moffett Field, CA 94025 April, 1992.

A Preliminary Investigation into the Performance of Heuristic Search Methods Applied to Compound Combinatorial Problems

Mike B. Wright and Richard C. Marett

Department of Management Science

The Management School

University of Lancaster

Lancaster LA1 4YX

UK

E-mail: m.wright@lancaster.ac.uk

Abstract:

Many real combinatorial problems involve several objectives. A common procedure for such problems is to combine these separate objectives into a single overall objective, by means of weightings. Such a problem is therefore *compound* in nature, since its objective is a single entity made up of several individual elements, as with chemical compounds.

However, it is reasonable to suppose that compound problems may exhibit different features from truly single-objective problems, because the natures of their solution spaces (and, in particular, the ‘shapes’ of their local optima) are likely to be rather different. It is hypothesized that this fact may significantly affect the relative effectiveness of heuristic search techniques used to solve these prob-

lems.

This paper presents the results of a preliminary investigation into three sets of compound problems (31 problems in total), using three well-known heuristic search techniques. A measure of local optimum shape is proposed and used. The results show that the above hypothesis appears to be true to at least some extent.

Key Words: Compound combinatorial problems, measures of complexity, multiple objectives, simulated annealing, tabu search.

1. Introduction

Several authors have carried out experiments to compare the performance of different heuristic search approaches, notably Tabu Search (TS) and Simulated Annealing (SA). For example, de Amorim, Barthélemy and Ribeiro (1992) discovered that both methods perform well for the clique partitioning problem and Hansen and Jaumard (1990) concluded that TS was better than SA for the maximum satisfiability problem. Hertz and de Werra (1987) showed that TS was significantly better than SA for the graph colouring problem, Reeves (1993) used TS to improve upon one of the best known implementations of SA for the flow-shop problem and Sinclair (1993) found that TS outperforms SA for the hydraulic turbine runner balancing problem.

However, these comparisons have nearly always been for *single-objective* problems. In contrast, many important real problems have many objectives. Even when these objectives are combined by means of weightings to form a single *compound* objective, these problems are likely to be rather different from the truly single-objective problems, because their solution spaces may be very different in nature. It is reasonable to suppose, therefore, that compound

problems may exhibit very different features concerning the progress of heuristic search methods, and that any conclusions reached from single-objective studies may not apply to compound ones. This may be especially true when the objectives to be combined are of various different types.

Marett and Wright (1996) compared the performance of classical forms of Repeated Descent (RD), Simulated Annealing (SA) and Tabu Search (TS) for a series of flow-shop problems, with the number of objectives ranging from 3 to 12. It was tentatively concluded that TS tends to outperform SA when the number of objectives is low and the time available high, but that SA tends to outperform TS when the number of objectives is high and the time available low. Both TS and SA consistently outperform RD. Other conclusions were also reached concerning the best forms of implementation for classical forms of RD, SA and TS.

This paper extends the previous work in two ways. Other series of problems are used—two sets of Travelling Salesman Problems (TSPs) and a set of Layout Problems (LOPs). In addition, the comparisons of the performance of RD, TS and SA are related not only to the number of objectives (which is a fairly crude measure of complexity) but also to the ‘shape’ of local optima.

The reason for concentrating on local optima rather than the overall solution space is that local optima play an important role for all three methods. RD repeatedly aims for a local optimum. TS moves from one local optimum to the next, never straying very far away. SA, though not linked to local optima in such a well-defined way, is still likely to spend much of its time very close to local optima.

2. The Problems Used

Two series of TSPs have been used for this study. One involves 30 cities, with a random asymmetric cost matrix. The other involves 100 cities, with a symmetric cost matrix derived from Euclidean distances.

The number of objectives for the 30-city TSPs ranges from 1 (*i.e.* a standard TSP) to 8. For the 100-city TSPs the number ranges from 1 to 14. Each objective other than the first consists of the cost of a tour for a subset of cities. Thus some pairs of objectives will tend to reinforce one another, while others will tend to be in conflict. Fuller details are in Appendix A.

One series of LOPs has been used. The number of objectives ranges from 1 to 6, though there are 9 problems overall. There is clear conflict between some pairs of objectives. Fuller details are in Appendix B. The total number of problems studied is thus 8+14+9, *i.e.* 31.

The specific applicability of the problems themselves are not important for this study. They are merely vehicles through which better understanding of compound problems and their solution by heuristic search methods may be achieved.

3. The Techniques Compared

The three basic approaches compared in this study are Repeated Descent (RD), SA and TS (see Papadimitriou and Steiglitz (1982, ch. 19), Eglese (1990) and Glover (1990) respectively for further details). It should be stressed, however, that only the ‘classical’ forms of SA and TS have been used for this study. For example, monotonic cooling has been used throughout for SA and only a very simple form of diversification has been used for TS. Many more radical

variations of each approach have been proposed and used, particularly for TS where it is now generally accepted that some form of intelligent diversification is highly desirable for complex problems. This is discussed further later in this paper.

However, within the classical frameworks, each one of these approaches can be implemented in a wide variety of ways. It was therefore necessary to try to ensure that the best possible variations of each approach were compared. Full details of how this was achieved are given by Maret and Wright (1996).

It is also necessary, for a valid comparison, to ensure that each approach is given an equal amount of time to achieve a good result. The results given here compare the best results achieved by each method after a number of perturbations equal to a hundred times the neighbourhood size. (Some results for numbers of perturbations less than this have been reported by Maret and Wright (1996), where it was tentatively concluded that SA was generally the best out of the three approaches at achieving a reasonably good solution quickly.)

Although these may not be the best possible techniques to use for these problems, they are representative of three different approaches. Classical TS is a highly intensifying procedure, whereas RD and SA both employ diversification but in very different ways.

In addition, several longer runs were made of SA and TS for every problem in order to try and achieve an improved upper bound for the optimal solution of each problem. While it is not known how good these upper bounds are, they are useful when trying to assess the results of each method in absolute as well as relative terms. (Further work is being undertaken to compare the behaviour of

each method over long time-scales.)

4. Results

Graphs 1 to 3 below show the results for each set of problems. Every symbol represents the average cost achieved by 100 separate runs for a single problem. The total number of runs directly contributing to these runs is thus $100 \times 3 \times 31 = 9,300$. Moreover, as stated above, each method was in some sense optimized separately for each problem. In all, the number of individual computer runs contributing directly or indirectly to these results was of the order of 250,000.

The y -axis gives p , the percentage by which the average cost exceeds the best known solution for that particular problem. Care should be exercised in inferring any trends in p , since of course the best known solution for any problem may still be some way from the optimum.

The x -axis gives m , the number of objectives that are combined to make the compound objective. Different values of m therefore refer to different problems. There are in fact three LOPs with $m = 2$ and two with $m = 3$. These are artificially separated on the graphs below for ease of inspection.

The dotted lines between the symbols thus have no formal meaning, but are merely aids to the interpretation of the graphs.

The differences between the three methods, in terms of the values of p , are all statistically significant at a 5% significance level (and often at a much lower level), except for the difference between SA and TS for the 2-objective 30-city TSP.

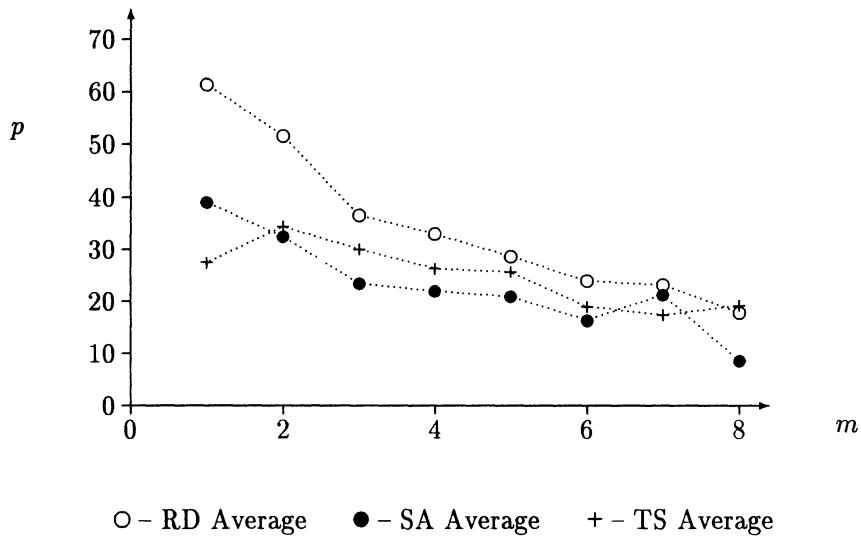


Figure 1: Performance Relative to Upper Bound Against Number of Objectives (30-city TSP)

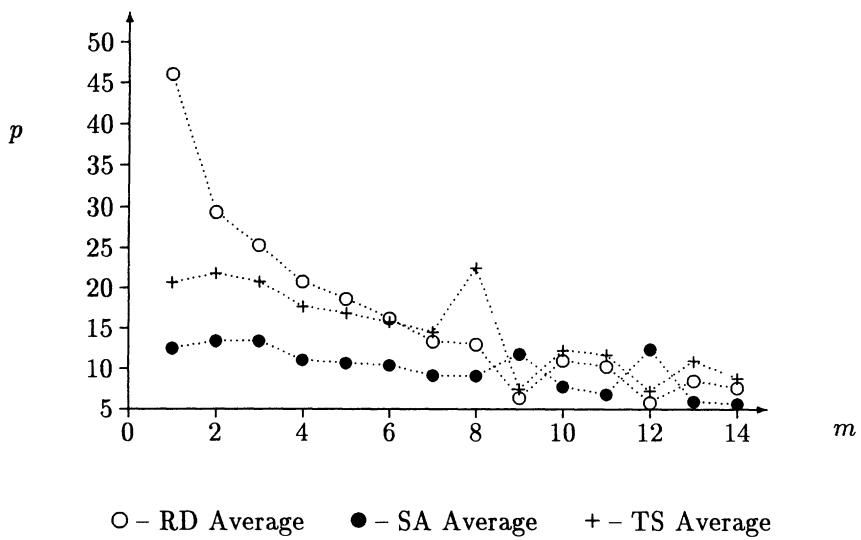


Figure 2: Performance Relative to Upper Bound Against Number of Objectives (100-city TSP)

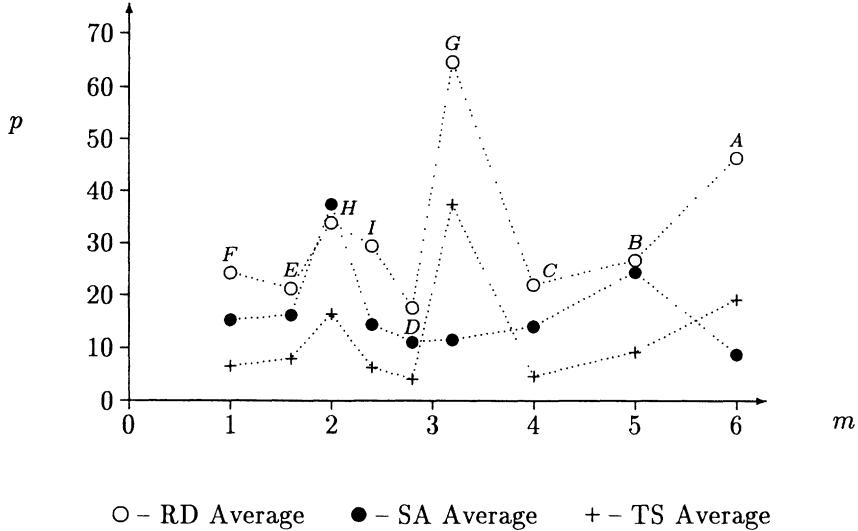


Figure 3: Performance Relative to Upper Bound Against Number of Objectives (LOP)

5. Initial Conclusions

The average results are all very much higher than the upper bounds. This highlights the extra difficulty caused by a multiplicity of objectives and emphasises the value of a sophisticated approach to such problems, *e.g.* by using some form of intelligent diversification.

On the whole, SA performs better than TS for the TSPs. One exception is not unexpected—TS is better for the single-objective 30-city TSP. However, there are some other exceptions which are not so easy to understand. These are discussed later.

By contrast, TS tends to perform better than SA for the LOPs, though again there are some anomalous-looking exceptions which are investigated further below.

As expected, RD is for the most part not as good as either SA or TS. However, it is interesting to note that RD does become

marginally better than TS for the 100-city TSPs once the number of objectives exceeds 6.

6. Investigation of the Local Optima

The results show that the number of objectives may be too crude a measure to use to give a useful description of these compound problems. This is not really surprising. Some objectives may reinforce one another, whereas others may conflict, and others may be wholly uncorrelated. Some objectives may make a large (possibly overwhelming) contribution to the overall cost, others a small (and possibly wholly uninfluential) contribution, with yet others in between. Some objective costs may change almost continuously for small solution perturbations, whereas others may change in occasional large steps.

Therefore a problem measure is needed which relates to the solution space itself. Maret and Wright (1996) used the average number of perturbations needed to move from a random initial solution to a local optimum by means of steepest descent for various flow-shop problems, with some success. However, this approach did not prove fruitful for the TSPs or LOPs. Therefore a rather more focussed measure was defined which relates specifically to the nature of local optima, since for all methods much of the time and effort is spent in their vicinity.

1,000 steepest descents were run for each of the 28 compound problems. On the final move to a local optimum from the previous solution, some *subcosts* clearly decrease while others (probably) increase. So, for the i^{th} steepest descent procedure, we can define δ_i^+ , the sum of the subcost increases, and δ_i^- , the sum of the subcost decreases for this final move. Let r be the sample correlation coefficient between δ_i^+ and δ_i^- .

If $r \approx 1$ then there is high linear association between δ_i^- and δ_i^+ , i.e. the value of δ^+ for a given value of δ^- is very predictable and the characteristics of neighbours of local minima are very similar. Also, since r is positive, increasing δ_i^- generally increases δ_i^+ , i.e. trying to increase the total amount by which the subcosts decrease generally results in a higher total amount by which the subcosts *increase*—there is some ‘conflict’ between the subcosts.

However, if $r \approx 0$ then there is very little linear association between δ_i^- and δ_i^+ , i.e. the value of δ^+ for a given value of δ^- is very *unpredictable* and the characteristics of neighbours of local minima are very *dissimilar*.

If $r \approx -1$ then there is again high linear association between δ_i^- and δ_i^+ . However, increasing δ_i^- generally *decreases* δ_i^+ , i.e. trying to increase the total amount by which the subcosts decrease generally results in a *lower* total amount by which the subcosts increase—the subcosts ‘help’ each other. We would expect this case to occur rarely, if at all.

Thus we would expect $|r|$ to measure the amount of ‘predictability’ between the subcosts near local minima, with $r \approx 0$ representing a small amount of predictability. Note, however, that r is a measure of *linear* association between δ_i^- and δ_i^+ —it is possible for there to be a very close non-linear relationship and yet $r \approx 0$.

Graphs 4 to 6 below show how p varies with r for the three sets of problems.

For the 30-city TSPs, r ranges from almost zero to about 0.26, but there is no clear pattern.

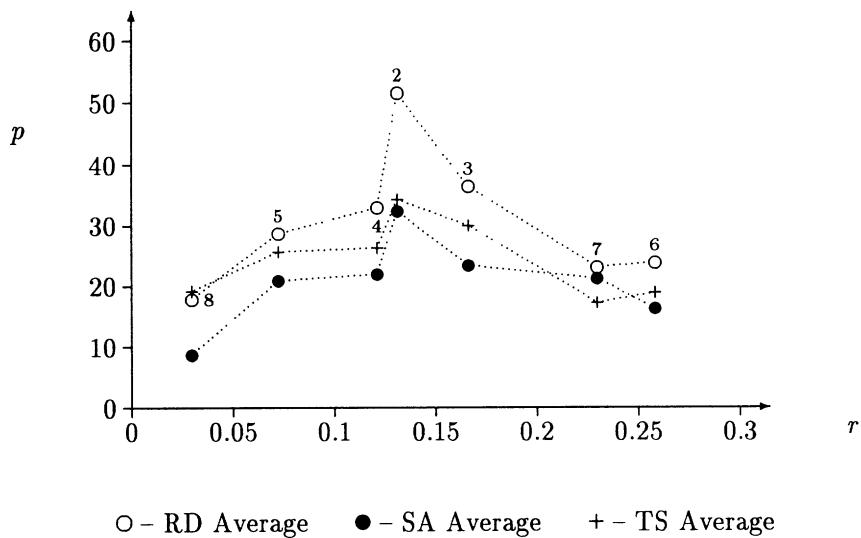


Figure 4: Values of p Against r for Problems 2 to 8 (30-city TSP)

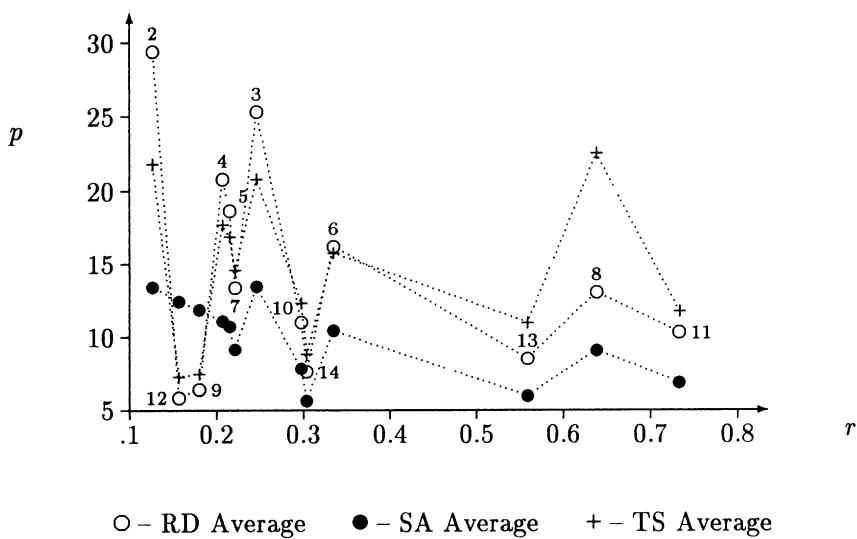


Figure 5: Values of p Against r for Problems 2 to 14 (100-city TSP)

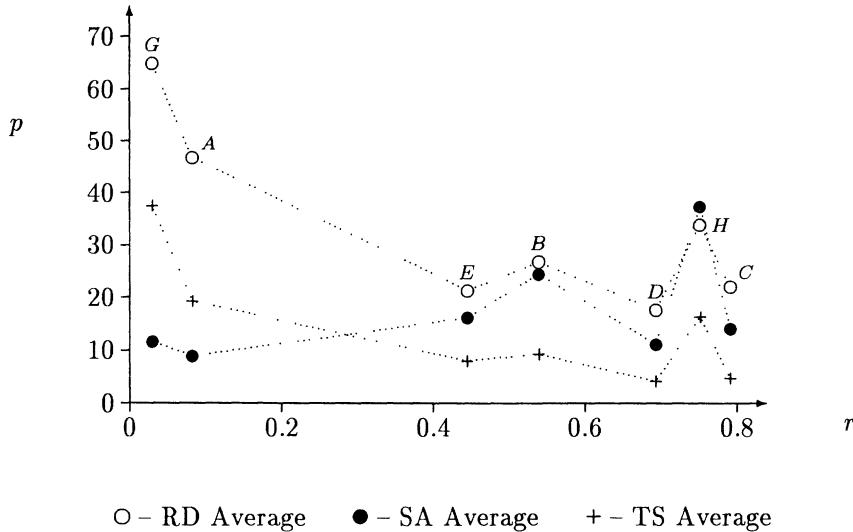


Figure 6: Values of p Against r for Problems A to E and G to H (LOP)

For the 100-city TSPs, the results look a little more interesting. There is a very wide range of values of r , from about 0.12 to about 0.74. The three problems with a particularly high value of r are all best solved by SA, with RD second and TS third. The two anomalous results from the earlier analysis (the problems with 9 and 12 objectives), which were best solved by RD, with TS second and SA third, both have low values of r . However, the picture is by no means clear.

For the LOPs, by contrast, the results look surprisingly clear-cut. The two anomalous problems (Problems A and G) have far smaller values of r than the other problems. It appears that SA is better than TS for low values of r (not far above zero) but that TS is superior for larger values of r . However, this conclusion is diametrically opposed to that for the 100-city TSPs. (Note that r could not be defined for Problem I because δ_i^+ was the same for every single one of the local optima.)

Though the picture is by no means clear, it would seem reasonable to infer the following tentative conclusions from these results:

1. The nature of the solution space in the vicinity of local optima may be very different for different versions of a problem.
2. This difference may affect the choice of best heuristic search technique.

In order to verify and strengthen these conclusions, further research is clearly needed along the following lines:

1. Other compound problems, of various types, should be analysed.
2. Other measures of the nature of the solution space need to be proposed and tested.
3. Variations of SA and TS, and possibly other meta-heuristic techniques, need to be used. Any conclusions that apply for simple versions may not do so for more complex ones, *e.g.* methods using some form of intelligent diversification, such as proposed by Hübscher and Glover (1994) and Wright (1994b) and Wright (1994a).

If this paper's preliminary tentative findings are confirmed, they are clearly of great potential importance.

References

- de Amorim, S. G., Barthélemy, J.-P. and Ribeiro, C. C. (1992). Clustering and clique partitioning: Simulated annealing and tabu search approaches, *Journal of Classification* **9**, 17–41.
- Eglese, R. W. (1990). Simulated annealing: A tool for operational research, *European Journal of Operational Research* **46**, 271–281.
- Glover, F. (1990). Tabu search: A tutorial, *Interfaces* **20**(4), 74–94.
- Hansen, P. and Jaumard, B. (1990). Algorithms for the maximum satisfiability problem, *Computing* **44**, 279–303.
- Hertz, A. and de Werra, D. (1987). Using tabu search techniques for graph colouring, *Computing* **39**, 345–351.
- Hübscher, R. and Glover, F. (1994). Applying tabu search with influential diversification to multi-processor scheduling, *Computers and Operations Research* **21**(8): 877–884.
- Marett, R. C. and Wright, M. B. (1996). A comparison of neighbourhood search techniques for multi-objective combinatorial problems, *Computers and Operations Research*. To be published in early 1996.
- Papadimitriou, C. H. and Steiglitz, K. (1982). *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, New Jersey, U.S.A.
- Reeves, C. R. (1993). Improving the efficiency of tabu search for machine sequencing problems, *Journal of the Operational Research Society* **44**(4): 375–382.

- Sinclair, M. (1993). Comparison of the performance of modern heuristics for combinatorial optimization on real data, *Computers and Operations Research* **20**(7): 687–695.
- Wright, M. B. (1994a). School timetabling using heuristic search, *Journal of the Operational Research Society*. Submitted Nov.
- Wright, M. B. (1994b). Timetabling county cricket fixtures using a form of tabu search, *Journal of the Operational Research Society* **45**(7), 758–770.

Appendix A: The Travelling Salesman Problems

The TSPs considered here are based on the classic single-objective TSP: a salesman starts out from a given city and makes a tour of $n - 1$ other cities, eventually returning to the city he started from. We have used two sets of TSPs, TSP-30 and TSP-100, with thirty and a hundred cities respectively.

For $1 \leq i \leq n - 1$ let m be the greatest integer such that $mi \leq n - 1$, and define subcost i as follows:

Subcost i : the distance from the starting city to the i^{th} of the $n - 1$ other cities, plus the distance from the i^{th} city to the $2i^{\text{th}}$ city, plus the distance from the $2i^{\text{th}}$ city to the $3i^{\text{th}}$ city ... plus the distance from the $(m - 1)i^{\text{th}}$ city to the mi^{th} city, plus the distance from the mi^{th} city to the starting city.

We thus have $n - 1$ possible subcosts in all. Subcost i for $1 \leq i \leq n - 1$ is most easily visualized as the distance travelled by the salesman in making a ‘skipped tour’: $i - 1$ cities are ‘skipped over’ before going to the next city. The subcosts will ‘conflict’ with each other because trying to minimize one subcost may well leave large

distances between the ‘skipped’ cities; these distances will be part of some of the other subcosts.

Random values were generated for the distances for TSP-30. TSP-30 is *asymmetric*, i.e. the distance from the i^{th} to the j^{th} city is not necessarily equal to the distance from the j^{th} to the i^{th} city. However, for TSP-100 the cities were placed randomly in a square. The distance between two cities is the Euclidean distance. Thus TSP-100 is *symmetric*.

For TSP-30 eight separate problems were created by using different combinations of the subcosts:

Problem 1: Subcost 1 only.

Problem 2: Subcosts 1 and 3.

...

Problem 8: Subcosts 1, 3, 5 ... 15.

For TSP-100 fourteen separate problems were created by using different combinations of the subcosts:

Problem 1: Subcost 1 only.

Problem 2: Subcosts 1 and 5.

...

Problem 14: Subcosts 1, 5, 9 ... 53.

The *swap* neighbourhood structure was used for the TSPs: the neighbours of a permutation of cities are those permutations obtained by swapping two cities.

Appendix B: The Layout Problems

The LOPs considered here are loosely based on the classic *quadratic assignment problem* (QAP). We have sixty-four departments of a building which are to be placed, or *laid out*, in an xy -plane. The departments are rectangular, with their top and bottom edges parallel to the x -axis. Each department may be translated by any distance along the x - or y -axis, or rotated by 90° about its centre. A department may overlap another department by any amount.

Each department has a *flow of information* with every other department—this might represent, for example, the average number of people who walk from the department to the other department each day. The flows of information are *symmetric*: for departments i and j the flow of information between departments i and j is equal to that between departments j and i .

For departments i and j the *flow \times distance* between departments i and j is defined to be the flow of information between departments i and j multiplied by the distance apart of the two departments. The distance is measured from the *centres* of the two departments using the *Manhattan metric*: the distance between (x_1, y_1) and (x_2, y_2) is $|x_1 - x_2| + |y_1 - y_2|$.

If departments i and j do not overlap, their *overlap* is defined to be zero. If they *do* overlap, they will overlap in a rectangle of width w and height h , say. Their overlap is then defined to be $(wh)^3$.

For any layout of the sixty-four departments, define the *bounding*

rectangle of the layout to be the smallest rectangle which contains all the departments.

Sixteen of the sixty-four departments are designated *tour* departments and put in a sequence so as to form a set tour. We would like to minimize the total distance travelled in moving from the first tour department to the second, and then from the second tour department to the third *etc.*, eventually returning to the first tour department. Again, the distances are measured from the centres of the departments using the Manhattan metric. This tour might represent, for example, departments that have to be connected in a given order by an expensive computer cable, the cable then returning to the starting department.

We would also like a square *courtyard*, of a given width, to be in the centre of the layout.

Define the following subcosts:

QAP Cost: The sum of all the flow \times distances between every pair of departments plus the sum of all the overlaps between every pair of departments. The sum of the overlaps is multiplied by the maximum flow of information attained between two departments.

x -Distance Cost: The width of the bounding rectangle.

y -Distance Cost: The height of the bounding rectangle.

x/y -Distance Disparity Cost: The absolute difference between the width and the height of the bounding rectangle.

Tour Cost: The total length of the tour of the sixteen tour departments.

Courtyard Cost: The sum of the overlaps between every department and the courtyard. The courtyard is centred on the centre of the bounding rectangle.

We thus have six subcosts in all. The following problems were created by using different combinations of the subcosts:

Problem A: QAP Cost, x -Distance Cost, y -Distance Cost, x/y -Distance Disparity Cost, Tour Cost and Courtyard Cost.

Problem B: QAP Cost, x -Distance Cost, y -Distance Cost, x/y -Distance Disparity Cost and Tour Cost.

Problem C: QAP Cost, x -Distance Cost, y -Distance Cost and x/y -Distance Disparity Cost.

Problem D: QAP Cost, x -Distance Cost and y -Distance Cost.

Problem E: QAP Cost and x -Distance Cost.

Problem F: QAP Cost only.

Problem G: QAP Cost, Tour Cost and Courtyard Cost.

Problem H: QAP Cost and Tour Cost.

Problem I: QAP Cost and Courtyard Cost.

The neighbourhood structure used for the LOPs consists of a mixture of translations and rotations of the departments. One of the departments is translated by between one and seven units along the x - or y -axis (in either direction) or one of the departments is rotated by 90° about its centre.

20

Tabu Search, Combination and Integration

Dr. Ahmed S. Al-Mahmeed

Department of Computer Science, College of Business Studies,
Public Authority for Applied Education and Training
P.O. Box 15084, Daiyah, State of Kuwait 35451
e-mail: mahmeed@paaetms.paaet.edu.kw

July 1995

Abstract:

This work studies the shortest path problem as represented by finding the route with least processing time in data communication networks. This is accomplished by a more effective procedure, known here as the SARA algorithm, that combines tabu search with other established search techniques such as backtracking, label-setting, and label-correcting.

The main objective of this work is the adaptive optimization of data network resources within specified constraints, and a demonstration of the benefits in applications to network management. In performing the optimization, consideration is given to both the total (processing) time for a message to travel from source node to destination node and the computational (processing) time to achieve this travel time.

Three major search techniques are combined with tabu search, namely backtracking, label setting (Dijkstra), and label correcting (Bellman-Ford). These techniques were chosen because of their common use to find shortest paths in data communication networks. Computations of the approach presented demonstrate that combining tabu search with the above search methods results in a procedure that enhances tabu search. Results and comparisons of the four methods are provided.

Key Words: backtracking , Combination, label-correcting, label-setting, SARA, tabu search.

1. Introduction

Many search algorithms have been extensively investigated, developed, and modified. A common approach used in dealing with search algorithms is that each algorithm, or method, is studied with respect to its parameters or characteristics generally on some classic problem such as the traveling salesman problem. Comparisons of different types of algorithms have also been extensively studied.

The development, testing and investigation of combined, integrated, or hybrid, systems have been less extensively studied. Many well known search algorithms offer the potential for combination and integration with others, with some prospect of enhanced performance.

SARA (Search Algorithm for Routing Adaptation) is named after the objective for which it was devised. The algorithm achieves its objective by combining the features of tabu search with other search techniques, in our case backtracking, label-setting and label-correcting, resulting in an enhanced and more effective search strategy. Three types of search techniques are combined with tabu search. These are: backtracking, label setting, and label correcting. Other methods could also be included, such as simulated annealing and genetic algorithms. The main tabu search method used is based on a branch and bound technique.

The concept of dynamic tabu search has not been applied satisfactorily to solve the problem of message routing [GeHL 91, ScSt 80]. In addition, knowledge based concepts (for example, dependent on past events and using the idea of look-ahead) have not been used extensively in managing data communication networks.

The goals of this work is to develop procedures, algorithms, or strategies to achieve better resource utilization and better control of the process of transmitting messages through data communication networks, with "least" cost and "greatest" reliability. Constraints to be considered when dealing with this include :

- Processing Delay.
- Propagation Delay.
- Maximum link capacity and message priority.
- Bandwidth limitation.
- Management rules and regulations.

For each message, we need to have a network with n nodes and m messages per node. We need to find a path P that has "least" processing time T for a message to travel through the network. This is to be accomplished using tabu search in conjunction with other search methods.

2. The Specific Objective

The core objective of the research is the optimal use of adaptive routing in data communication network using tabu search concepts, within specified constraints and restrictions, and to demonstrate the benefits of application to network management.

Consider the case where we have n nodes (or stations). A prescribed P_{ij} of two-way paths (or channels) join each pair (i,j) of nodes. These paths are used in a form of incidence matrix of the digraph representing the network in concern. Since the goal is to find the "quickest" path between source node i and destination node j , messages need not go directly from i to j . They may be routed through other nodes, depending on certain factors such as: the number of queued tasks at j , the number of nodes in a route or path, the number of available routes from source to destination, and the overall average processing speed of the intermediate nodes (nodes between source node and destination node). Delay penalties can be considered when the search is forced to choose a "longer" path due to some circumstances and the adaptivity concept is applied.

In Figure 1, a network of 8 nodes is to be used to transmit one message from node 1 (source) to node 4 (destination). [Assume all nodes have the same processing rate]. The oval shapes with an arrow pointing to each one of the intermediate nodes represent the number of messages waiting to be processed at that node. A node with more messages queuing will have longer processing time for a message traveling from node 1 to node 4 since no pre-emption is allowed (no message priority). The network has two "quickest" paths. These are 1-2-3-4 and 1-8-4 each with total of 2 waiting messages. Since path 1-8-4 has fewer nodes, this secondary criterion is used to select this path as the one to use.

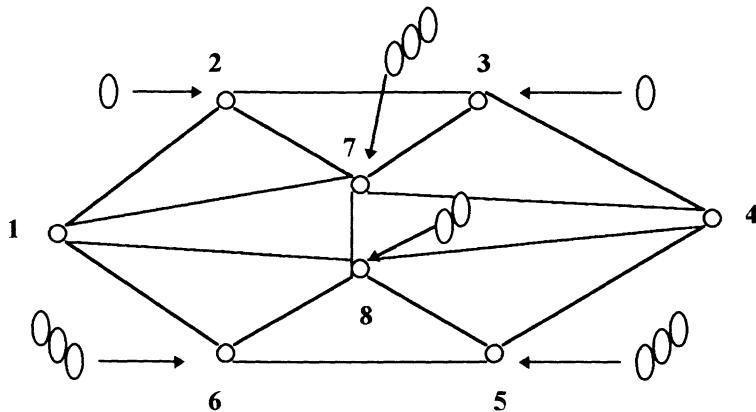


Figure 1: A Simple Communication Network

Assume T is the total processing time in a path in seconds and all messages have the same processing time. The goal is to minimize T , where

```

for i,j = 1 to n
    if  $t_{ij} < t_{i+1,j+1}$ 
         $T = T + mt_{i+1,j+1}$ 
    else  $T = T + mt_{ij}$ 
subject to  $60 \geq t \geq 0$  and  $m, n, \geq 1$ 

```

where n is number of nodes, m number of messages at each intermediate node, t_{ij} is the processing time for a message to move from node i to node j in seconds, i is the source node index, and j is the destination node index. t_{ij} is generated randomly and ranges from 0 to 60 sec., n and m are given as input parameters.

Then the required algorithm can be represented as:

1. Find the quickest path in a network to move a message from source node i to destination node j using predetermined search technique.
2. If processing time at node j is less than processing time at any neighboring nodes then move the message to node j and update tabu list (for tabu search), else move to the next "quickest" path.
3. If two paths have exactly the same total processing time T , then choose the path with least number of intermediate nodes if any exists.
4. Use the result obtained in step 1 as a starting solution for processing tabu search.
5. Perform steps 2 and 3 for tabu search.
6. If the result obtained in step 4 "better" than the result obtained in step 3, then output the "better" result, otherwise output the same result of step 3.

This work contributes to network management by studying the advantages of using search techniques derived through the combination and integration of tabu search with several other types of search strategies.

In more specific terms, this work demonstrates that with the use of tabu search technique in conjunction with major search methods enhances certain aspects of the management of data networks, specifically message routing.

3. The SARA Approach

The SARA (Search Algorithm for Routing Adaptation) algorithm uses a new approach based on the features of tabu search applied to data network for enhancing message routing. It is a deterministic approach applied to distributed type network. Three main factors are studied here in this approach. These factors are the CPU time used to reach the desired results, search effort represented as iterations per second, and quality of results.

The kind of combination process used in implementing SARA algorithm approach, specially the backtracking integration, is a simple long term memory function of tabu search strategy proposed by [Glov 90] where the recording of not just good past solutions, or moves, but rather have the goal of returning to good unselected neighbors of past solutions. This concept is expressed in terms of "restructured neighborhoods" [see [Glov 90]], where attractive unselected options are stored in sets of current and next moves (that start empty).

The approach used here corresponds to a very simple version of search technique where "best solutions" in the next move (which also implicitly records unselected neighbors of these solutions if the same choice made before it is prevented). Then periodically transferring the best element of next move to current move, choosing this to continue the search. The restructured neighborhood template allows tabu memory associated with solutions to be transferred with them.

To distinguish tabu search combined with SARA from branch and bound, notice first of all that a branch and bound policy of going back to good solutions is called "jump tracking" rather than "backtracking" (because backtracking always goes to the most recent predecessor in the tree where an unexplored branch exists, regardless of quality). Consequently, the type of long term tabu search approach that periodically resumes search from "screened solutions" from the search history more nearly resembles a jump

tracking strategy. But jump tracking is tied to different memory mechanisms, where each solution is embedded in a tree [LaGl 91]. Unexplored neighbors recovered by reference to tabu search memory, if their short term (recency based) memory is recovered with them, establish a search pattern that correspond to "undercutting" earlier parts of a tree structure, and shifting entire tree segments together (instead of shifting a single branch).

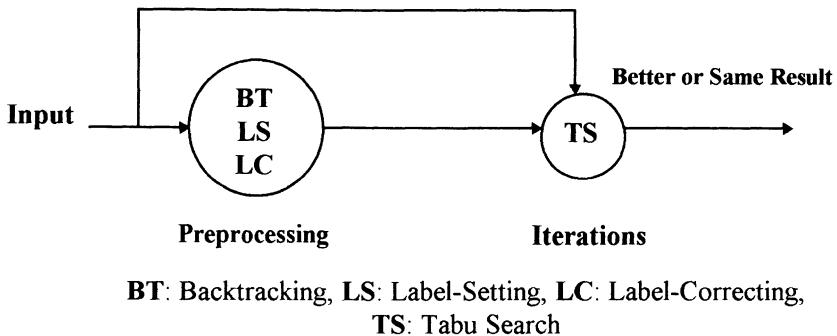


Figure 2: SARA Approach

The approach used here is a simple process of combining tabu search with three major search techniques. This is done to enhance the performance of tabu search with the aid of other search methods. Figure 2 shows the principle idea of SARA. The three search methods are used as a preprocessing stage where data and known information are entered. The three methods perform calculations of their own and then send the result to tabu search for more refinement of the result. Each method is used by itself with tabu search.

Considering tabu search, the SARA algorithm shows promise and potential benefits for data communication networks and network management. The routing decision for a message is taken at each node independent of the source of the message. A node takes the decision based on the properties of the network, such as the number of nodes, or hops, in a path and link reliability, the traffic at each node and the capacities of the links. Using all these factors, a minimum value of the objective function is to be calculated using three major search techniques and then refined by tabu search. The message is routed on the path with the lowest value of the objective function, in case of minimization. The objective function value is the value obtainable by SARA algorithm and it might not be the global

value. The most important aspect of this approach is that it generates very few redundancies [LaGl 91]. The full detailed SARA algorithm can be found in [Mahm 94].

4. Computational Results

Since the purpose of this experiment is to compare the performance of search methods under similar circumstances, a common starting point and a tabu_size of 7 were used [GILa 92]. These selections are based on preliminary experience with tabu search [HZLW 90], where a number of different settings were tested. Each method was allowed to run to termination at the termination count. Comparisons are made on the basis of solution quality obtained within the specified iteration count. Table 1 shows the results for the tabu search for the five test sets used with number of nodes n and number of messages per node m as input parameters and the processing time at each node generated randomly.

Number of Nodes	100	200	400	800	1000
Number of Messages	25	50	100	200	250
CPU Time in Seconds	6.3	27.7	153.7	696.3	916.3
Iterations	151	355	1102	2628	2720
Average Objective Function	6203643	27045910.8	111422355.4	441543907.3	709171175.5

Table 1: Tabu Search Factors of Comparisons

5. Combined Search Algorithms

Routing methods may involve the use of a number of simple graph theoretic algorithms. Consider, for example, the shortest path problem, in our case the path with least total processing time. We are given a processing time for every node in the network, and the objective is to find a path joining two given nodes that has minimum processing time.

The problem of obtaining efficient routing procedures for fast delivery of messages to their destinations is of utmost importance in the design of modern data communication networks. A large variety of routing algorithms have been developed and implemented. In the absence of sufficiently developed theories many of these algorithms are based on intuition, heuristics, and simulation.

Using the same data sets, the combination of the three main search methods with tabu search (TS) is tested. When implementing search methods in data communication networks, most search

methods are not considered ideal search techniques as they stand. By combining the above search methods with TS, an improved search may be established. Backtracking combination (TSBT), label setting combination (TSLS), and label correcting combination (TSLC) all have been tested and computational experiments performed. The results are shown in tables 2, 3, and 4.

Hybrid methods usually used to combine two or more moves, or solutions, into the same search method as in tabu search hybrid where it combines insert and swap moves, but the purpose of these experiments is to investigate whether combining other search methods into tabu search will provide simpler computational performance.

Number of Nodes	100	200	400	800	1000
Number of Messages	25	50	100	200	250
Average Objective Function	6204339.0	27042137	111453577.6	441538076.5	701998577.7
CPU Time in Sec.	7.9	41.2	220.6	1294.3	5201.5
Iterations	196	567	1644	5176	17135

Table 2: Tabu Search Backtracking (TSBT) Combination

Number of Nodes	100	200	400	800	1000
Number of Messages	25	50	100	200	250
Average Objective Function	6212311.6	27075937	111601272.8	439740883.7	709497012.6
CPU Time in Sec.	6.3	31.4	154.2	697.7	916.3
Iterations	146	407	1038	2622	2708

Table 3: Tabu Search Label Setting (TSLS) Combination

Number of Nodes	100	200	400	800	1000
Number of Messages	25	50	100	200	250
Average Objective Function	6215536.2	27075896.8	111603155.8	441738290.3	709785659.6
CPU Time in Sec.	6.3	31.3	154.4	698	914.3
Iterations	145	407	1043	2617	2695

Table 4: Tabu Search Label Correcting (TSLC) Combination

In Figure 3 we notice that TSBT has the highest number of iterations and seems to increase exponentially with the increase of problem size with the other search methods seems almost linear up to problem size of 800/200 nodes/messages where they level off beyond that. Iterations tend to increase with the increase of problem size. TS, TSLS and TSLC seem to level off for problem size greater than 400 nodes.

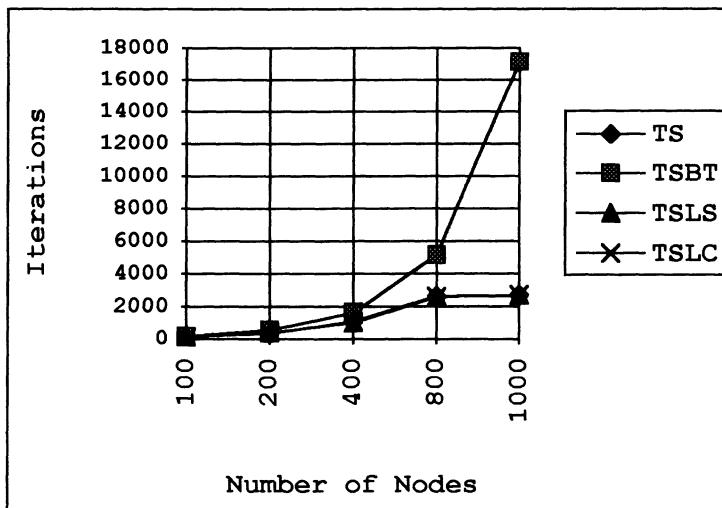


Figure 3: Iterations versus Number of Nodes

Notice that TS, TSLS, and TSLC are on each other but not exact

6. Testing and Evaluation

In this work four important factors are of concern as comparison tools. These are as follows:

- *Quality of result* means which of the four methods reaches least value of the objective function, in case of minimization process. This is rated as good or no-good, where good is considered a positive point.
- *The effort* exerted by an algorithm to make an improvement and it is classified as high or low. Low effort is considered a positive point.
- *CPU time* in seconds which is the time that a method takes to complete the experiment. It is rated long or short, where short is considered a positive point.

The goal is to choose an algorithm that has most positive points. That is the algorithm with good result, low effort, and short CPU

time. The algorithm that reaches, or come close to, the above goal is the better algorithm.

In the following tables, tables 5 to 8, the overall rating for each method is shown using the information obtained from tables 1 to 4.

Problem Size	100	200	400	800	1000
Result Quality	Good	No Good	Good	No Good	No Good
Iterations (Effort)	High	Low*	High	High	Low*
CPU Time	Short	Short	Short	Short	Short*

Table 5: Tabu Search Evaluation

* Indicates insignificant difference (quality close)

Problem Size	100	200	400	800	1000
Result Quality	Good*	Good	No Good	No Good	Good
Iterations (Effort)	High	Low	High	High	High
CPU Time	Long	Long	Long	Long	Long

Table 6: TSBT Evaluation

* indicates insignificant difference (quality close)

Problem Size	100	200	400	800	1000
Result Quality	No Good				
Iterations (Effort)	Low	High	Low*	Low*	Low
CPU Time	Short	Short*	Short*	Short*	Short

Table 7: TSLS Evaluation

* indicates insignificant difference (quality close)

Problem Size	100	200	400	800	1000
Result Quality	No Good				
Iterations (Effort)	Low	High	Low*	Low*	Low
CPU Time	Short	Short*	Short*	Short*	Short

Table 8: TSLC Evaluation

* indicates insignificant difference (quality close)

The way in which these tables are established based on the goal of the output results. The goal is to have "good" (minimized) average result quality, low iterations (low effort), and short CPU time. These are called goal elements. Each element is classified into two states. The element of result quality is rated as "good" and "no-good". This is determined by considering the "lowest" value of the

objective function calculated by one of the methods as "good", and the others as "no-good". If the difference between two, or more, results are not very good, they are considered having the same rating. Significance of difference in results depends on how far these results are from each other. The element of iteration is rated either high or low, and CPU time is either short or long.

Using the information shown in previous tables, the evaluations method is established as follows. We have four factors of interest. Every method that satisfies, or close to satisfy, the goal elements will have its element evaluated as positive and receives "positive" point. For example, when one search method have the iterations element rated low, then this search method will have a positive point toward this element. Table 9 shows the evaluation of the four search methods using this scheme.

	Tabu Search	TSBT	TSLS	TSLC
Result Quality	2	3	1	0
Iterations (Effort)	2	1	4	4
CPU Time	5	0	5	5

Table 9: The Positive Points Evaluation

From Table 9, TSBT has performed better than tabu search with respect to effort, TSLS and TSLC performed better than tabu search with respect to effort. Tabu search performed better in some goal elements, this supports the claim in this work that combining tabu search will enhance the search process depending on the type of the combined search method. This is clear if we compare the result quality between tabu search, TSBT, and TSLC.

In conclusion, TSBT should be used where result quality is more important, TSLS and TSLC should be used if iterations are more important, and TSLC should be used when the optimization rate is more important.

7. Conclusion

From the work presented here, the combination techniques show promising outcome in improving data network management than using tabu search alone. The drawbacks of TS have been eliminated. TSBT is outperforming TS in regard to value of objective function and search process effort (iterations).

The process of combining tabu search with other major search methods provides an improved procedure for the management of data networks and routing. Almost every known search technique has some drawbacks and by using the combination process, represented by the SARA algorithm, these drawbacks can be reduced or eliminated.

8. References

[GeHL 91] M. Gendreau, A. Hertz, and G. Laporte, *A Tabu Search Heuristics for the Vehicle Routing Problem*. Management Science, (1991).

[GiLa 92] Fred Glover and Manual Laguna, *Tabu Search: A Chapter in Modern Heuristic Techniques for Combinatorial Problems*. Graduate School of Business Administration, University of Colorado at Boulder, USA (1992).

[Glov 90] Fred Glover, *Tabu Search - Part II*. ORSA Journal on Computing, 2(1990) 4-32.

[HZLW 90] H.H. Huang, C. Zhang, S. Lee, and H.P. Wang, *Implementation and Comparison of Neural Network Learning Paradigms: Back Propagation, Simulated Annealing, and Tabu Search*. Technical Report, Department of Industrial Engineering, University of Iowa, USA(1990).

[LaGl 91] Manual Laguna and Fred Glover, *Integrating Target Analysis and Tabu Search for Improved Scheduling Systems*. Expert Systems with Applications: An International Journal, (March 5, 1991).

[Mahm 94] Mahmeed, Ahmed S., *Adaptive Routing in Networks Using Tabu Search*. Ph.D. Thesis, University of Strathclyde, Glasgow, Scotland, UK, 1994.

[ScSt 80] M. Schwartz and T. E. Stern, *Routing Techniques Used in Computer Communication Networks*. IEEE Transaction on Communications, (April 1980).

Vector Quantization with the Reactive Tabu Search

Roberto Battiti*, Giampietro Tecchiolli† and Paolo Tonella†

*Dip. di Matematica, Università di Trento

†Istituto per la Ricerca Scientifica e Tecnologica

38050 Povo (Trento), Italy

E-mail: battiti@science.unitn.it, {tec, tonella}@irst.itc.it

Abstract:

A novel application of the Reactive Tabu Search to Vector Quantization (RTS-VQ) is presented. The results obtained on benchmark tasks demonstrate that a performance similar to that of traditional techniques can be obtained even if the *code vectors* are represented with small integers. The result is of interest for application-specific VLSI circuits.

Key Words: Tabu Search, Vector Quantization, Reactive Heuristics.

1. Introduction

Sub-symbolic Machine Learning (ML) has become a paradigm for the development of systems that tackle recognition or classification tasks by mimicking the adaptation and learning capabilities of living organisms. While the “knowledge” in symbolic systems is mainly represented through logical rules (e.g., in Expert Systems, Artificial Intelligence), in sub-symbolic systems the knowledge is implicit in the values of the system parameters.

In sub-symbolic systems a strong emphasis is put on the self-organization properties: the system *evolves* toward a better functionality. There is a strong link between self-organizing systems and optimization: if the effectiveness of the system for a given task is expressed by a function of its parameters, a better functionality is

obtained by optimizing this function (while the system architecture is related to its *generalization* capabilities).

As soon as the importance of optimization for learning was recognized, many algorithms have been proposed that are often derived from continuous optimization (like the “backpropagation” algorithm in Rumelhart *et al.* (1986)). The approach presented here adopts a different strategy: first the search space is transformed from continuous to combinatorial by discretizing the internal parameters, then a heuristic for combinatorial optimization, the Reactive Tabu Search (RTS) of Battiti and Tecchiolli (1994), is applied. The RTS algorithm complements the Tabu Search heuristics proposed in Glover (1989, 1990) with reactive schemes to self-tune parameters and a with second diversification mechanism to avoid search confinements analogous to *chaotic attractors*.

Vector Quantization is the problem of finding an optimal set of vectors (*codebook*) so that the *distortion* (i.e., distance) associated to the mapping of input vectors to vectors selected from the codebook is minimized. This mapping permits the replacement of a vector with its closest codebook entry. VQ is of interest in classification tasks and data compression applications: by transmitting the indices of code vectors a sizable reduction in the transmission rate can usually be accomplished with a limited loss in signal quality.

In the following sections, first the use of RTS in the framework of Vector Quantization is described (Sec. 2). Then the experimental results obtained on a set of benchmark tasks are presented and compared with those of alternative techniques (Sec. 3). A final discussion concludes the paper (Sec. 4).

2. Vector Quantization

A Vector Quantizer (VQ) Q of dimension k and size N is a mapping from a vector in k -dimensional Euclidean space, R^k , into a finite set C containing N output or reproduction points, called *code vectors* or *codewords*:

$$Q : R^k \rightarrow C \quad (1)$$

where $C = \{y^{(1)}, y^{(2)}, \dots, y^{(N)}\}$ and $y^{(i)} \in R^k$ for each $i \in \{1, 2, \dots, N\}$. The set C is called *codebook* or, simply, the *code*.

Every vector quantizer Q generates a *partition* of R^k into N regions or *cells*, R_i for $i \in \{1, 2, \dots, N\}$. The i -th cell is defined by:

$$R_i = \{x \in R^k : Q(x) = y^{(i)}\} \quad (2)$$

and it is sometimes called the *inverse image* or *pre-image* of $y^{(i)}$ under the mapping Q .

The performance of a VQ is measured by the *distortion*. A distortion measure d is a nonnegative cost $d(x, y^{(i)})$ associated with the mapping of an input vector x to a reproduction vector $y^{(i)}$. The performance of a system can be quantified by an average distortion $D = E[d(x, y)]$ between the input random variable x and the final reproduction $y = Q(x)$.

The most convenient and widely used measure of distortion between an input vector x and a quantized vector $y^{(i)} = Q(x)$ is the *squared error* or squared Euclidean distance:

$$d(x, y^{(j)}) = \sum_{i=1}^k (x_i - y_i^{(j)})^2 \quad (3)$$

This measure is frequently associated with the energy or power of an error signal and therefore has some intuitive appeal in addition to being an analytically tractable measure.

An important special class of VQ's that is of particular interest, called *Voronoi* or *nearest neighbor* VQ's, has the feature that the partition is completely determined by the codebook and a distortion measure. In a *Voronoi* VQ the partition cells are given by:

$$R_i = \{x : d(x, y^{(i)}) \leq d(x, y^{(j)}) \ \forall y^{(j)} \in C\} \quad (4)$$

Each cell R_i consists of all points x which have less distortion when reproduced with code vector $y^{(i)}$ than with any other code vector. Thus the encoding operation simply selects the minimum distance code vector from the codebook. The design of a Voronoi VQ consists of finding the code vectors that minimize the average distortion D . Among the available methods to determine the codebook are:

K-means It is the most widely used method for determining C , based on an iterative codebook improvement algorithm. Each

iteration consists of two steps. In the first step the input vectors are classified according to the minimum distance from code vectors, in the second step the centroid is computed for each class, and it becomes the new code vector for that class (see Gersho and Gray (1992)). K-means implementation is that of Appl. Statistics (1979).

Simulated Annealing Successful applications of SA for VQ codebook design have been reported in Vaisey and Gersho (1988), and Flanagan *et al.* (1989) (here the details for a fast implementation can be found). This algorithm proceeds backwards: first the input vectors are assigned to partition cells, and then the centroids of the partition cells are computed to form the codebook. After starting from an initial random assignment, at each iteration one input vector is randomly selected in order to change its assignment to a cell. The change is accepted or rejected with a probability that depends on the distortion variation caused by the perturbation.

Self Organizing Maps A different approach to VQ is studied in Kohonen (1989), in the framework of Unsupervised Learning. A *Self-Organizing Map* (SOM) consists of a finite-dimensional array of units (*neurons*) that are positioned in the high-probability density portions of the input space. There is evidence that feature extracting maps are present in the brain, where input vectors representing similar features are mapped onto close neurons of the cortex. Biologically-plausible weight update rules are the starting point for Kohonen's SOM algorithm: each neuron n_i is connected to every input component x_j through a weight w_{ij} . An input vector is mapped onto the neuron n_c that minimizes the distance $\sum_j (w_{ij} - x_j)^2$. The learning algorithm selects the winning neuron and its topological neighbors, and updates their weights in order to make them more similar to the input: $w_{ij}(t+1) = w_{ij}(t) + \alpha(t)(x_j(t) - w_{ij}(t))$, where t represents time, $\alpha(t)$ is a time-decreasing learning factor. Kohonen SOM can be used as a VQ, if each neuron is regarded as a code vector (its weights being the code vector components). In fact, the learning algorithm forces the weights to adapt to the input probability density, in this way minimizing a distortion measure given by

the Euclidean distance. The method has the additional property that the “topology” of the input vectors is preserved, that is, the input vectors are mapped onto the network in an ordered fashion.

2.1 RTS-VQ: use of the Reactive Tabu Search

As in the case of K-means, the input vector is mapped to the closest code vector (the one with minimal Euclidean distance). The average distortion measure to be minimized is therefore:

$$D = \sum_{i=1}^n \min_j d(x^{(i)}, y^{(j)}) \quad (5)$$

where n is the number of input vectors $x^{(i)}$, and $y^{(j)}$ is the j -th code vector. While K-means code vectors move in the “continuous” space R^k , RTS code vectors are discretized. Each component y_i is allowed to assume 2^l different integer positive values \hat{y}_i (l is the length of the binary string representing the component). The relation between the integer values and the real code vector is:

$$y_i = Min_i + (Max_i - Min_i) * \hat{y}_i / (2^l - 1) \quad (6)$$

where Min_i and Max_i represents the minimum and maximum values of the i -th input vector component x_i , respectively.

The task is now combinatorial and the RTS algorithm can be applied in a straightforward way to search for a good set of \hat{y} vectors. The average distortion is minimized in the configuration space given by the binary string obtained by concatenating all integer-valued components, coded with the Gray encoding \hat{y}_i . The strong assumption is that no significant loss of performance derives in passing from real-valued to integer-valued parameters. It is quite of interest to see that this assumption is confirmed even when small integers are used, as we show in the experimental subsection.

3 RTS-VQ: Benchmark Results

The four VQ’s presented before (K-means, Kohonen’s SOM, SA and RTS-VQ) have been applied to the following benchmark data sets ($U[a, b]$ represents a random variable uniformly distributed in the range $[a, b]$, $G[a, b]$ a random variable normally distributed with mean a and square variance b):

Uniform: 2-D probability distribution that is uniform over quadrants II and IV and zero otherwise: $\{(x, y) \in R^2 : x = U[-1, 0], y = U[0, 1] \text{ or } x = U[0, 1], y = U[-1, 0]\}$, 8000 patterns with 2 components (i.e. $n = 8000, k = 2$).

Sin: 2-D probability distribution that is different from zero only on the curve $y = \sin(x)$, x being uniformly distributed: $\{(x, y) \in R^2 : x = U[-1, 1], y = \sin(x)\}$, 20000 patterns with 2 components.

Gaussian: 2-D Gaussian probability distribution: $\{(x, y) \in R^2 : x = G[0, 1/4], y = G[0, 1/4]\}$, 16000 patterns with 2 components.

Device: Measurements taken from a real device “normally operating” or “overheating” and available from Kohonen SOM-PAK (obtained by Helsinki University of Technology) 3840 patterns with 5 components.

The first three data sets are artificial, and are used to test the algorithms with well known probability distributions. The last data set contains real measurements, and Vector Quantization can be used to map them onto a finite set of states, useful for successive diagnosis.

3.1 RTS-VQ: Results

If the input probability distribution is known and the input components are independent and identically distributed random variables, it is possible to use the Signal to Noise Ratio (SNR) as a performance measure:

$$SNR = -10 \log_{10} \left(\sum_i \min_j dist(x^{(i)}, y^{(j)})^2 / (n * \sigma^2) \right) \quad (7)$$

where σ is the variance of each component of the input signal. Under these assumptions, an upper bound is available (Shannon’s limit):

$$SNR_{max} = 20 \log_{10} \frac{\sigma}{2^R} \quad (\text{where } R = \log_2(N)/k) \quad (8)$$

If the probability distribution of the input is not known, or the input is multidimensional with components that are not identically distributed random variables, the Quantization Error (QE) can be used to evaluate the VQ performance:

$$QE = \sum_i \min_j dist(x^{(i)}, y^{(j)})/n \quad (9)$$

VQ training has been executed with K-Means, SOM, SA, and RTS on the four available data sets. Code vectors have been initialized by setting them equal to randomly extracted vectors of the data set. In RTS a subsampling of the neighborhood is used (6 randomly extracted moves are evaluated at each step). 8 bits per code vector component are used in RTS, while double-precision floating-point numbers (64 bits) are used in K-Means, SA, and SOM.

On average, VQ training with RTS required 1900 seconds, 5200 sec with SA, 95 sec with SOM and 5 sec with K-means. The fastest algorithm is K-means, but RTS has the advantage to require only integer-valued parameters, and to be realizable on a parallel architecture in a straightforward way, thus reducing its execution time.

In SOM close input vectors are mapped to close neurons. Therefore, in the following figures, each Kohonen code vector is connected with lines to its neighbors. Fig. 1 shows the location of the code vectors on the plane for the first (*Uniform*) data set as it was determined by the four algorithms considered. Eight code vectors are located by K-Means and RTS very close to the known optimal positions (the eight points with coordinates: $(\mp 1/4, \pm 1/4)$, $(\mp 3/4, \pm 1/4)$, $(\mp 1/4, \pm 3/4)$, $(\mp 3/4, \pm 3/4)$). The length of the binary string representing a configuration is 128 ($= 8 \cdot 8 \cdot 2$). For *Sin* data set 20 code vectors were used with a configuration string of 320 bits ($= 20 \cdot 8 \cdot 2$), the final location is shown in Fig. 2. Fig 3, 4 shows the location of the 16 code vectors for the *Gaussian* data set. In this case the RTS binary string has a length of 256 bits. The vectors are placed correctly in the high probability density regions of the input space.

It is worth noticing that for the Gaussian distribution RTS determines a code vector location based on two concentric circles, with a very regular polygonal disposition on them (Fig. 3).

Tables 1 and 2 show a quantitative comparison between the different algorithms. Let us note that good vector quantizers must have high SNR but low quantization error. SNR is not available

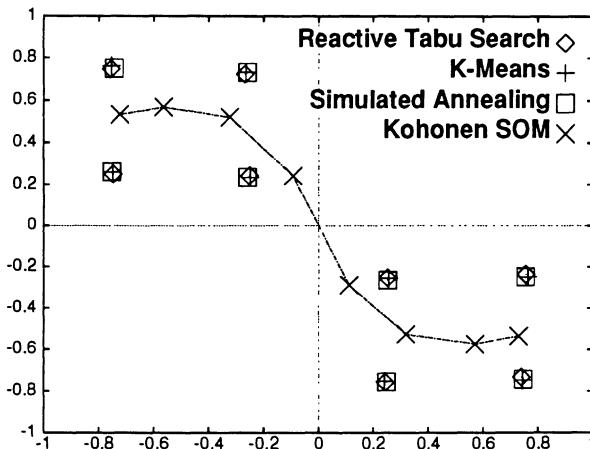


Figure 1: Code Vectors for a Uniform Distribution over Quadrants II and IV.

for the *Sin* data set because the two components are not identically distributed, and for the *Device* data set because input statistics are unknown. Shannon's limit is far from the actual results for the *Uniform* data set. This is because it is reachable by a VQ only when the vector dimension goes to infinity, but nothing more accurate is stated for finite values of input dimension.

Table 1: Signal to Noise Ratio (dB)

Data Set	Kohonen	K-Means	SA	SLS	RTS	Limit
Uniform	0.00	3.06	3.06	3.06	3.06	16.81
Gaussian	8.77	11.82	11.00	11.75	11.75	12.04

It is worth noticing that RTS and K-means perform in a similar way, but RTS uses fewer bits for coding vector components (8 versus 64): the computation required during operations can be much less if appropriate hardware is used (small integer adders and multipliers with respect to floating-point units). Kohonen's SOM performs worse because minimization of QE is not the only goal of this algorithm: its capability to preserve input ordering makes close input vectors be mapped to topologically close code vectors.

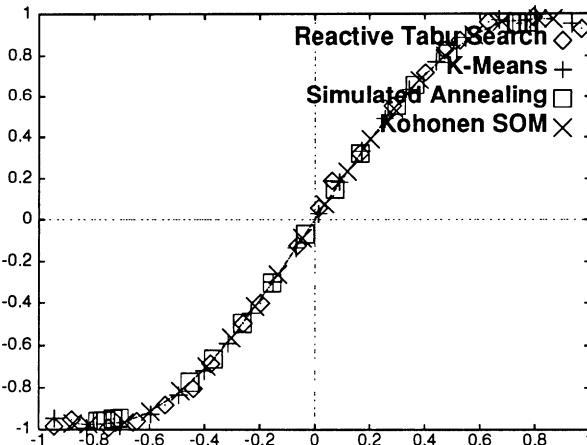


Figure 2: Code Vectors for the Sin Function.

Table 2: Quantization Error

Data Set	Kohonen	K-Means	SA	SLS	RTS
Uniform	0.256	0.190	0.190	0.190	0.190
Gaussian	0.149	0.124	0.123	0.125	0.125
Sin	0.871	0.038	0.073	0.038	0.045
Device	3.494	2.291	4.283	2.169	2.361

SA has been implemented according to Flanagan *et al.* (1989). For the choice of the parameters of SA a preliminary run of 1024 iterations has been executed. During each iteration input vectors are randomly assigned to code vectors, and the magnitude Δ of the distortion change is computed. Initial and final temperatures are then given the values $T_0 = 2\Delta_{MAX}$, and $T_N = 0.01\Delta_{MEAN}$, where N is the iteration number. The *schedule* parameter α is chosen by assuming a temperature exponential decrease ($T \leftarrow T/\alpha$ at each iteration) so that after N iterations the system is *frozen*: $\alpha = \sqrt[N]{\frac{T_0}{T_N}}$. SA obtains results similar to those of RTS and K-means on the *Uniform* and *Gaussian* data sets, while it performs worse on *Sin* and *Device*. For a comparison, the results of Stochastic Local Search (SLS) have been reported. In SLS one random change is

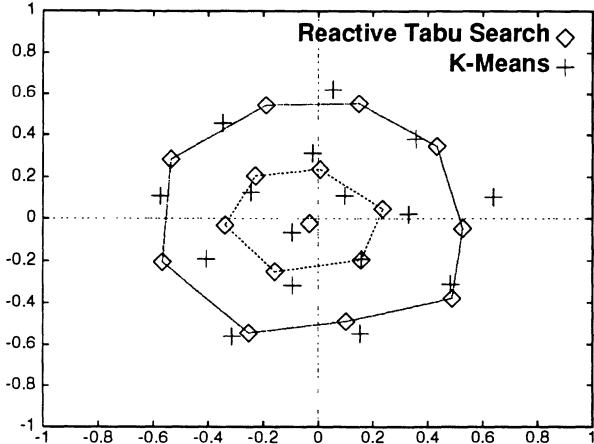


Figure 3: Code Vectors for a Gaussian Distribution (RTS and K-Means).

tried at each step, only changes that generate a decrease in the distortion are accepted, while SA accepts also worsening changes, with a probability diminishing with time. In other words, SLS is a “zero temperature” SA version. SLS produces lower or equal values of quantization error than SA, proving that, at least for these data sets, benefits are not provided by the detailed mechanisms of SA, but come from the transformation of the *fitness* function. In passing, the necessity of comparing complex heuristics with simple schemes is underlined.

4. Conclusions

RTS can be used as an effective training algorithm in the field of Vector Quantization, where code vectors are traditionally chosen in a continuous space. The discretization of the code vector components, required in order to apply the RTS combinatorial optimization heuristics, did not affect the performance in a significant way for the tasks considered. Let us note that the possibility of using small integers is of particular relevance if application-specific VLSI circuits are developed for VQ: the size of the hardware structures is greatly reduced with respect to those needed for floating-point computations, with effects on cost, speed, and possibility of parallelism at the chip level.

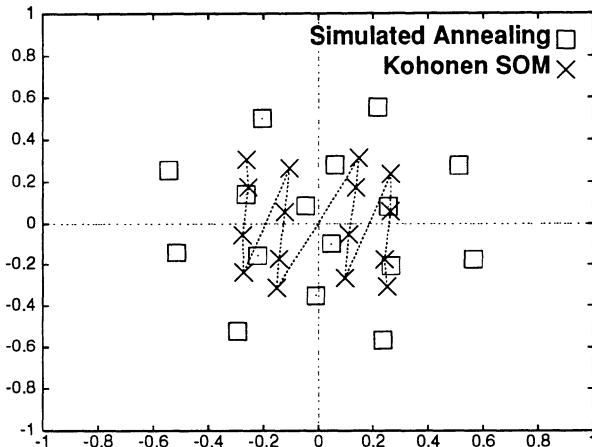


Figure 4: Code Vectors for a Gaussian Distribution (SA and SOM).

These results and those previously obtained in the field of sub-symbolic Machine Learning by Battiti and Tecchiolli (1995), *in press*, show that heuristics for Combinatorial Optimization can be profitably used. The ability of RTS to escape from local minimizers and to automatically diversify the search makes it a competitive technique in the ML field. Application-specific circuits developed according to this approach are of interest and producing a series of results in different application areas (Battiti *et al.* (1994)).

5. References

- Algorithm AS 136, *Appl. Statistics*, vol. 28, n. 1 (1979).
- R. Battiti and G. Tecchiolli, The Reactive Tabu Search, *ORSA Journal on Computing*, 6 (1994) 126.
- R. Battiti and G. Tecchiolli, Training Neural Nets with the Reactive Tabu Search, *IEEE Transactions on Neural Networks*, in press.
- R. Battiti, P. Lee, A. Sartori, and G. Tecchiolli, TOTEM: a Digital Processor For Neural Networks and Reactive Tabu Search, *Proceedings of the Fourth International Conference on Microelectronics for Neural Networks and Fuzzy Systems, MI-*

CRONEURO 94, Torino, IT, (IEEE Computer Society Press, 1994) p. 17.

- J.K. Flanagan *et al.*, Vector quantization codebook generation using simulated annealing, *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, (Glasgow 1989) p. 1759.
- A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*, ed. Kluwer Academic Publishers, (1992).
- F. Glover, Tabu search: part I, *ORSA Journal on Computing*, 1 (1989) 190.
- F. Glover, Tabu search: part II, *ORSA Journal on Computing*, 2 (1990) 4.
- T. Kohonen, *Self-Organization and Associative Memory*, ed. Springer-Verlag, (Berlin, 1989).
- D.E. Rumelhart, G.E. Hinton, and R.J. Williams, Learning internal representations by error propagation, *Parallel Distributed Processing*, ed. MIT Press, 1 (Cambridge, 1986), p. 318.
- J. Vaisey and A. Gersho, Simulated annealing and codebook design, *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, (New York, 1988), p. 1176.

Tabu Thresholding for the Frequency Assignment Problem

Diane Castelino* and Nelson Stephens**

* Department of Computing Mathematics
University of Wales College of Cardiff
P.O. Box 916, Cardiff, CF2 4YN, UK
E-mail: scmdjc@cm.cf.ac.uk

** Department of Mathematical Sciences
Goldsmiths College, University of London
New Cross, London SE14 6NW, UK
E-mail: nelson@cm.cf.ac.uk

Abstract:

The frequency assignment problem is to allocate frequencies to communication links such that the total interference is minimised. In this paper we investigate a tabu thresholding procedure to solve the frequency assignment problem. Several variants of the method are implemented using large computer-generated but realistic data sets. We report on these investigations, the selection of suitable parameters and compare the method with a tabu search for the same problem. We conclude that, in the same amount of time, tabu thresholding provides better quality solutions than tabu search.

Key Words: Frequency assignment problem, tabu thresholding, tabu search, heuristic, algorithm.

1. Introduction

The radio link frequency assignment problem occurs in both military and civil applications, Chiba, Takahata and Nohara (1992),

Raychaudhuri (1992), where it is necessary to assign radio frequencies to a large number of transmitters in a restricted region so that interference is minimised. The problem may be modelled as a combinatorial optimisation problem and is NP-hard, so that the generation of an optimal solution to a large instance of the problem requires excessive computing time. Recent research has focused on the use of heuristic algorithms to find suboptimal solutions.

The problem consists of a number of fixed sites. Each site has transmitters and receivers to be used for establishing communication links between personnel at different sites. The available frequencies for the links are separated by 50 kHz, equivalent to one channel, and hence come from a finite set. Some channels within a range may not be available for the application. Interference can occur when two links are assigned frequencies which are close. This arises when a transmitter or receiver of two links are at the same site (co-site) or are at a distance of several kilometers (far-site). The precise difference between frequencies required to avoid interference depends on the physical distances between equipment and the geography of the terrain as well as the frequencies themselves. A constraint, an absolute upper bound for the separation between two frequencies, can be computed from the terrain data by preprocessing. We consider the problem of finding an assignment of all frequencies of this model which reduces interference in the sense that as few constraints as possible are violated or that the total amount by which all constraints are violated is minimised. This model is actually a simplification of another model which also includes some constraints involving a lower bound on the absolute value of small linear combinations of three or more frequencies. Our tabu thresholding algorithm can easily be modified to consider these constraints as well. This also applies to any other heuristic algorithm implementation.

If there are N links and the available frequencies (measured in channels) come from F , a finite set of integers, an assignment can be represented as $\mathbf{f} = (f_1, \dots, f_N)$ with $f_j \in F, j = 1, \dots, N$. For each pair of links (i, j) with $1 \leq i < j \leq N$, preprocessing the terrain information gives an integer c_{ij} (possibly zero) such that no interference occurs if

$$|f_i - f_j| \geq c_{ij}.$$

There are $N(N - 1)/2$ such constraints. A large number of these, C , about $N^2/8$, are non-trivial, i.e. $c_{ij} > 0$.

Formally, the frequency assignment problem is to determine N frequencies $f_i \in F$, $i = 1, \dots, N$ so that interference is minimised. In this paper we consider two measures of total interference. The first, T_1 , is obtained by counting the number of violations. The second, T_2 , is obtained by summing, for each violation the difference between the constraint value and the separation between the frequencies. Hence

$$T_1(\mathbf{f}) = \sum c_{ij} \quad T_2(\mathbf{f}) = \sum (c_{ij} - |f_i - f_j|)$$

where each sum is over all i, j with $i < j$ and $|f_i - f_j| < c_{ij}$.

In our implementation, we use data obtained by simulating realistic Combat Net Radio assignment scenarios to determine the values of c_{ij} . Typically, the set F might be $\{1, \dots, A\}$ with $A = 50$ and the far-site values of c_{ij} are 0, 1, 2 or 3 and the co-site values are larger (about 10). Our test data consists of 6 scenarios with N taking the values 252, 282, 410, 450, 490 and 726. The value of C for these scenarios ranges from 7,059 to 75,306. In each scenario we have shown, by considering the subproblem consisting only of the co-site constraints, that a perfect solution without any interference does not exist.

The frequency assignment problem is equivalent to the classical graph colouring problem in the special case when every value of $c_{ij} = 0$ or 1 , Metzger (1970). In this formulation, each vertex represents a transmitter, each edge corresponds to $c_{ij} = 1$ and each colour represents a frequency and so colouring the nodes of the graph leads to a frequency assignment plan.

Hale (1980) produced a graph formulation of the frequency assignment problem and proposed to find the minimum span and order assignment. Costa (1993) has implemented simulated annealing and tabu approaches for finding the minimum span t -colouring for random graphs. De Werra and Gay (1994) developed a greedy algorithm heuristic also for random graphs. Note that the t -colouring problem is a special case of the frequency algorithm where there are no constraints between three or more transmitters.

The channel assignment problem has been studied by Duque-Antón, Kunz and Rüber (1993) who implemented a simulated annealing algorithm and by Mathar and Mattfeldt (1993) who investigated the use of several algorithms based on the simulated annealing approach but using a different model.

Castelino, Hurley and Stephens (1995) developed a tabu search algorithm for solving realistic frequency assignment problems. The tabu search implementation incorporated recency based and frequency based memory strategies to guide the search. In a comparative study of tabu search with a genetic algorithm and local steepest decent heuristic, the tabu search algorithm compared favourably with respect to solution quality and computation time.

In this paper, we propose a tabu thresholding technique for solving the frequency assignment problem. Tabu thresholding (TT) is an optimisation approach recently proposed by Glover (1995) that integrates ideas from tabu search, Glover (1986, 1989, 1990), and candidate list strategies, Glover (1974), into one overall framework. A significant distinction between tabu search and TT, however, is the absence of memory structures in TT. The objective of our research is to compare results obtained with Castelino, Hurley and Stephens (1995) to investigate whether the foregoing strategy can be relied upon as a primary guidance mechanism as opposed to the traditional memory based strategies. Other researchers have investigated the application of the tabu thresholding method in several areas such as graph drawing problems, Valls, Martí and Lino (1993), and rectangle packing problems, Dowsland (1995).

The outline of this paper is as follows. The next section reviews the basic ideas behind the tabu thresholding approach. Section 3 describes in detail our implementation of several variants and parameters of the tabu thresholding technique for solving the frequency assignment problem. Section 4 describes computational experiments obtained by applying these implementations to a set of computer-generated, realistic problem instances to determine good values for the parameters. In Section 5, our results are compared to a tabu search approach from the literature to examine the effects on the quality of the solution obtained and the total computation time. Finally, concluding remarks are discussed in Section 6.

2. Tabu Thresholding

In this Section we review the main ideas and principles of the tabu thresholding approach. For a full description of TT and enhancements, refer to Glover (1995). TT is a direct embodiment of the strategic oscillation approach, Glover (1977), with close links to the movement of the objective function value. The procedure is simple in that it does not depend on memory structures to guide the search, using only a candidate list strategy to isolate subsets of moves to be examined at each iteration. The primary aim of employing a candidate list is to reduce the overall computation time while reinforcing the aggressive aspect familiar to tabu search methods of taking a best move from each subset. Such an approach seems reasonable to employ since in our frequency assignment problem we are dealing with problems of a very large size.

TT consists of two alternating phases known as an *improving phase* and a *mixed phase*. Each iteration of the improving phase examines one subset of the candidate list. If the best move in this subset improves on the best solution so far in the phase then that move is carried out. The method proceeds in this way, disregarding a subset if no suitable move is found, until all such subsets in the candidate list have been tested and found unsuccessful. In order to prevent re-examination of the subsets in the same order after processing the entire list, the list is segmented into successive blocks of size b , where b is chosen to be relatively small in proportion to the total number of subsets. Each time a block is encountered the ordering of the subsets in the block is changed randomly. Hence, migration of the subsets from a particular block to an adjacent block in a subsequent pass is possible if b is not a divisor of the total number of subsets. Termination of the improving phase occurs when no improvement is possible (i.e. at a local optimum), resulting in an initiation of the mixed phase.

In the mixed phase, the subsets in the candidate list are completely reshuffled subject to placing the subset that provided the last improvement at the end of the list. The phase is similar to the improving phase but the acceptance criterion for moves is extended to allow non-improving moves as well as improving moves. The duration of this phase is in accordance with a tabu timing parameter t , although this phase may be terminated abruptly if certain aspi-

ration criteria are satisfied, such as improving the current globally best solution. The last solution of the mixed phase is then used as the initial value for the next improving phase.

The two phases are alternated, retaining the best solution obtained until a certain number of iterations have passed without an improvement in the objective function. The overall performance of the TT approach is dependent on three factors, viz, the determination of the choice of subsets in the candidate list and the value of b , the definition of acceptance criterion for selecting moves and the selection of technique for determining the tabu timing parameter t .

The acceptance criterion in Glover (1995) is a probabilistic best criterion based on ideas from probabilistic TS. This uses controlled randomisation to fulfil certain functions otherwise provided by memory. The duration of the mixed phase is determined by a randomly selected parameter t where t is allowed to vary between L and U each denoting fixed or variable upper and lower bounds on the number of iterations to be performed.

Dowsland (1995) has demonstrated the use of TT for solving the pallet loading problem. Results were compared to a non-monotonic simulated annealing (NMSA) approach on a testbed of problems for which there was a known optimal solution. Six different subset partitioning methods were compared using eight different variants. Results showed that screening in the improving phase appeared beneficial and contributed to cycling if used in the mixed phase. A probabilistic sampling of the subsets in the mixed phase was used to counter the effect of cycling. Overall it was reported that there appeared to be no significant difference between the performance of NMSA and TT.

Valls, Marti and Lino (1993) apply TT to the problem of minimising the number of arc crossings on a bipartite graph. They find that TT gave optimal solutions, when known, and outperformed the best algorithms in the literature.

3. Implementation

In this Section we describe our implementation of TT to solve the frequency assignment problem. Nine variants of the method are

implemented to varying levels of sophistication to assess the impact on the solution quality and computation time. We describe fundamental decisions required for the TT technique such as definition of subsets in the candidate list, the value of b , tabu timing parameter, choice criterion for accepting moves and screening.

An assignment is represented by $\mathbf{f} = (f_1, \dots, f_N)$ with $f_j \in F$ for $1 \leq j \leq N$. The neighbours of \mathbf{f} are those assignments where the array differs in precisely one component. Thus $\mathbf{f}' = (f'_1, \dots, f'_N)$ is a neighbour if there is an index j such that $f_j \neq f'_j$ but for each $i \neq j$, $f_i = f'_i$. An assignment has $N(a - 1)$ neighbours where a is the size of F . As usual, TT determines a sequence of assignments $\mathbf{f}^{(0)}, \mathbf{f}^{(1)}, \mathbf{f}^{(2)}, \dots$, where $\mathbf{f}^{(k+1)}$ is a neighbour of $\mathbf{f}^{(k)}$. The initial solution, $\mathbf{f}^{(0)}$, was made by randomly assigning the frequencies f_j .

3.1 Definition of Subsets - Improving Phase

The candidate list in the improving phase is divided into N subsets, one for each link. The neighbours in subset j are the $a - 1$ possible assignments to f_j . The N subsets are partitioned into blocks of size b , chosen empirically, and the subsets within each block are processed at random. An assignment in a subset is selected if it improves on the last improving assignment.

3.2 Initial Screening Approach

To accelerate the method, we investigate the use of a screening approach for the first iteration of the improving phase as suggested by Glover (1995). This requires the use of a partial evaluation process which we later describe to isolate potentially attractive moves before applying a full evaluation. To construct the initial candidate list we place the transmitters in order of importance.

For this construction, we assign a weight to each constraint in relation to its separation value c_{ij} . This means that constraints involving larger separation values, because they are harder to satisfy, receive larger weights. For example, $c_{ij} = 1$ is considered to be less important than a value of 10. The constraint weights are then used to create an individual weight for each transmitter representing the total of all the constraint weights pertaining to it. We class the important transmitters to be the ones that exceed the average separation weight and are screened into a critical set. It

is conceived that by generating thresholds to identify limits of this type and by screening moves by a partial evaluation, the algorithm may converge more rapidly to a solution than a random construction. After every subset in the critical set has been tested and no improvements made, we extend the candidate list to consider all possible subsets (i.e. all N transmitters). Termination of the foregoing process occurs when all N transmitters have been tested without improvements and hence a local optimum is reached.

3.3 Mixed Phase

In the mixed phase the choice rule is extended to accept the best move from each subset irrespective of whether or not it results in an improvement in the objective function. We return to the improving phase after a certain number of iterations have passed or an aspiration criterion has been satisfied, using the best move found in the mixed phase.

The subsets consist of the neighbours corresponding to K links where K is a parameter the value of which we choose empirically. Hence, finding the best move in a subset involves investigating $K(a - 1)$ frequency changes. We adopt the notion of Glover (1995) for the determination of the tabu timing parameter. After experimenting, we fix L and U , so that, each time on entry to the mixed phase, a uniformly random value for t is chosen satisfying $L \leq t \leq U$. The value of t determines the maximum number of iterations performed for that particular phase. However, we adopt an aspiration criterion to override this value so that, whenever a move was found which led to a best ever solution, an immediate return to the improving phase was made.

3.4 Subsequent Screening Measure

After the initial improving phase, a different screening criterion is employed for either phase that isolates subsets (i.e transmitters) exceeding an average measure of interference. We experiment with two such measures. One is based on the number of violations, T_1 , and the other is based on the sum of the positive discrepancies, T_2 . It was conceived that the latter would give a more realistic reflection of the severity of the interference. The closer the assignment reaches the desired separation values, the better the solution. Screening transmitters exceeding this measure allows transmitters

with gross incompatibilities to be rectified first and leaves minor problems for later. Note that we experimented with both T_1 and T_2 as screening measures within both phases irrespective of the measure of interference used as the global objective function.

We investigate whether the two screening options in either of the phases are beneficial in reducing the computation time or the quality of the results by helping the algorithm to locate good solutions more easily. Nine variants of the tabu thresholding approach were investigated and compared to assess their impact on the quality of the solution obtained also taking into account the computation time required to achieve a satisfactory solution. These variants are denoted by *BB*, *BD*, *BR*, *DB*, *DD*, *DR*, *RB*, *RD* and *RR*. Here *B* means that the screening uses T_1 as the measure; *D* means that the screening uses T_2 and *R* means that no screening took place at all. The first letter indicates the approach for the improving phase and the second letter the approach for the mixed phase. So, for example, *DR* means that the discrepancy measure was used to screen all the improving phases except the first but no screening was used in the mixed phase.

To summarise the screening techniques, at the beginning of the phase, the procedure is driven to satisfy constraints involving important links and at a later stage the search undertakes to satisfy constraints associated with lower weights.

3.5 Tabu Thresholding Algorithm The tabu thresholding approach is described as follows. The variables are:

max_iterations = maximum number of iterations.

k = iteration number.

t = tabu timing parameter.

rejected = set of subsets rejected.

Initialisation.

Set *rejected* = empty.

Generate a random initial solution.

Set *k* = 0.

Improving Phase.

Step 1. Screen; apply initial screening first time and *B*, *D* or *R* subsequently.

Step 2. Evaluate all moves in next subset (1 link) on candidate list

and select best move.

(On entering a block of size b randomise order of subsets.)

Step 3. If move improves then update solution and current best solution,

set $\text{rejected} = \text{empty}$

else add subset to rejected .

Step 4. If rejected full then go to *Mixed Phase*.

Step 5. $k = k + 1$. If $k < \text{max_iterations}$ then go to *Step 2*, else stop.

Mixed Phase.

Step 1. Screen (B , D or R) and select t where $L \leq t \leq U$ and set $t_cnt = 0$.

Step 2. Evaluate all moves in next subset (K links) on list and select best move.

Step 3. Update current solution. If current best solution improves then

update current best solution and go to *Improving Phase*.

Step 4. Set $t_cnt = t_cnt + 1$. If $t_cnt > t$ then go to *Step 1* of *Improving Phase*.

Step 5. $k = k + 1$. If $k < \text{max_iterations}$ then go to *Step 2*, else stop.

The algorithm terminates in either phase at *Step 3* if zero interference is obtained or at *Step 5* if we reach a pre-determined maximum number of iterations. The best solution found by the algorithm is then a suboptimal solution for the frequency assignment problem. The screening approach makes it possible for the candidate list to be a dynamic list that varies according to the status of the current solution. This dynamic strategy may improve the overall efficiency and diversity of the search process.

4. Experimental Results

The data for testing the algorithm consists of 6 scenarios with different values of $N = 252, 282, 410, 450, 490$ and 726 . The parameters of the TT algorithm are the nine screening variants (BB , BD , etc.), the block size b of the improving phase, the value of K and the bounds L, U of the mixed phase. We consider both objective functions, T_1 and T_2 . The termination criterion is to halt after either a fixed number of iterations or a fixed amount of time. Note that the values of K, L, U and b affect the average time required to perform

one iteration, so that these criteria are essentially different. Time is important because assignments are required within a specified time (e.g. 2 hours). However, the iteration count is useful for extrapolation to more complex objective functions where the bulk of the time is taken up with the computation of the measurement of interference.

Throughout the experimental stage and the comparison of TT with tabu search, the set F was equal to $\{1, \dots, 50\}$.

In this section we describe how we chose the parameters for TT. In the next section we describe how we compared TT with an implementation of tabu search. We estimate that it would require 10^8 hours to consider all the possible combinations of the parameters, so clearly a strategy is necessary for finding good parameters which is less time consuming.

Initial experiments indicated that the choice of screening variant was the most significant parameter and that the technique appeared to be robust with respect to variation of the other parameters, the data sets, the objective functions and the termination criteria. Consequently, we fixed $b = 5$, $N = 252$, $L = N/4$, $U = N/2$, T_1 , and we varied K and the screening variant for 5 runs of 2,000 iterations each (see Table 1). As a result of this and the initial experiments, we concluded that DR was as good or better than any other screening variant but that further experiments were required to determine a good value of K .

Table 1: Average number of violations for 5 runs of each K value and each screening variant with 2,000 iterations.

$$N = 252, b = 5, L = N/4, U = N/2.$$

K	TT Method								
	BR	BB	BD	DR	DB	DD	RR	RB	RD
1	36.2	37.0	38.2	34.0	34.0	44.2	37.6	56.0	53.4
2	35.8	37.8	38.4	33.2	37.2	45.4	35.0	54.8	54.6
3	33.0	37.6	37.8	31.2	37.6	43.6	32.6	56.0	55.4
4	31.4	37.0	38.0	31.4	37.0	45.2	34.0	54.8	54.6
5	32.4	38.0	37.8	32.2	38.0	44.4	32.0	56.0	55.4

The second experiment was to fix DR , $b = 5$, $N = 252$, $L = N/4$, $U = N/2$, T_1 and varied K for each data set for 5 runs. Each variation was run for 10,000 iterations and for 1 hour. The value of $K = 4$ and 7 gave the best results. Next we fixed DR , $N = 252$, $L = N/4$, $U = N/2$, T_1 and varied K and b for 5 runs of 10,000 iterations. There was little to choose between many combinations but $K = 7$, $b = 5$ was the best. Finally, we fixed DR , $b = 5$, $N = 252$, $K = 7$, T_1 and varied $(L+U)/2$ for 5 runs of 10,000 iterations each (see Table 2). The value $(L+U)/2 = 201 = 4N/5$ gave the best quality solutions, so we selected the values $L = 15N/20$ and $U = 17N/20$.

Table 2: Average number of violations, standard deviation and average time for each $(L+U)/2$ value for 5 runs of 10,000 iterations.

Screening variant DR , $N = 252$, $K = 7$, $b = 5$.

$(L+U)/2$	Average Interference	Standard Deviation	Average CPU time (secs)
25	27.8	2.28	1738.80
50	26.0	2.34	1246.92
75	27.2	2.38	1071.50
100	26.6	1.51	978.8
126	28.0	1.58	928.9
151	28.2	1.78	892.62
176	26.4	1.14	872.58
201	25.0	1.22	859.54
226	27.4	1.51	835.68
252	27.2	1.64	825.64

Hence, our final platform for the TT algorithm uses block size $b = 5$ and T_2 as the screening measure in the improving phase and uses $K = 7$, $L = 15N/20$, $U = 17N/20$ and random screening selection in the mixed phase. These values we apply to all test data using either global objective function and either termination criterion. We appreciate that individual problems may require further fine tuning of the parameters.

In the experiments using the screening variant DR , there was no

evidence of long-term cycling. We attribute this to the randomisation of the subsets within a block of the improving phase and the randomisation of all subsets at the start of each mixed phase.

5. Comparison of Tabu Thresholding with Tabu Search

With the parameters fixed as described in the previous section we were able to compare TT with tabu search (TS) on the frequency assignment problem. In Castelino, Hurley and Stephens (1995), the authors had found that TS was superior to both genetic algorithms and local steepest descent heuristic on the same test data, having done similar experiments to determine good values of the parameters.

For the sake of completeness, we describe briefly the implementation of TS with which TT was compared. The notion of neighbourhood and move are as described in section 3 for TT. A neighbour obtained by changing f_j is considered tabu if such a change has been made in the last r iterations (recency) or if the proportion of changes over the whole history of the search exceeds a value s (frequency). The aspiration criterion is to select a tabu move if it generates a solution better than any previously encountered. The values of the parameters are $r = 2N/5$ and $s = 7/(4N)$. Like TT, the technique TS is robust with respect to small changes in these parameters, so precise values are not critical. No screening for TS is employed.

For each scenario, and for each objective function T_1, T_2 , we applied TT and TS five times for 1 hour each and also for longer varying times. The algorithms were coded in C and run on a SPARC-Server 10/51 workstation (27.3 Mflops). The results of the comparative tests are summarised in Table 3. They indicate clearly that tabu thresholding outperforms tabu search in almost all the experiments and that, for this problem, tabu thresholding is the preferred technique. In particular, in 48 comparisons of the best and average in Table 3, only three times is TS better than TT and then only marginally so.

6. Summary and Conclusions

In this paper we develop several variants of a tabu thresholding algorithm for solving frequency assignment problems. These in-

Table 3: Best, Average, Standard Deviation (SD) and Average Percentage Decrease (APD) of 5 runs of 60 minutes (first table) and longer (second table).

N			252	282	410	450	490	726	
Best	T_1	TT	23	118	374	184	635	1575	
		TS	25	141	405	190	724	2160	
	T_2	TT	148	445	1418	779	2346	5481	
		TS	145	476	1482	843	2534	6319	
Ave	T_1	TT	24.4	122.6	383.0	186.0	653.8	1611.8	
		TS	26.2	144.0	427.5	197.2	746.6	2226.4	
	T_2	TT	150.4	458.6	1423.6	801.0	2369.8	5519.4	
		TS	149.8	458.4	1517.6	852.6	2577.0	6355.4	
SD	T_1	TT	0.89	2.96	7.38	1.58	13.21	22.44	
		TS	1.09	3.00	12.86	5.63	14.69	45.54	
	T_2	TT	1.81	9.28	5.22	14.83	22.09	22.45	
		TS	3.27	7.73	22.94	7.89	27.47	22.99	
APD	T_1	TT	93.87	85.05	78.21	85.99	75.21	69.82	
		TS	93.41	82.66	75.70	85.31	71.69	58.86	
	T_2	TT	85.70	75.63	62.11	73.92	55.48	49.59	
		TS	85.78	73.33	59.16	72.88	52.44	41.05	
N			252	282	410	450	490	726	
time (mins)			(300)	(100)	(153)	(176)	(222)	(475)	
Best	T_1	TT	22	118	364	179	618	1499	
		TS	22	128	379	176	658	1535	
	T_2	TT	143	448	1370	774	2329	5187	
		TS	146	480	1497	815	2455	5530	
Ave	T_1	TT	22.6	121.4	371.2	183.4	624.2	1523.6	
		TS	23.8	134.0	400.8	188.0	678.9	1602.8	
	T_2	TT	144.8	456.8	1386.4	785.4	2034.4	5268.8	
		TS	148.0	491.6	1471.2	831.8	2492.8	5575.4	
SD	T_1	TT	0.54	2.60	6.61	3.21	4.91	26.26	
		TS	1.09	3.71	10.86	6.50	16.15	31.33	
	T_2	TT	1.92	6.06	14.88	7.77	25.35	57.76	
		TS	1.41	1.43	17.05	14.34	32.17	50.90	
APD	T_1	TT	94.32	85.08	78.88	86.35	76.33	71.75	
		TS	94.02	83.78	76.94	85.89	74.33	70.72	
	T_2	TT	85.88	75.59	62.77	73.93	57.28	52.08	
		TS	85.75	73.73	59.94	72.41	54.04	48.86	

corporate several features of TS such as intensification and diversification strategies to guide the search instead of using memory structures. A computational study on a set of frequency assignment problems is studied to assess whether the more difficult the problem the more sophisticated the TT algorithm needs to be. We investigate the advantage of several screening measures with respect to saving computation time. Our results show that TT works well for frequency assignment problems and provides sufficient diversity without the need for other diversification strategies. On the whole, we believe this preliminary investigation of TT deserves fine tuning and further study to enhance the performance. It has the advantage of dispensing with recency and frequency parameters and the experimental work required to determine good values for them. It requires less memory and appears to give better quality solutions for all the problem scenarios. Future work to improve its performance will be to investigate new strategies for the improving and mixed phases; for example, it would be worthwhile to study the ideas of Glover (1992) and use probabilistic best solutions to increase the speed of the technique. We will also investigate suitable techniques and parameters for the *updating assignment problem* where a small amount of the data is modified. These problems need to be solved more speedily than the bulk assignment problem.

7. Acknowledgements

We express our gratitude to Fred Glover for his helpful suggestions. We thank Steve Bracking, Ray Bradbeer and Major Rick Barford of the Defence Research Agency at Malvern and Dr David Tamarind and Captain Mike Griffiths of the Army EMC Agency at Blandford for discussions which enabled the generation of realistic test data for Combat Net Radio assignment scenarios. We thank Steve Hurley, Department of Computing Mathematics, Cardiff for the use of the computer generated data based on these discussions. We thank the referees for helpful comments in suggesting improvements to the presentation.

8. References

- D.J. Castelino, S. Hurley and N.M. Stephens, Solving frequency assignment problems using tabu search. To appear *Annals of Operations Research* (1995).

- K. Chiba, F. Takahata and M. Nohara, Theory and performance of frequency assignment schemes for carriers with different bandwidths under demand assignment SCPC/FDMA operation. *IE-ICE Trans. Commun.*, Vol. E75-B, No. 6, (1992)476-486.
- D. Costa, On the use of some known methods for t-colourings of graphs, *Annals of Operations Research*, 41(1993)343-358.
- W. Crompton, S. Hurley and N.M. Stephens, A parallel genetic algorithm for frequency assignment problems, *Proceedings of the IMACS/IEEE Conference on Signal Processing, Robotics and Neural Networks*, (Lille, France, 1994) p. 81-84,
- K.A. Dowsland, Simple tabu thresholding and the pallet loading problem. European Business Management School, Swansea University, Swansea (1995).
- M. Duque-Antòn, D. Kunz and B. Rüber, Channel assignment for cellular radio using simulated annealing, *IEEE Transactions on Vehicular Technology*, 42(1993)14-20.
- F. Glover, D. Karney, D. Klingman and A. Napier, A computational study on start procedures, basis change criteria and solution algorithms for transportation problems, *Management Science*, 20(1974)793-813.
- F. Glover, Heuristics for integer programming using surrogate constraints, *Decision Sciences*, 8(1977)156-166.
- F. Glover, Future paths for integer programming and links to artificial intelligence, *Computers and Operations Research*, 13(1986) 543-549.
- F. Glover, Tabu search - Part 1, *Orsa Journal on Computing*, 1(1989) 190-206.
- F. Glover, Tabu search - Part 2, *Orsa Journal on Computing*, 2(1990) 4-32.
- F. Glover, Tabu thresholding : improved search trajectories by non-monotonic search trajectories *Orsa Journal on Computing* to appear (1995).
- W.K. Hale, Frequency assignment: theory and applications, *Proceedings of the IEEE*, 68(1980)1497-1514.
- R. Mathar and J. Mattfeldt, Channel assignment in cellular radio networks, *IEEE Transactions on Vehicular Technology*, 42(1993) 647-656.
- B.H. Metzger, Spectrum management technique, presented at 38th National ORSA Meeting (Detroit, MI), 1970.
- A. Raychaudhuri, Optimal multiple interval assignments in fre-

- quency assignment and traffic phasing, *Discrete Applied Mathematics*, 40(1992)319-332.
- V. Valls, R. Marti and P. Lino (1993) A tabu thresholding algorithm for arc crossing optimization in biparite graphs. Internal Report. Departamento de Estadistica e Investigacion Operativa. Universidad de Valencia, Spain.
- D. de Werra and Y. Gay, Chromatic scheduling and frequency assignment, *Discrete Applied Mathematics*, 49(1994)165-174.

A New Tabu Search Approach to the 0–1 Equicut Problem

Mauro Dell'Amico

Francesco Maffioli

Dipartimento di Elettronica e Informazione

Politecnico di Milano

P.zza L. da Vinci, 32, 20133 Milano (ITALY)

E-mail: dellamic@elet.polimi.it

Abstract:

Given an undirected graph, the 0–1 equicut problem consists of finding a partition of the vertex set into two subsets of equal size, such that the number of edges going from one subset to the other is minimized. A classical heuristics for this problem was presented 25 years ago, whereas simulated annealing, genetic algorithms, tabu search and a greedy randomized procedure have been developed in the last 5 years. In this paper we present a new tabu search algorithm and show, thorough extensive computational experiments, that in most cases it beats the other methods.

Key words: Network design, equicut, tabu search, metaheuristics.

1. Introduction

We consider a classical problem in graph theory: the *equicut* problem, which consists in finding a partition of the vertex set of a given undirected graph into two *shores* of equal size, such that the sum of the weights of the edges from one shore to the other is minimized. This problem has great relevance in VLSI design since it models the problem of optimally placing “standard cells” in order to minimize the “routing area” required to connect the cells (Dunlop and Kernighan, 1985). Moreover the equicut models the problem of minimizing the number of holes on a circuit board, subject to pin preassignment and layer preferences and has also applications in Physics, where it models the problem of finding the

ground state magnetization of spin glasses having zero magnetic field (Barahona, Grötschel, Jünger and Reinelt, 1988). To be more precise let $G = (V, E)$ be an undirected graph with vertex set V , edge set $E = V \times V$ and weights $w_e \in Z^+$, $e \in E$ (for each edge $e = (u, v)$, with $u, v \in V$, we will write $w(u, v)$ instead of $w_{(u,v)}$). Moreover let $\overline{E} = \{e \in E : w_e > 0\}$ be the set of edges with positive weight. For any proper subset $S \subset V$ a *cut* associated with S is $\delta(S) = \{(u, v) \in E : u \in S, v \in V \setminus S\}$. Let $n = |V|$: a cut is an *equicut* if $|S| = \lfloor \frac{n}{2} \rfloor$. The *equicut problem* is to find the equicut $\delta(S)$ which minimizes the cut weight

$$z(S) = \sum_{e \in \delta(S)} w_e$$

It is immediate to see that one can always consider instances with an even number of vertices, indeed, if n is odd, we can simply add one vertex with all edges connected to this node having zero weight. (Therefore, in the following, we consider graphs with an even number of vertices.) The equicut problem is known to be \mathcal{NP} -hard (see e.g. Garey and Johnson, 1979), even if $w_e \in \{0, 1\}$, $e \in E$. In this case we call it the *0-1 equicut problem*.

In the following section we briefly review the literature describing the algorithms presented for solving the problem. In section 3 we introduce a basic tabu search approach for the 0-1 equicut problem, whereas in the next section 4 we improve our basic procedure introducing sophisticated techniques. The effectiveness of the resulting algorithm is proved by direct comparison with a GRASP approach, on 3.400 randomly generated instances.

2. Results from the literature

The classical heuristic for the equicut problem is due to Kernighan and Lin (1970) (*KL* in the following). It starts with a randomly generated partition of the vertices and iteratively improves it as follows. Consider a generic iteration and let A and B be the two shores of the actual partition (i.e. $A \subset V, B \subset V, |A| = |B|, A \cup B = V, A \cap B = \emptyset$). For each vertex $a \in A$ let $E_a = \sum_{j \in B} w(a, j)$ be the *external cost* and $I_a = \sum_{j \in A} w(a, j)$ be the *internal cost* (similarly, for a vertex $b \in B$, define $E_b = \sum_{j \in A} w(b, j)$ and $I_b = \sum_{j \in B} w(b, j)$). The *difference* of the external and internal costs $D_j = E_j - I_j$ is the variation of the cut weight if vertex j is moved from

one shore to the other. If a vertex $a \in A$ and a vertex $b \in B$ are *exchanged* (i.e. a is moved to set B and b to A), the new partition still defines an equicut and its objective function value is $D_a + D_b - 2w(a, b)$ unit less than that of the previous equicut. The algorithm determines a subset of $k \leq n/2$ vertices from set A to be exchanged with k vertices from B as follows. First determine $a(1) \in A$ and $b(1) \in B$ such that the gain $g(1) = D_{a(1)} + D_{b(1)} - 2w(a(1), b(1))$ is a maximum among all possible pairs a, b . Then vertices $a(1)$ and $b(1)$ are exchanged, the values D_i ($i \in V$) are updated and $a(1), b(1)$ are removed from sets A and B , respectively. A second pair of vertices $a(2) \in A, b(2) \in B$ for which $g(2) = D_{a(2)} + D_{b(2)} - 2w(a(2), b(2))$ is a maximum, is identified. This procedure is repeated $n/2$ times to compute $g(1), \dots, g(n/2)$ and finally the value k which maximizes $G = \sum_{i=1}^k g(i)$ is determined. If $G > 0$ the set of vertices $\{a(1), \dots, a(k)\}$ from set A are exchanged with the set $\{b(1), \dots, b(k)\}$ from B and a new iteration is performed. If $G \leq 0$ the procedure terminates.

A first approach to the equicut problem using metaheuristics was presented in Kirkpatrick, Gelatt and Vecchi (1983) and in Kirkpatrick (1984), where the *simulated annealing* (*SA*) technique was considered. The computational results presented were limited and a not efficient implementation of the Kernighan-Lin algorithm was used as a competitor.

Johnson et al. (1989) presented a deep and extensive study of the simulated annealing approach to the 0–1 equicut problem. They start with a partition $(S, V \setminus S)$ of the vertices, which not necessarily determines an equicut (i.e. $|S|$ may be different from $n/2$). For each $v \in S$ the new partition $(S \setminus \{v\}, V \setminus S \cup \{v\})$ is said to be a *neighbor* of $(S, V \setminus S)$. In order to consider the lack of balance of a partition, the cost of the associated cut is increased by a quadratic factor of the size of the imbalance. For a given cut $\delta(S)$ the cost is then $z'(S) = |\delta(S)| + \alpha(|S| - |V \setminus S|)^2$ where α is an appropriate *imbalance factor*. Using this neighborhood and the evaluation function $z'(S)$ they experimented different techniques to generate starting solutions, various cooling strategies and criteria to select the next neighboring solution. Finally they determined, through extensive computational experiments, the best parameter setting. The graphs used in the test problems were of two kinds: (a) *ran-*

dom graph $G_{n,p}$, were n is the number of vertices and $0 < p < 1$ is the probability that the edge between a given pair of vertices has weight 1; (b) *geometric graph* $U_{n,d}$, generated drawing from an uniform distribution n point in an unit square, associating a vertex to each point and giving weight 1 to each edge (u, v) iff the euclidean distance between the two points u and v is less or equal to d . The expected average vertex degree is $\hat{\nu}$ is equal to $p(n - 1)$ for the random graphs, whereas it is approximately $n\pi d^2$ for the geometric graphs. A comparison with the *KL* algorithm show that simulated annealing beats the traditional heuristic for large random graphs ($n = 250$ and $\hat{\nu} \geq 20$ or $n \geq 500$) even when running time is taken into account. It is substantially outperformed for geometric graphs.

Recently Laguna, Feo and Elrod (1994) have presented a *Greedy Randomized Adaptive Search Procedure (GRASP)* which, compares well with the *KL* algorithm, both for random and geometric graphs, when the same running time is given to the two algorithms. Furthermore the behaviour of *GRASP* considerably improves when larger computational times are allowed and it finds the optimal solution in many geometric graphs within 30 seconds on a DECstation 5000/120. The main body of *GRASP* consists of two phases: (i) identify an initial solution through a greedy randomized algorithm; (ii) improve this solution using a local search procedure. Since phase (i) is randomized the two phases can be repeated to look for new solutions. Phase (i) starts with an empty partition $A = B = \emptyset$, than selects one vertex at a time to be added to set A or B , alternatively. For each unselected vertex i the difference D_i is computed with respect to the current sets A and B , and a vertex is randomly selected among the first k elements with smallest value D_i (were k is a fixed parameter). For the improving phase the authors have considered local search procedure based on the exchange of two vertices (one from set A and one from set B). Among different strategies they have chosen the *slightest swap* technique which search, among all possible pairs, the first exchange with minimum positive gain (i.e. a pair a, b , with $a \in A, b \in B$ such that $g = D_a + D_b - 2w(a, b) = 1$.) Basing themselves on a wide computational experience on random and geometric graphs the authors conclude that: “*GRASP* compares well with the average *KL* solution when both methods employ the same computational effort. The the average solution quality considerably improves with a modest increase in computational

effort". Therefore *GRASP* can be considered the most effective technique with respect to *KL* and *SA*.

For the equicut problem with generic weights a genetic algorithm was presented in Rolland and Pirkul (1992) and it was improved in Pirkul and Rolland (1994). They report results on dense and sparse instances: the sparse instances have up to 80 vertices with vertex degree from 20 to 34 and computational times up to 7000 seconds on a Prime 9955 minicomputer. Following Dongarra (1995) the speed of the DECstation 5000/120 used by Laguna, Feo and Elrod (1994) is no more than seven times that of a Prime 9955. Since the *GRASP* approach takes about 30 seconds to find near optimal solutions on sparse graphs with 500 vertices and vertex degree from 5 to 80, we conclude that this genetic algorithms, developed for the general equicut, are not appropriate for 0–1 equicut.

A tabu search approach for the general equicut problem has been presented by Rolland, Pirkul and Glover (1995). They start by randomly generating a feasible solution with $|S| = |V \setminus S|$. Then a tabu search technique is applied, which uses a restricted version of the neighborhood introduced in Johnson et al. (1989). A neighboring solution is obtained from the current one by moving a single node v from one shore to the other, according to the following two rules: (a) if the absolute value of the imbalance $|S| - |V \setminus S|$ is less or equal to a specified value *ImBalanceFactor*, then chose v indifferently from S or from $V \setminus S$; (b) if otherwise the imbalance is greater than *ImBalanceFactor*, then chose v from the largest shore. Among the not-tabu solutions of the neighborhood they choose the one which gives a maximum improvement of the objective function value. The value *ImBalanceFactor*, initially zero, is dynamically updated so as to obtain a *strategic oscillation* (see Glover and Laguna, 1993). The resulting algorithm *TS* has been tested on a set of 144 dense and sparse graphs with $n \leq 80$ and different weighting functions. Our computational experiments made with a C version of *TS* obtained with a one-to-one translation from the original Pascal code implemented by the authors, show that *TS*, for the 0–1 case, is beaten by *GRASP*, by our simple tabu search algorithm *BasicTS* (see next section) and also, for many instances, by the *KL* heuristic. So *TS* has not to be considered as a competitor for the 0–1 instances.

For the exact solution of equicut problems a branch-and-cut algorithm was developed by Brunetta, Conforti and Rinaldi (1994), basically using the theory developed in Conforti, Rao and Sassano (1990). They solved to the optimum various kind of instances with up to 60 vertices, within 127000 seconds on a SUN SPARC 10/41 workstation.

3. A basic tabu search approach

In this section we introduce a new tabu search algorithm for the 0–1 equicut problem and compare it with the *GRASP* approach which actually is the most effective algorithm for the 0–1 case (see previous section). We have carefully implemented, in C language, the *GRASP* and the *KL* heuristics as described by Laguna, Feo and Elrod (1994) and by Johnson et al. (1989). We run our computational experiments on a PC 486 DX2/66 with MS-DOS operating system and DOS/4GW extender. From Dongarra we have that our computer and the DECstation 500/120 used by Laguna, Feo and Elrod (1994), have similar speeds. To be sure that our implementation of *GRASP* is correct we made some comparisons between our results and the ones presented by the authors.

Table 1: Comparison of different implementations of *GRASP* for $G_{n,p}$; running times for 1 iteration.

n	impl.	$\hat{\nu}$					
		2.5	5	10	20	40	80
124	L.F.E.	0.013	0.014	0.015	0.018	0.024	0.037
	DA.MF.	0.012	0.011	0.016	0.016	0.024	0.032
250	L.F.E.	0.051	0.055	0.057	0.064	0.080	0.112
	DA.MF.	0.047	0.051	0.058	0.064	0.075	0.107
500	L.F.E.	0.200	0.202	0.215	0.233	0.270	0.345
	DA.MF.	0.195	0.207	0.212	0.235	0.283	0.352
1000	L.F.E.	0.790	0.800	0.826	0.870	0.970	1.168
	DA.MF.	0.761	0.826	0.858	0.983	1.153	1.195

In Table 1 we report the running times for 1 iteration of *GRASP* on random graphs $G_{n,p}$ with $n = (124, 250, 500, 1000)$ and average vertex degree $\hat{\nu} = (2.5, 5, 10, 20, 40, 80)$. In rows labeled L.F.E. we report the computing times obtained from Table II of Laguna, Feo and Elrod (1994), dividing the “Total time”, by the “Total itera-

tions” value. In rows labeled DA.MF. we report the average computing times for one iteration of our implementation of *GRASP*, obtained from 20 instances, and 10 *GRASP* iterations per instance. The running times of our implementation and that from the original authors are very similar. We also checked the average performances of *GRASP* versus *KL* for random graphs with 500 vertices (G_{500}), by generating 100 random instances for each value of $\hat{\nu}$ and running our implementation of *GRASP* (*GR.* in the tables) for the same time used by *KL*. In Table 2 we give the number of best solution found (BT), the average objective function value (AV) and the average standard deviation (SD). If the sum of the BT values of *GRASP* and *KL* it is different from 100, it means that the two algorithms have found the same solution value, for some instances. In rows labeled L.F.E. we report the values of Table I of Laguna, Feo and Elrod (1994); in rows labeled DA.MF. we report our results.

Table 2: Comparison of average performances values for different implementations of *GRASP* for G_{500}

impl.		$\hat{\nu}$											
		2.5		5		10		20		40		80	
		<i>GR.</i>	<i>KL</i>	<i>GR.</i>	<i>KL</i>	<i>GR.</i>	<i>KL</i>	<i>GR.</i>	<i>KL</i>	<i>GR.</i>	<i>KL</i>	<i>GR.</i>	<i>KL</i>
L.F.E.	BT	75	23	52	45	34	62	50	47	65	34	74	24
	AV	65	70	253	255	714	711	1730	1729	3922	3933	8524	8543
	SD	9	7	13	15	27	24	29	24	44	47	60	68
DA.MF.	BT	69	24	53	43	35	63	46	42	60	38	70	27
	AV	62	69	252	254	710	706	1724	1717	3911	3916	8512	8526
	SD	8	9	12	14	19	20	29	28	44	48	60	66

From Tables 1 and 2 we have that our implementation of *GRASP* is equivalent to that of Laguna, Feo and Elrod (1994).

Our basic tabu search (*BasicTS*) considers as starting solution the first partition obtained by Phase (i) of *GRASP*, than improves it, using a standard tabu search technique. Figure 1 gives the details of the algorithm (see e.g. Laguna and Glover, (1993) for a recent survey on tabu search techniques). We use a neighborhood different from that of Rolland, Pirkul and Glover (1995). We consider as neighboring all partitions obtained from the current one by exchanging two vertices (a, b) , with $a \in A$ and $b \in B$. Our experience with tabu search algorithms (Dell'Amico and Trubian, 1993,

Dell'Amico, 1995) shows that good performances can be obtained when the neighborhood is accurately explored to identify the most promising solutions.

Algorithm *BasicTS*(s^* , *max_iter*)

input: s^* = a feasible solution;

max_iter = maximum number of iterations

output: s^* = improved solution

tabu_list := \emptyset ; *num_iter* := 0; $z^* := z(s^*)$; $s = s^*$;

while (*num_iter* < *max_iter*)

find $\hat{s} : z(\hat{s}) = \min\{z(s') : s' \text{ is a neighboring solution of } s, s' \notin \text{tabu_list}\}$;

if $z(\hat{s}) < z^*$ **then** $z^* := z(\hat{s})$; $s^* := s$;

tabu_list := *tabu_list* $\cup \{\hat{s}\}$; *num_iter* := *num_iter* + 1; $s := \hat{s}$;

end while

Figure 1: Algoritm *BasicTS* implements a standard tabu search.

So, at each iteration, we considered all pairs (a, b) , compute the corresponding gain $g = D_a + D_b - 2w(a, b)$ and choose the not-tabu pair with largest gain. Since the *move* which leads from a solution s to a new solution s' is the exchange of a pair of vertices (a, b) , it seems obvious to consider as a tabu the move (b, a) . However a straightforward tabu list based only on this idea does not avoid to return quickly in an already examined solution (see figure 2).

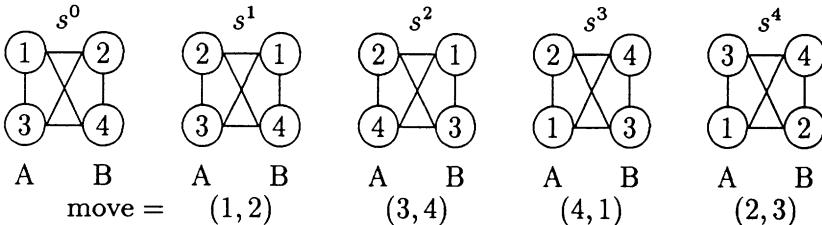


Figure 2: possible loop when the straightforward tabu list is used.

For this reason we introduced a second tabu list which consists of two sets: *tabu_A* and *tabu_B*. When a pair (a, b) with $a \in A$ and $b \in B$ is exchanged we put a in *tabu_A* and b in *tabu_B*. All the exchanges which move vertex a to set A (or vertex b to set B) are forbidden if $a \in \text{tabu}_A$ ($b \in \text{tabu}_B$). The same fixed length *tabu_tenure* was used for both tabu lists (i.e. a vertex remains in

the tabu list for *tabu_tenure* iterations). The first list can be stored using a square ($n \times n$) integer matrix (see e.g. Laguna and Glover, 1993), whereas the second one requires only two integer vectors of length n . We fixed the value of *tabu_tenure* to 10, basing ourselves on a set of 1000 preliminary experiments.

In order to test algorithm *BasicTS* we have generated random graphs with $n = (124, 250, 500, 1000)$ and average vertex degree $\hat{\nu} = (2.5, 10, 20, 40, 80)$, and geometric graphs with $n = (500, 1000)$ and average vertex degree $\hat{\nu} = (5, 10, 20, 40, 80)$. For each pair $(n, \hat{\nu})$ 100 random instances were generated and solved running *GRASP* and *BasicTS* for 60 seconds, each one. Tables 3 and 4 report the same performance indices of table 2 and the average time spent to find the best solution (Tbt). Algorithm *BasicTS* is denoted as *BTS*. *BasicTS* compares very favourably with *GRASP* for the random graphs, indeed it beats *GRASP* for the instances with $\hat{\nu} \geq 10$ *GRASP* remains the best algorithm for very sparse graphs ($\hat{\nu} = 2.5$).

Table 3: *GRASP* versus *BasicTS* using $G_{n,p}$ graphs.
Sixty seconds run on a PC 486 DX2/66; averages over 100 instances.

n		$\hat{\nu}$									
		2.5		10		20		40		80	
		<i>GR.</i>	<i>BTS</i>	<i>GR.</i>	<i>BTS</i>	<i>GR.</i>	<i>BTS</i>	<i>GR.</i>	<i>BTS</i>	<i>GR.</i>	<i>BTS</i>
124	BT	10	40	0	15	1	5	0	14	2	20
	AV	13.8	13.4	171.6	171.5	435.8	435.6	1010.7	1010.6	2248.6	2248.1
	SD	2.96	3.20	8.18	8.11	17.48	17.42	16.43	16.41	24.88	24.98
	Tbt	1.96	0.93	5.43	1.48	1.92	1.83	3.90	1.83	8.07	2.12
250	BT	80	9	0	98	0	100	3	80	0	82
	AV	25.0	28.2	338.2	334.7	850.3	847.5	1947.4	1944.6	4307.4	4304.7
	SD	4.92	4.28	12.02	11.03	14.56	15.49	29.70	29.44	29.37	28.79
	Tbt	9.52	2.74	17.25	9.99	18.48	5.92	15.31	5.19	13.43	9.17
500	BT	89	10	0	98	0	99	1	98	0	99
	AV	55.6	61.8	693.0	681.5	1698.0	1685.4	3886.3	3864.8	8463.6	8439.9
	SD	7.84	7.19	14.76	14.40	14.89	15.14	27.48	30.42	58.42	54.66
	Tbt	9.83	7.03	16.35	15.07	14.02	12.02	16.72	11.60	16.28	15.45
1000	BT	70	8	11	80	0	96	0	100	0	100
	AV	113.5	118.3	1400.6	1390.1	3430.4	3376.6	7732.9	7674.6	16850.8	16798.5
	SD	7.88	9.50	17.59	22.50	47.67	47.45	54.88	52.16	83.07	94.63
	Tbt	12.70	5.14	18.65	17.12	10.92	14.80	17.64	17.49	14.94	20.43

For geometric graphs *BasicTS* win, with respect to index BT, only for $n = 500, \hat{\nu} \geq 40$ and $n = 1000, \hat{\nu} = 80$. Moreover it is less robust than *GRASP* (the standard deviations of *BasicTS* are larger than those of *GRASP*).

Table 4: *GRASP* versus *BasicTS* using $U_{n,d}$ graphs.
Sixty seconds run on a PC 486 DX2/66; averages over 100 instances.

n		$\hat{\nu}$									
		5		10		20		40			
		<i>GR. BTS</i>									
500	BT	76	5	67	18	37	33	11	47	26	48
	AV	3.5	6.6	33.9	40.2	153.3	160.9	511.2	510.8	1591.9	1593.2
	SD	1.94	3.75	9.81	12.34	30.28	40.72	84.28	86.08	192.29	199.04
	Tbt	1.34	0.96	2.06	0.89	1.81	0.86	1.60	0.49	1.90	0.46
1000	BT	67	4	58	23	55	38	54	27	24	51
	AV	6.9	9.6	53.3	62.7	216.5	233.6	731.2	791.5	2270.5	2296.2
	SD	4.13	5.62	14.93	25.15	33.25	48.72	100.18	138.79	254.26	315.98
	Tbt	2.49	1.60	2.84	2.31	2.96	2.59	3.06	1.98	2.75	1.70

4. An improved tabu search algorithm

Motivated by the computational results of previous section we have modified the *BasicTS* including many ideas and strategies to obtain an *Enhanced Tabu Search* algorithm (*EnTaS* for short). From the computational experience with *BasicTS* we have noted that this approach is no able to reach solutions with a structure of the partition very different from the starting one. In particular difficult instances are those which require a “90 degree rotation” of the partition (i.e. a change of $n/4$ nodes). An example of this instances is reported in figure 3. The partition with $S = (1, 2, 3, 7, 8, 9)$ has weight 6, other 35 partitions have the same weight (for example $S' = (3, 7, 8, 9, 11, 12)$), many others have a weight larger than 6 and only one has weight 0. So *BasicTS* tends to be captured by the local minima with value 6.

A key for obtaining good solutions is then *diversification*: all the techniques we will describe in the following have been introduced to this end.

Balanced greedy algorithm for the initial solution

We developed a new procedure to obtain good starting solution. Our *Randomized Balanced Greedy Algorithm* is as follows: we ran-

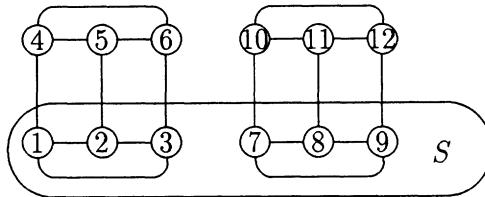


Figure 3: A difficult instance for *BasicTS*.

domly select a pair of “seed” vertices, one for set A and one for set B . Our guess is that the two seeds are in different shores, in an optimal solution. We increase the size of the two shores, up to $n/2$, by means of $n/2 - 1$ iterations. At each iteration we select, among all possible pairs of not yet selected vertices, the pair (a, b) which minimize the increment of the objective function value, if a is assigned to A and b to B . To increase the probability of starting from seeds which really belongs to different shores of an optimal solution we repeat the procedure two times fixing the seed for A and randomly selecting that for B . The better solution is chosen. It is not difficult to see that the probability that our guess is correct is about 75%.

Dynamic updating of the tabu list length

For the two tabu lists we used different lengths and updating strategies. Given the value *tabu_tenure* used for *BasicTS* we assign to the first tabu list (which inhibits the exchange of pairs of vertices) the fixed value $2 \cdot \text{tabu_tenure}$. The second tabu list has instead an initial length $\ell = \text{tabu_tenure}$ and this value is changed according to the evolution of the algorithm. When we are in an improving phase, in order to make the algorithm more aggressive we decrease the value of ℓ . On the contrary, in a not improving phase, the length is augmented to try to escape quickly from the last local minimum. In all cases the possible values of ℓ are limited to belong to an interval of length $2 \cdot \text{tabu_tenure}$. To be more precise we define a parameter Δt_l and: (a) if the last Δt_l moves have been improving we set $\ell = \max(\ell - 1, \text{tabu_tenure}/2)$; (b) if the last Δt_l moves have been non improving we set $\ell = \min(\ell + 1, \frac{3}{2}\text{tabu_tenure})$. With preliminary experiments on 1000 instances we have chosen the value 5 for Δt_l .

Move aspiration criteria by objective

We used two *move aspiration* criteria, i.e. conditions that eliminate the possible tabu status of a move, if satisfied. The first one is a simple global criterion which consists of accepting a move if determines a solution with objective function value better than the best solution found so far. The second criterion is based on local considerations. Let \hat{z} be the objective function value of solution \hat{s} considered immediately before the current solution s (i.e. s has been obtained from \hat{s} with a single move). If the objective function value of a solution $s' \in N(s)$ is less than $0.99 \cdot \hat{z}$, then s' is accepted without considering the tabu restrictions. This second criterion is particularly important to limit the excess of restrictions induced by the second tabu list.

Diversification by evaluated solutions

During the explorations of the various neighborhoods many solutions are evaluated, but only the best is chosen to continue the search. In the 0–1 equicut problems it is frequent to have many neighboring solutions with equal objective function value. During the search we store in a list L , ordered by nondecreasing solution values, the best solutions evaluated. When the current search selects nonimproving moves for *max_nonimproving* consecutive times then the actual solution is substituted by the best solution $s^* \in L$ and the search continues from s^* . Our idea is to restart exactly with the conditions (tabu list and other parameters) present when s^* was generated, but memory limitations prohibit to store the first tabu list (an $n \times n$ matrix). So we reconstruct exactly the second tabu list (two vectors of n elements) and other parameters, but not the first tabu list. The tabu status of all moves in the first tabu list than is removed. The length of list L was experimentally set to 5 whereas the value *max_nonimproving* was initially set to 10, if the average number of edges per vertex was less or equal to 20, otherwise it was set to 15. If the number of restarting from a solution of L became greater than 4% of n then *max_nonimproving* is increased by 5 units.

Restarting

Another possibility to examine different regions in the space of the solutions is to use the randomization of the initial solution. Our Balanced Greedy algorithm can produce, at most, $\lceil \frac{n-1}{2} \rceil$ different

solutions (we have $n - 1$ different seeds for set B and the procedure tries two seeds for each call). To be sure to generate all this solutions we substituted the random selection of the seed for B , by a randomly ordered list of the $n - 1$ vertices different from the seed of A . At each call the procedure chooses the first two elements of the list, removing them (the last call do not generate the second solution). When the tabu search has been restarted from a previously evaluated solution, from list L , for a number *max_previous* of times, the algorithm is reinitialized with a new call to Balanced Greedy. The value *max_previous* was experimentally set to the minimum between 5 and one half of the average number of edges per vertex.

The overall algorithm *EnTaS* was compared with *GRASP* using the same test problems described in the previous section. The results are reported in tables 5 and 6 where *EnTaS* is denoted by *ETS*. For the random graphs the tabu search approach results the winner for all classes, indeed *EnTaS* has a very much larger number of best solution found (BT) than *GRASP*, and its average solution values (AV) are always smaller than those of *GRASP*.

Table 5: *GRASP* versus *EnTaS* using $G_{n,p}$ graphs.
Five seconds run on a PC 486 DX2/66; averages over 100 instances.

n		$\hat{\nu}$									
		2.5		10		20		40		80	
		<i>GR.</i>	<i>ETS</i>	<i>GR.</i>	<i>ETS</i>	<i>GR.</i>	<i>ETS</i>	<i>GR.</i>	<i>ETS</i>	<i>GR.</i>	<i>ETS</i>
124	BT	0	61	0	33	1	26	9	21	17	25
	AV	14.0	13.1	172.6	172.1	436.8	436.4	1005.1	1005.0	2240.9	2240.8
	SD	3.03	2.91	8.87	8.62	16.74	16.70	23.24	23.50	28.92	28.39
	Tbt	1.01	0.28	1.65	0.38	1.62	0.77	1.75	1.81	2.01	1.81
250	BT	6	57	0	100	0	95	8	86	4	95
	AV	26.4	25.5	346.8	341.6	859.5	854.2	1965.8	1960.8	4323.8	4317.0
	SD	4.34	4.12	11.31	11.35	15.70	15.03	35.05	34.02	40.69	39.48
	Tbt	1.47	1.32	2.63	1.68	2.07	1.77	2.86	2.46	2.70	2.43
500	BT	18	81	0	100	4	95	3	96	2	97
	AV	60.5	57.9	703.7	686.1	1717.9	1697.0	3908.7	3880.3	8510.6	8475.1
	SD	7.04	6.92	16.12	14.79	25.28	25.15	41.60	39.46	61.16	64.87
	Tbt	2.54	2.24	2.64	2.67	2.85	3.09	2.15	3.29	2.42	3.36
1000	BT	28	65	4	95	3	96	0	97	8	90
	AV	118.8	116.4	1418.6	1400.7	3442.8	3402.5	7794.0	7733.4	16931.1	16846.1
	SD	8.54	8.50	23.22	23.14	39.44	36.72	68.39	55.98	85.58	96.31
	Tbt	2.47	3.09	2.84	3.12	2.90	3.06	2.93	3.70	3.01	3.60

For geometric graphs *EnTaS* beats *GRASP* for all $\hat{\nu} > 10$. The two algorithms are almost equivalents for $\hat{\nu} = 10$ whereas for $\hat{\nu} = 5$ the winner is *GRASP*. In particular *GRASP* seems to be able to find better solutions when the objective function value is very close to zero.

Table 6: *GRASP* versus *EnTaS* using $U_{n,d}$ graphs.
Five seconds run on a PC 486 DX2/66; averages over 100 instances.

n		$\hat{\nu}$							
		5		10		20		40	
		<i>GR. ETS</i>							
500	BT	48	35	40	38	4	57	0	63
	AV	3.5	4.0	33.9	34.5	153.3	141.4	511.2	489.6
	SD	1.94	2.01	9.81	7.57	30.28	24.76	84.28	70.96
	Tbt	1.34	1.30	2.06	0.93	1.81	0.68	1.60	0.49
1000	BT	63	29	39	53	24	67	0	74
	AV	6.9	8.0	53.3	51.2	216.5	202.5	731.2	694.3
	SD	4.13	3.32	14.93	13.03	33.25	30.00	100.16	64.64
	Tbt	2.48	1.45	2.84	2.02	2.94	2.85	3.06	1.57

To test the effectiveness of the algorithms in finding near optimal solutions we solved again the geometric instances giving 60 seconds of running time per algorithm. The results are reported in table 7.

Table 7: *GRASP* versus *EnTaS* using $U_{n,d}$ graphs.
Sixty seconds run on a PC 486 DX2/66; averages over 100 instances.

n		$\hat{\nu}$							
		5		10		20		40	
		<i>GR. ETS</i>							
500	BT	35	20	41	32	10	35	0	40
	AV	2.6	2.9	32.8	33.9	148.3	141.4	494.3	489.6
	SD	1.98	1.89	8.91	7.66	27.90	24.76	73.68	70.96
	Tbt	12.62	8.47	7.32	1.06	4.86	0.69	11.71	0.50
1000	BT	45	40	43	38	0	75	0	65
	AV	5.3	6.6	47.5	47.9	202.3	191.3	705.6	691.7
	SD	2.65	3.32	13.12	12.95	30.39	22.03	76.56	64.61
	Tbt	14.77	13.52	17.46	5.68	22.03	9.69	11.46	2.74

The algorithms present a relative behavior very similar to that

of table 6, however the solution's quality improves. Infact one can see that the average solution values are reduced, for all classes, with respect to table 6. Also the sum of the best solution found by *EnTaS* and *GRASP* is smaller, indicating that the two algorithms have found more solutions with identical solution value. The best solutions have been found, on average, within 22 seconds by *GRASP* and 14 by *EnTaS*; in general *EnTaS* determine its best solution before *GRASP*.

Finally we compared the performances of the two algorithms for long runs (5 minutes), on random and geometric graphs with $n = 500$ and different density. For each class and value of $\hat{\nu}$ ten instances were generated. From Table 8 one can see that *EnTaS* definitively beats *GRASP* on the random graphs. Instead for the geometric graph it is confirmed that *GRASP* is the winner for very sparse graph (see Table 9). The small values of index *BT* also show that the two algorithms find the same solution value for many instances on the geometric graphs.

Table 8: *GRASP* versus *EnTaS* using $G_{500,p}$ graphs.
Five minutes run on a PC 486 DX2/66; averages over 10 instances.

$\hat{\nu}$										
	5		10		20		40		80	
	GR.	ETS	GR.	ETS	GR.	ETS	GR.	ETS	GR.	ETS
BT	0	10	0	10	0	10	10	10	0	10
AV	242.0	232.4	690.0	675.4	1695.4	1681.8	3885.6	3868.0	8445.8	8435.40
SD	13.37	14.18	18.19	16.79	17.12	17.51	23.38	19.40	26.20	27.99
Tbt	166.45	145.58	134.27	72.10	37.01	98.61	129.92	90.42	169.52	106.31

Table 9: *GRASP* versus *EnTaS* using $U_{500,d}$ graphs.
Five minutes run on a PC 486 DX2/66; averages over 10 instances.

$\hat{\nu}$										
	5		10		20		40		80	
	GR.	ETS	GR.	ETS	GR.	ETS	GR.	ETS	GR.	ETS
BT	4	1	5	2	1	1	0	0	0	4
AV	2.0	2.8	28.8	37.2	139.4	141.8	492.4	492.4	1560.4	1558.2
SD	2.10	2.23	6.52	7.47	16.99	19.54	34.51	34.51	57.14	54.72
Tbt	5.13	7.59	3.60	1.20	3.95	0.65	59.57	0.58	47.64	0.93

5. Conclusions and future research

We have presented a new tabu search approach for the 0–1 equicut

problem, and we have studied its performances versus a *GRASP* approach from the literature, which actually is the most effective algorithm for the 0–1 case. We have tested the algorithms for equal length running times on random and geometric graphs. The computational results show that a simple version of tabu search beats the *GRASP* procedure on random graphs, with the exception of instances with average vertex degree less than 10. An enhanced tabu search algorithm *EnTaS* has been proposed which beats *GRASP* on all problems, except on the very sparse geometric graphs. From the literature we have that *GRASP* compares well with the classical Kernighan and Lin algorithm and with the simulated annealing approach, so our tabu search is one of the best algorithms to solve 0–1 equicut problems. Preliminary computational experiments with a tabu search procedure *TS* recently developed for the *general* equicut problem show that *TS* is not competitive for the 0–1 case. The comparison of *TS* with the other approaches on weighted instances is actually under investigation.

Acknowledgement

This work was partially supported by research contracts MPI 40% and 60% of the Italian Ministry of University and Scientific Research. We would like to thank Dr. Erik Rolland to make us available the Pascal version of algorithm *TS*.

6. References

- Barahona, F., M. Grötschel, M. Jünger, and G. Reinelt, An Application of Combinatorial Optimization to Statistical Physics and Circuit Layout Design. *Oper. Res.*, 36 (1988) p. 493
- Brunetta, L., M. Conforti and G. Rinaldi, A Branch-and-Cut Algorithm for The equicut Problem. Preprint n. 6, 14/02/1994, Dip. di Matematica Pura e Applicata, Università di Padova, (1994).
- Conforti, M., M.R. Rao and A. Sassano, The Equipartition Polytope I: formulation, Dimension and Basic Facets. *Math. Prog.* 49 (1990) p. 49.
- Conforti, M., M.R. Rao and A. Sassano, The Equipartition Polytope II: Valid Inequalities and Facets. *Math. Prog.*, 49 (1990) p. 71.
- Dell'Amico, M. and M. Trubian, Applying Tabu Search to the Job-Shop Scheduling Problem. *Annals of Operation Research*, 41 (1993) p. 231.

- Dell'Amico, M. Shop Problems with Two Machines and Time Lags. *Oper. Res.* to appear (1995).
- Dongarra J.J., Performances of Various Computers Using Standard Linear Equations Software. CS-89-85, Computer Science Dep., University of Tennessee, Knoxville (1995).
- Dunlop, A.E. and B.W. Kernighan, A Procedure for Placement of Standard-Cell VLSI Circuits. *IEEE Trans. on C.A.D.*, 1 (1985) p. 92.
- Garey, M.R. and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. (W.H. Freeman, San Francisco, 1979).
- Johnson, D.S., C.R. Aragon, L.A. McGeoch and C. Schevon, Optimization by Simulated Annealing: An Experimental Evaluation; Part I, Graph Partitioning. *Oper. Res.* 37 (1989) p. 865.
- Kernighan, B.W. and S. Lin, An Efficient Heuristic Procedure for Partitioning Graphs. *Bell Sys. Tech. J.*, 49 (1970) p. 291.
- Kirkpatrick, S., C.D. Gelatt, Jr. and M.P. Vecchi, Optimization by Simulated Annealing. *Science*, 220 (1983) p. 671.
- Kirkpatrick, S., Optimization by Simulated Annealing: Quantitative Studies. *J. Statis. Phys.*, 34 (1984) p. 975.
- Laguna, M., T.A. Feo and H.C. Elrod, A Greedy Randomized Adaptive Search Procedure for the Two-Partition Problem. *Oper. Res.*, 42 (1994) p. 677.
- Laguna, M., G. Glover, Tabu Search. In: *Modern Heuristic Techniques for Combinatorial Problems*, ed. C. R. Reeves (Blackwell Scientific Publications, Oxford, 1993).
- Pirkul, H. and E. Rolland, New Heuristic Solution Procedures for the Uniform Graph Partitioning Problem: Extensions and Evaluation. *Computers and Ops. Res.*, 21 (1994) p. 895.
- Rolland E. and H. Pirkul, Heuristic Solution Procedures for the Graph Partitioning Problem. In: *Computer Science and Operations Research: New Developments in Their Interfaces*. ed. O. Balci (Pergamon Press, Oxford, 1992).
- Rolland E., Pirkul H. and F. Glover, A Tabu Search for Graph Partitioning, *Annals of Operations Research*, to appear (1995).

Simple Tabu Thresholding and the Pallet Loading Problem.

Kathryn A. Dowsland.

Statistics and O.R. Research Group

E.B.M.S.

University of Wales Swansea

Swansea, SA2 8PP U.K.

E-mail: K.A.Dowsland@swansea.ac.uk

Abstract:

Rectangle packing problems are particularly challenging for local search methods as they frequently give rise to spiky solution landscapes. This paper is concerned with the development of a simple tabu thresholding algorithm for the pallet loading problem, with the ultimate objective of comparing it with a non-monotonic version of simulated annealing. Although the focus of the discussion is problem specific, some of the conclusions are relevant to the implementation of simple tabu thresholding in any problem domain.

Key Words: Tabu Search, pallet loading, tabu thresholding, simulated annealing.

1. Introduction.

A number of recent studies (Marques et al. (1991), Lutfiyya et al. (1992), Oliveira and Ferreira (1993) and Blazewicz et al (1993)) indicate a growing interest in the use of both simulated annealing (SA), and tabu search (TS), in the solution of cutting and packing problems. In real-life such problems frequently have the objective of finding a

feasible solution subject to a set of practical constraints. Even where this is not the case it is common to extend the solution space to allow layouts with overlapping pieces, and to penalise this in the cost function. Thus the satisficing problem of reducing such penalty costs to their optimal value of zero, thereby producing feasible solutions, is an important area of investigation.

The motivation for the work described in this paper is the apparent similarity between a non-monotonic variant of SA introduced in Dowsland (1993) and simple tabu thresholding (STT), a memoryless variant of tabu search suggested by Glover (1992). Dowsland (1993) describes some experiments with SA in the solution of two classes of rectangle packing problem: the pallet loading problem in which all the pieces to be packed are identical; and the two-dimensional strip packing problem involving rectangular pieces of different dimensions. Two of the main conclusions of that study were:

1. The usual monotonic cooling schedule was not able to guide the search towards optimal solutions. This led to the development of a non-monotonic versions of SA. (NMSA).
2. Although the results for the pallet loading problem suggested that the algorithm was only moderately successful, the experiments on this simple problem were invaluable in building a successful implementation for the more difficult strip packing problem.

It was conjectured that the failure of monotonic cooling was due to the spiky nature of the solution landscape, combined with the fact that only globally optimal solutions with zero cost were acceptable. The paper also concluded that tabu search might have been a more natural and successful approach under these conditions. However, the need for different memory functions would make this a more complicated procedure.

Recently Glover (1992) has introduced the idea of simple tabu thresholding, which is designed to mimic the behaviour of tabu search

without the use of memory functions. The technique has been used by Valls *et al.* (1993) on a graph drawing problem with encouraging results. This paper investigates the implementation of STT on the simpler of the two rectangle packing problems - that of pallet loading. As in Dowsland (1993) the aim of this research is *not* to develop a competitive heuristic for the pallet loading problem. Instead we use this relatively simple problem as the basis of some initial experiments with STT. The objectives are threefold: First to make some initial observations and decisions which can be transferred to an implementation of STT on more complex packing problems. Past experience with simulated annealing and other metaheuristics suggests that this will be a fruitful avenue of investigation. Second, to compare the performance of STT and NMSA on a simple, well understood problem. Third, to make any general observations which may be of use in other studies applying STT to satisficing problems or problems with spiky landscapes.

The next section starts with a formal definition of the problem and summarises the relevant details of the original SA study. This is followed by a brief outline of the simple tabu thresholding technique. As with other metaheuristic search techniques there are a number of decisions which must be made in order to implement STT in the solution of a particular problem, and much of the remainder of the paper is concerned with this decision making process.

Computer graphics have been used extensively throughout the study. Initial experiments involved watching the search progress from layout to layout on a small set of test problems. This enabled the shortcomings of different decisions to be identified and, where possible, rectified. A similar approach was taken in analysing the reasons for differing performances among the variants. Where possible these conclusions have been backed up by other data, but in some cases this evidence does not impart all the information gleaned from dynamic observation of the search process.

2. The pallet loading problem and the SA study.

The classical pallet loading problem is that of finding optimal arrangements of identical rectangles in a larger containing rectangle. Formally it can be expressed as follows: Given a containing rectangle (or pallet) of dimensions $X \times Y$, and a smaller rectangle (or box) of dimensions $a \times b$, maximise the number of copies of the smaller rectangle which can be accommodated in the larger rectangle, orthogonally (i.e. with their edges parallel to the edges of the containing rectangle) and without overlap. A fast exact algorithm is available for this problem (Dowsland (1987)). As the aim of this study was to experiment with SA on some satisficing problems from the field of cutting and packing, this information was used to define the simpler problem of fitting the known optimal number of rectangular pieces into the containing rectangle orthogonally without overlap. As with any SA implementation a number of decisions concerning both problem specific and generic factors had to be made. These were 'optimised' as the result of considerable experimentation which is fully described in Dowsland (1993). The options and parameters used in the final version are outlined below:

* The set of feasible solutions was defined as the set of layouts containing the required number of pieces. No account was taken of the overlap constraints. The size of the solution space was cut down by considering a reduced set of feasible positions for each piece. These correspond to placements which lie at an integral combination of lengths and widths from the nearest horizontal and vertical edges of the pallet. As any layout can be converted to one which satisfies these conditions, simply by sliding the pieces as far as possible to the nearest horizontal and vertical edges, at least one optimal solution exists within the solution space.

* The neighbourhood of any solution was originally defined as the set of solutions obtained by moving a single piece to a new position. However, empirical testing showed that solution times could be improved if only overlapping pieces were considered for moving.

- * The cost function was measured as the sum of the overlap between each pair of pieces. The objective was to reduce this to its minimum value of zero.
- * Starting solutions were generated by placing the correct number of pieces in random feasible positions, and the neighbourhoods were sampled using a uniform random distribution with replacement.

It has already been stated that the most interesting feature of the implementation was the 'cooling' schedule. Standard cooling functions and a constant temperature approach both proved unsatisfactory. Instead the temperature was set to cool down or heat up depending on whether the last move had been accepted or rejected respectively. After an accepted move the temperature was reduced according to the Lundy and Mees (1986) schedule i.e. $t \rightarrow t/(1+\beta t)$. After a rejected move the temperature was increased according to $t \rightarrow t/(1-\alpha t)$. The ratio $\alpha:\beta$ roughly determines the ratio of accepted to rejected moves. Experimentation suggested that a value of 0.02 for β and a ratio of 1:100 produced good results.

This implementation was tested on 3 data-sets each containing 100 problems with the number of pieces fitted in the ranges 5-10, 10-20 and 20-30 respectively. Experiments carried out on an IBM 80386 machine using Prospero FORTRAN showed high levels of optimality for the first two data-sets, but the results in the 20-30 range were less impressive and this was conjectured to mark the limits of the algorithm's capability. Nevertheless, further work showed that this framework was able to provide a firm foundation for a successful implementation for the more complex strip packing problem.

3. Simple tabu thresholding.

Like SA and TS, simple tabu thresholding is a local search method which avoids becoming trapped in an early local optimum by allowing some uphill moves. It therefore requires the basic ingredients of a well-defined solution space, neighbourhood structure and cost function. Full

details of the technique are presented in Glover (1992) and only a brief outline will be given here. The method works with two alternating phases - an improving phase and a mixed phase. The neighbourhood moves are partitioned into subsets, and only one subset is considered at each iteration. A 'best' move in the subset is selected as a candidate for acceptance. In the improving phase this move will be accepted only if it results in a cost function improvement. In the mixed phase the move is accepted, regardless of whether it decreases or increases the cost. The subsets are searched using a *block random order scan*. This is basically a systematic search in which each subset is allocated to a position on a cyclic list. The list is searched sequentially, starting again at the beginning, once one cycle has been completed. However, in order to avoid the repeated pattern of sampling implied by this process, the subsets are blocked into blocks of size k . Each block is shuffled on the list before its elements are due to be sampled. If there are m blocks, then as long as k is not a divisor of m , this will allow some migration of subsets along the list. If k is small compared to m then a subset will be sampled approximately once every m iterations, but the migration will mean that there is a possibility of any pair of subsets being sampled in succession. This effectively avoids cycling by simulating the effects of a tabu list of length approximately m . This is illustrated in Figure 1. Here we have 11 subsets and a block size of 3. Before sampling, the first three sets are blocked and shuffled into the order shown in the shaded portion. Once these three have been sampled the next block is shuffled, and so on, with a wrap around effect when the end of the list is reached.

1	2	3	4	5	6	7	8	9	10	11
2	1	3	4	5	6	7	8	9	10	11
2	1	3	6	4	5	7	8	9	10	11
2	1	3	6	4	5	7	9	8	10	11
11	1	3	6	4	5	7	9	8	2	10

Figure 1: Block Random Order Scan.

Thus at least 8 and at most 12 iterations must pass between subsequent samplings from the same subset, but the actual proximity of any two subsets on the list is constantly changing.

The improving phase is obviously an aggressive intensification phase, and is allowed to continue until all subsets have been sampled without an acceptance i.e. until a local optimum is reached. The method then switches to the mixed phase which is designed to diversify the search by allowing some uphill moves. The mixed phase continues for t iterations. It is suggested that t should be randomly generated between prespecified limits each time the mixed phase is invoked.

For a given problem it is obviously important to get the right balance between intensification and diversification, or equivalently to match the pressure for convergence to local optima with the ability to escape from them. This is partly provided by the parameter t , which controls the time spent in the mixed phase. It is also dependent on the size and number of subsets. Larger subsets will result in bigger downhill steps in the improving phase and smaller uphill steps in the mixed phase. Whatever size the subsets, the selection of the best move from each will tend to tip the balance in favour of intensification and may not provide enough flexibility to climb away from the valleys in the solution landscape. Glover (1992) suggests that, rather than choosing the best move from each subset, a probabilistic best move may be chosen. The idea here is to choose a move which may not be the absolute best in its subset, but is likely to be better than most. Glover suggests that a move should be chosen from the r -best moves in the subset, for some value of r , and outlines a number of methods to approximate this. Other ideas for changing the balance between intensification and diversification include using different subset sizes for the two phases and even replacing the mixed phase by a disimproving phase in which only uphill moves are accepted.

The above discussion implies that even if a problem has already been specified in terms of its solution space, cost function and neighbourhood structure, there are still a number of decisions to be made in the development of a successful STT solution method. As these decisions determine the balance between the ability to home in on good solutions, and the ease with which the search is able to climb over hills in the solution space, they need to be made with the topography of the solution landscape in mind. Perhaps the most important factor is the definition of the subsets which are used to partition the moves. Once this has been decided other factors can be considered. The next two sections are concerned with this decision making process for the pallet loading problem.

4. STT for pallet loading.

As the motivation of this study is to compare NMSA with STT, the problem specific decisions relating to solution space, neighbourhood structure, cost function and starting solution were not changed from those used in the SA implementation. In order to reduce the number of variable parameters in the initial investigation, the timing parameter, t , was set at a constant value of 5. A number of different ways of defining the subsets and other options were then developed and those which showed some initial promise were subjected to extensive experimentation. These are described in the following sections.

4.1 Subsets indexed by boxes.

The natural way to partition the set of neighbourhood moves is to define one subset for each box in the layout. Thus subset i will contain all possible moves for box i . As the solution space is defined by a considerably reduced set of feasible positions, the number of potential moves for each piece is relatively small. For problems with up to 30 boxes this is typically less than 250. For example, the problem of packing a 5×3 box onto a pallet of dimensions 19×18 has an optimal solution of 22 boxes and involves 136 feasible box positions. This would imply 22 subsets, each with 135 elements (a box is not

allowed to 'move' to its current position). If we restrict our experiments to problems with up to 30 boxes, this type of partition results in relatively few subsets, and the value of k must be correspondingly small. Following the suggestions of Glover a value of 5 was adopted except where the number of boxes (i.e. the number of subsets) is divisible by 5. For all other problems, apart from those with 30 subsets, a value of 6 was used and for 30 subsets k was set equal to 7. Implementations based on box indexed subsets will be denoted BI.

Initial experiments showed that these parameter settings produced a workable heuristic. However, a comparison of this method with the best simulated annealing implementation may not be strictly fair, as the SA experiments showed a significant improvement in performance when the set of feasible moves were limited to those involving boxes with overlap. In STT the improvement phase will be automatically restricted in this way, as moves involving boxes which are making no contribution to the cost function cannot result in its reduction. The mixed phase is not subject to any such restriction, and so an option to consider only those subsets indexed by overlapping boxes was added. This is in line with Glover's suggestion that screening for attractive moves can increase the efficiency of the search.

The graphical experiments displayed mixed results concerning the effectiveness of this option. As with the SA implementation it frequently results in faster solution times. This is because time is not wasted moving pieces one or two units in any direction and then moving them back in a later cycle. However, there were several instances where the search became trapped in a short cycle of moves close to local optima. The reason for this is that if only two or three pieces overlap, the tabu effect of the random order scan is diminished, as it corresponds to a very short tabu list. If the layout is such that there is a single best move for each piece, and this does not cause any new pieces to join the set of overlapping pieces, then cycling is inevitable. The obvious way to avoid this is to replace absolute best

with probabilistic best. As our aim was to avoid cycling, we did not adopt any of the methods suggested by Glover, but opted for the simpler option of sampling only $p\%$ of the current subset. This was applied only to the mixed phase. It was not used in the improving phase for two reasons. First, there are frequently a number of equal best moves in any subset. Thus even if the search samples from the same subset and the same current solution, it is possible for it to follow a different path and find new local optima. Second, the downwards pressure in choosing a best move in the improving phase was judged to be a positive factor in a spiky landscape.

The limited graphical experiments suggested that relatively high values of p (around 90%) are sufficient to avoid cycling. However, the value of p also has a secondary effect, in that the lower the value, the more likely the search is to take a relatively large uphill step in the mixed phase. Thus p determines the ease with which the search is able to climb over sizeable hills in the solution space, and it is possible that lower values of p will improve the chances of reaching a global optimum. It was therefore decided to try a range of values for p . Limited experiments were also carried out using the r -best strategy.

The graphical experiments also displayed a marked difference between problems involving almost square boxes and those involving longer, thinner boxes, with the latter appearing to be more difficult. Closer observation suggested that the search has difficulty in rotating thinner boxes. Figure 2 illustrates why this is so. Figure 2a concerns a problem with a box of dimensions 5×4 . The left hand layout is a local optimum. In this case the optimal solution can be reached in two moves, as shown, without any increase in the cost function. Figure 2b shows a similar situation with a 7×3 box. Here the obvious route to the optimal solution is to turn the two horizontal boxes, but the first such move will result in an increase of 6 units, as shown in the middle layout. Such a move is unlikely as there are several horizontal moves for each box which will result in smaller cost function increases.

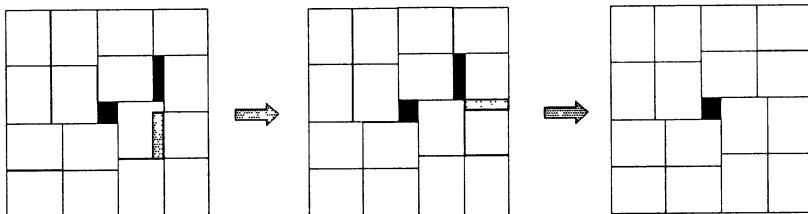


Figure 2a.

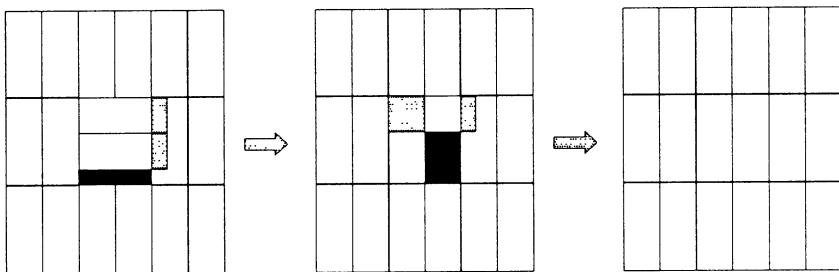


Figure 2b.



Waste.



Overlap.

Figure 2: Escaping local optima.

However, if moves are limited to those which use a vertical position then the required move is a least cost move. Therefore a second variant of the BI option was introduced. Here each subset is split into two, with one set containing the horizontal positions and the other the vertical positions for the box in question. The subsets will now be of slightly different sizes, but the differences are unlikely to be sufficient to cause significant problems of bias.

4.2 Subsets indexed by positions

An alternative natural partition is to index the subsets by position, and include in a particular subset moves of all boxes into that position. Initially this seemed less intuitively appealing, as the best move in many of these subsets will involve considerable overlap and is likely to lead to a large increase in cost. However, the improving phase will obviously ignore such positions, and will provide the necessary pressure for convergence. We have already observed that the shape of the landscape may make it desirable for the mixed phase to include a number of large uphill steps. This method of indexing may be better at providing this. As the number of subsets is now typically between 50 and 250 we again follow the recommendation of Glover and set $k = \min\{5, m/20\}$, where the number of subsets, m , is now equal to the number of positions. If k divides m its value is increased by one, until it reaches a suitable value. Implementations based on this type of partition will be denoted PI.

This representation does not lend itself to the option of screening moves in the mixed phase by considering only overlapping boxes. Close to local optima, when very few boxes overlap, this will effectively reduce the size of the subsets to two or three elements. Ideally we need a mechanism for removing whole subsets (i.e. positions) from consideration. This means restricting the search to positions which are in some way more promising. Intuitively positions covering gaps are more rewarding than those which are wholly occupied by other pieces. We therefore define our screening criterion for all PI implementations as that of using only moves which place boxes in positions that cover some of the unused pallet area. All moves in the improving phase will automatically satisfy this condition. As all overlap must be removed from an optimal layout, but some gaps may remain, this condition is not as strong as that used for the BI options.

Any differences between the quality of results produced by box defined subsets and position defined subsets may not be due to the

different structures, and may simply be a result of the different balance between the number and size of subsets. Therefore a second PI option was considered. Here groups of subsets are sampled together. The number of subsets in each group is chosen so that the number of moves in the group is as close as possible to the number in the BI partition for the same problem. For example, in the above problem with 22 boxes and 136 positions, 6 of the 136 subsets would be merged together for sampling, giving a total of 132 moves per selection. This is not quite the same as merging subsets permanently to form larger subsets as the groupings will be constantly changing. However it avoids the question as to whether position subsets should be merged randomly, or should be formed by contiguous regions on the pallet. Again several different values of p were considered to simulate different levels of probabilistic best.

4.3 Random subsets.

The final class of partitions involved allocating moves randomly. Two options were considered. In the first the number of subsets was set equal to the number of boxes and in the second it was set equal to the number of positions. In both cases the screening option in the mixed phase was based on overlapping boxes and a range of values for p were used to test probabilistic best. These implementations will be denoted RI.

5. The experiments

Once the small graphical experiments had identified that each of the above options was capable of solving problems in the range 10-30 boxes fitted, a set of more extensive experiments was designed. Their objectives were to compare the different variants of STT and to provide some initial indications as to how STT compares with NMSA. Two sets of 100 random problems were generated, the first in the range 10-20 boxes fitted and the second in the range 20-30 boxes fitted. All problems were attempted using all 6 subset partitioning methods described above, namely:

- BI1:** box indexed subsets, all positions in each subset.
- BI2 :** box indexed subsets, horizontal and vertical positions split.
- PI1:** position indexed subsets.
- PI2:** position indexed subsets, several merged for sampling.
- RI1:** random allocation of moves to subsets, number of subsets = number of boxes.
- RI2:** random allocation of moves to subsets, number of subsets = number of positions.

Each of the 6 was implemented with 8 variants, corresponding to probabilistic sampling with $p = 100\%$, 90% , 50% and 10% , with and without the relevant screening option. Each was run 5 times i.e. using 5 different random number streams, on each of the 200 problems. The variant of SA which proved most successful in the original study was also run 5 times on each problem.

As the overall objective is to find an optimal i.e. zero cost solution, the quality of sub-optimal solutions is not an important factor. Instead, we need to measure how successful each method is in reaching this target, and the efficiency with which it does so. Unlike standard SA neither NMSA or STT has a natural stopping point. We therefore have to determine a stopping criterion which will allow all the variants to be compared on equal terms. Three measures were considered: *time*, *number of evaluations* and *number of accepted moves*. *Time* was eliminated, primarily because at this stage we are more interested in the general behaviour of each method, rather than the efficiency with which moves can be evaluated. For example, there are a number of ways of screening out moves in the improving phase without a complete cost function evaluation. A fair comparison based on *time* would require that all computationally efficient screening options be implemented for all variants. *Number of evaluations* was eliminated for similar reasons. Smaller subsets and lower values of p will all lead to less evaluations per iteration. This issue is further complicated by the observation that it is sometimes possible to screen out whole subsets. We therefore chose to use *number of accepted moves*, while

recognising that it too has its drawbacks. In particular it is difficult to make a direct comparison with SA as the criteria for acceptance and rejection are completely different.

All the variants were programmed in FORTRAN and implemented on a DEC Alpha running at 60 mips. Each STT attempt was allowed up to 1000 acceptances, as experience gained during the graphical experiments suggested that convergence after this could be regarded as an almost random event. In order to get a very rough, initial comparison the NMSA attempts were limited by a *time* criterion. For each of the two data-sets the time taken for each attempt requiring the full 1000 moves was averaged over the total number of such attempts. NMSA was allowed to run for approximately this amount of time. This meant that NMSA was given a maximum of 2 seconds per attempt for the 10-20 data-set and 6 seconds for the 20-30 data-set. In addition to this full set of experiments, two further experiments were conducted. The first involved a variant of STT using the best set of parameters from the mainstream experiments, but allowing the value of t to vary randomly between 5 and 10. The second involved the r -best sampling method, and was run using the screened option and the best partitioning rule from the mainstream experiments. Random sampling from the set of r -best solutions was applied in both phases using $r = 5\%$, 10% and 20% of the subset size. Each option was implemented using 2 different sampling mechanisms. In the first each move was given a probability relating to its quality, using roulette wheel selection. In the second all of the r best moves were sampled with equal probability. None of these additional variants was able to improve over the best results in the mainstream experiments.

6. The results

The results were analyzed by recording for each variant the number of problems which were solved n times (for $n = 0$ to 5) within 100, 200, 500 and 1000 acceptances respectively. These values are reproduced in Table 1.

Table 1a: Percentage of problems solved optimally for dataset 10-20.

Solved	10% sampling					50% sampling					90% sampling					100% sampling				
	5	4	3	2	1	5	4	3	2	1	5	4	3	2	1	5	4	3	2	1
BI1 U	34	44	50	60	79	38	48	57	67	83	35	44	56	70	75	35	46	56	69	78
	44	57	63	73	90	51	62	70	80	90	44	52	64	75	84	46	54	67	77	86
	65	76	84	87	94	65	79	85	92	95	57	71	81	86	91	58	69	80	85	95
	76	84	88	94	97	80	90	93	96	100	70	85	88	91	98	76	81	87	91	97
BI1 S	42	53	63	77	88	51	66	79	85	93	52	64	73	84	93	43	60	72	83	93
	62	72	80	89	97	67	80	93	96	99	63	77	83	87	96	53	69	83	89	95
	78	88	93	97	99	88	94	96	98	100	77	84	86	92	98	64	79	87	93	98
	92	94	97	100	100	95	97	99	100	100	83	87	91	95	98	67	84	90	93	98
BI2 U	32	42	56	68	85	32	44	55	66	82	33	43	60	73	82	32	37	48	63	84
	49	58	71	80	94	43	58	69	76	86	47	58	72	82	91	39	47	61	75	91
	72	78	87	93	99	67	74	79	89	94	64	77	85	94	97	50	66	79	86	93
	80	91	95	97	99	77	84	89	95	97	76	86	94	96	97	53	71	84	87	94
BI2 S	39	56	64	74	87	41	56	66	76	91	42	57	71	80	91	40	53	68	80	90
	58	68	82	88	96	61	74	80	91	94	54	71	84	91	95	50	68	78	88	97
	78	85	92	95	98	84	89	90	95	97	69	85	93	95	98	57	77	85	94	97
	90	97	98	98	100	90	91	97	98	98	76	92	94	96	100	60	81	90	94	97
PI1 U	32	44	49	53	72	26	41	50	68	78	31	43	56	69	79	31	43	56	69	79
	44	50	61	72	88	40	58	67	80	92	45	56	71	76	90	45	56	71	76	90
	59	67	82	91	92	67	74	89	93	99	63	75	80	91	97	63	75	80	91	97
	69	81	89	93	96	77	87	92	97	99	77	85	89	97	100	77	85	89	97	100
PI1 S	33	46	54	69	84	37	46	62	73	89	37	46	61	71	86	37	46	61	71	86
	43	54	73	84	94	54	67	80	87	96	49	66	74	84	92	49	66	74	84	92
	67	77	88	90	99	78	88	95	97	99	75	86	90	93	98	75	86	90	93	98
	82	92	96	98	99	93	96	99	99	100	89	94	96	97	99	89	94	96	97	99
PI2 U	31	45	53	68	85	32	45	58	67	83	36	48	60	73	87	36	48	60	73	87
	49	59	73	83	94	50	61	72	84	93	53	61	76	87	96	53	61	76	87	96
	66	80	92	96	99	69	78	89	95	100	71	82	90	95	99	71	82	90	95	99
	94	96	99	100	100	82	91	95	100	100	81	89	94	98	100	81	89	94	98	100
PI2 S	39	50	58	73	89	40	52	65	79	92	40	54	62	75	89	40	54	62	75	89
	57	69	79	90	98	55	68	82	92	97	53	62	72	80	95	53	62	72	80	95
	80	90	95	99	100	76	89	96	99	100	73	78	80	89	97	73	78	80	89	97
	94	96	99	100	100	88	96	99	99	100	74	82	88	93	98	74	82	88	93	98
RI1 U	23	33	45	58	73	25	38	51	57	74	34	40	52	65	82	30	46	56	65	79
	34	52	57	69	86	40	56	65	76	87	46	56	67	82	92	46	60	69	79	88
	55	64	76	88	97	68	78	83	88	95	68	76	87	95	97	68	80	86	90	98
	67	78	88	95	100	77	88	92	94	99	82	88	98	98	99	81	85	91	96	99
RI1 S	26	38	46	56	78	26	40	51	64	78	26	47	57	68	84	35	43	52	65	82
	38	51	64	74	91	44	52	68	81	90	52	59	68	83	92	46	58	72	84	93
	56	69	82	89	94	64	73	85	92	96	69	80	87	97	99	68	83	87	94	97
	72	82	87	92	96	79	90	96	98	99	84	92	97	98	100	83	92	94	99	99
RI2 U	20	31	37	52	69	24	36	45	59	73	28	39	49	60	78	28	39	49	60	78
	32	45	55	66	80	41	51	63	68	86	39	52	58	73	87	39	52	58	73	87
	54	65	74	81	92	61	67	75	85	95	58	71	81	88	96	58	71	81	88	96
	64	76	84	89	96	69	78	86	95	98	73	84	89	93	98	73	84	89	93	98
RI2 S	28	43	47	54	71	30	36	46	61	74	20	36	51	61	75	20	36	51	61	75
	45	53	61	70	84	42	49	60	75	84	43	57	64	75	88	43	57	64	75	88
	59	69	80	86	94	59	69	80	92	95	66	76	82	89	98	66	76	82	89	98
	71	86	87	91	97	69	83	90	94	98	73	84	89	93	98	73	84	89	93	98

U = unscreened, S = screened.

Rows 1 to 4 in each matrix are % solved in 100, 200, 500 and 1000 acceptances respectively.

Table 1b: Percentage of problems solved optimally for dataset 20-30.

	10% sampling					50% sampling					90% sampling					100% sampling					
solved	5	4	3	2	1	5	4	3	2	1	5	4	3	2	1	5	4	3	2	1	
BI1 U	4	7	10	15	26	3	5	11	21	29	2	10	13	18	33	2	10	13	18	23	
	9	13	16	24	37	11	18	21	25	45	9	16	21	28	44	9	15	21	29	47	
	15	21	26	36	50	18	23	31	44	59	16	26	33	45	62	19	25	32	41	60	
	21	27	39	48	59	24	29	42	51	68	26	35	42	54	72	24	36	40	50	70	
BI1 S	6	13	19	24	47	11	21	25	39	57	13	18	26	35	52	16	22	26	35	49	
	17	24	37	47	66	19	39	43	54	71	21	26	36	46	63	20	32	37	43	59	
	34	46	55	71	82	35	54	69	74	86	34	47	60	70	82	32	41	48	60	72	
	51	61	72	83	91	58	71	79	86	91	48	60	74	81	90	42	52	57	67	78	
BI2 U	3	8	11	15	28	4	6	10	13	32	4	5	1	7	27	5	6	11	17	25	
	9	14	18	23	44	8	13	11	23	38	7	12	15	24	38	10	14	17	24	34	
	16	22	27	37	53	17	24	31	44	57	13	21	27	35	56	14	18	23	33	46	
	25	31	36	52	64	25	33	43	52	67	21	31	39	46	61	15	19	26	38	53	
BI2 S	6	12	15	25	40	6	12	19	25	46	8	14	21	31	47	11	15	22	28	45	
	15	18	28	39	56	19	24	35	45	62	17	26	35	47	59	20	24	31	41	56	
	29	38	50	57	67	33	44	58	64	77	29	41	49	64	77	26	33	42	50	70	
	36	53	59	70	84	44	60	68	72	82	40	51	59	73	83	29	39	44	54	72	
PI1 U	2	4	11	13	23	4	4	8	14	24	4	6	11	15	21	3	6	8	13	25	
	8	14	17	20	30	7	10	16	25	35	7	13	17	22	32	8	11	16	24	33	
	14	19	26	33	51	14	24	28	33	52	15	20	25	35	54	16	20	28	35	54	
	21	31	33	41	63	23	30	37	48	64	21	29	35	50	70	23	31	36	46	71	
PI1 S	6	9	12	16	28	6	10	13	16	35	9	13	15	20	34	11	12	14	20	29	
	13	15	18	22	41	10	19	21	32	52	14	17	24	35	50	15	16	20	28	45	
	17	21	28	43	65	19	31	45	57	77	19	31	38	54	77	17	31	39	48	73	
	26	34	49	64	78	35	49	66	73	85	35	46	61	73	84	28	44	57	71	84	
PI2 U	2	8	12	19	29	7	7	16	18	31	5	10	15	23	37	6	10	14	20	35	
	8	14	18	27	46	10	17	20	29	47	8	15	24	37	44	9	14	23	34	45	
	16	21	33	48	63	19	27	37	50	64	24	31	39	53	69	17	32	39	54	66	
	30	41	52	62	83	34	40	52	62	80	34	45	55	67	81	33	44	51	63	76	
PI2 S	10	12	14	20	38	12	17	19	26	44	11	15	20	36	41	12	15	21	32	48	
	15	20	27	41	58	16	23	30	44	65	15	22	33	43	59	17	23	32	41	64	
	31	39	52	62	76	28	43	57	66	81	25	39	50	62	67	30	37	49	67	79	
	41	53	66	77	88	47	61	72	80	89	41	48	55	67	78	41	55	67	75	86	
RI1 U	2	3	6	1	3	24	2	7	10	13	19	4	5	7	12	26	2	4	7	15	29
	5	9	13	17	37	6	13	15	20	38	7	13	15	24	38	7	10	19	26	43	
	12	14	25	34	52	11	18	31	38	58	15	24	30	41	56	16	24	34	46	60	
	19	28	37	45	63	20	29	45	57	75	23	36	46	57	72	30	39	46	57	68	
RI1 S	1	3	6	11	25	1	3	9	16	27	2	7	12	18	27	3	6	12	19	29	
	4	8	14	21	40	7	12	18	27	48	11	17	21	29	44	6	18	21	28	50	
	14	22	28	44	57	17	24	40	48	62	25	34	39	51	67	22	31	39	53	74	
	23	31	41	59	66	29	41	50	62	74	33	45	56	67	77	36	48	59	71	82	
RI2 U	1	1	4	5	16	2	3	8	11	20	2	4	4	9	21	2	3	8	10	22	
	3	4	7	1	7	29	3	7	11	24	33	6	6	13	17	37	7	11	16	18	35
	10	15	19	27	50	14	21	24	35	53	13	16	24	34	56	17	20	22	33	50	
	17	19	31	42	66	22	25	31	43	66	21	29	35	49	69	22	30	38	45	62	
RI2 S	3	5	9	12	24	2	4	6	11	21	1	3	6	10	19	1	3	7	11	24	
	6	11	16	20	29	5	10	13	20	30	5	8	18	25	42	6	10	16	23	41	
	10	17	24	32	57	14	17	25	32	49	18	27	35	50	66	13	23	30	45	63	
	17	19	31	42	66	20	25	37	49	67	27	39	48	57	75	28	35	47	60	77	

U = unscreened, S = screened.

Rows 1 - 4 in each matrix are % solved in 100, 200, 500, 1000 acceptances respectively.

Perhaps the most significant result is the effect of screening in the mixed phase. For lower values of p screening improves all results for the BI and PI partitions. For higher values of p and the 10-20 data-set, the unscreened option sometimes registers more solved problems. This is especially true for the higher number of acceptances. For example Figure 3 shows the results for 50% and 100% sampling and the BI1 option. This type of behaviour is in line with our observations concerning the possibility of cycling with an *absolute best* selection option. Those problems which do solve do so more quickly, but the deficit in total number of problems solved is probably due to cycling. This observation was borne out by the graphical experiments. Problems which had solved more easily under the unscreened option repeatedly displayed evidence of cycling under the screened option. There was also a tendency for the PI2 option to cycle more readily, which is in line with the more frequent occurrence of this phenomenon with the PI2 option.

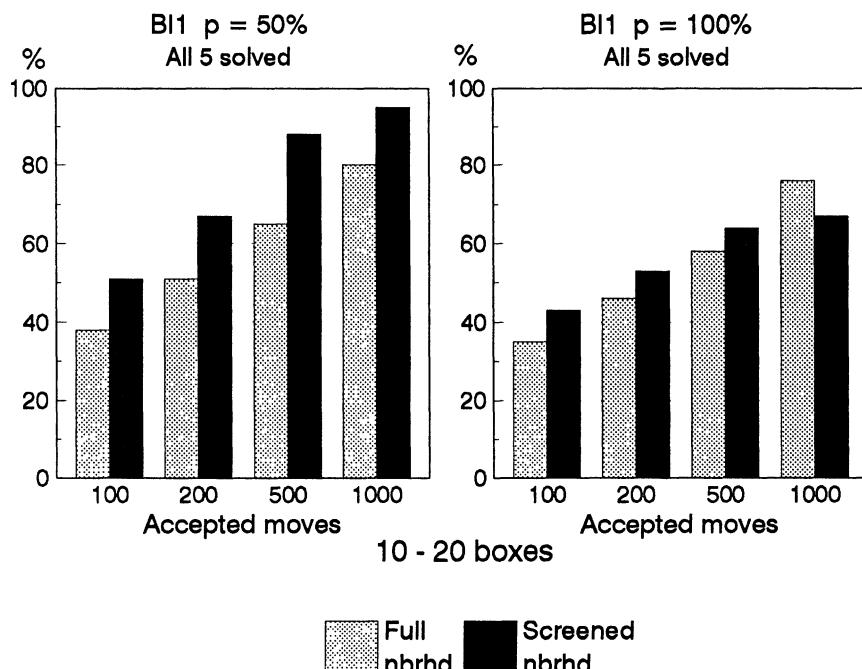


Figure 3: Results for the BI1 option.

The differences are not so apparent for the RI options. The unscreened results using RI1 are roughly in line with those of other methods, but the screened results are inferior. The reasons for this become apparent if we consider the effect of screening, based on overlapping boxes, as the search gets close to local optima. Here there may only be one or two feasible moves in a given subset, which means that the mixed phase is accepting moves of arbitrarily poor quality. This upsets the balance too far in favour of diversification. Again this is best observed dynamically. However Figure 4 adds weight to this argument. This is a plot of the 200 moves following the first local optimum for BI1 and $p=50\%$ (the best BI option) and RI1 and $p = 100\%$. The BI plot shows that local optima are escaped by climbs of varying sizes, with the search frequently maintaining a low trajectory. However, when the RI option is at low cost local optima it usually involves large jumps. Observation of the changing layouts throughout the search confirms that pieces are frequently placed in positions where they are entirely overlapped by existing pieces.

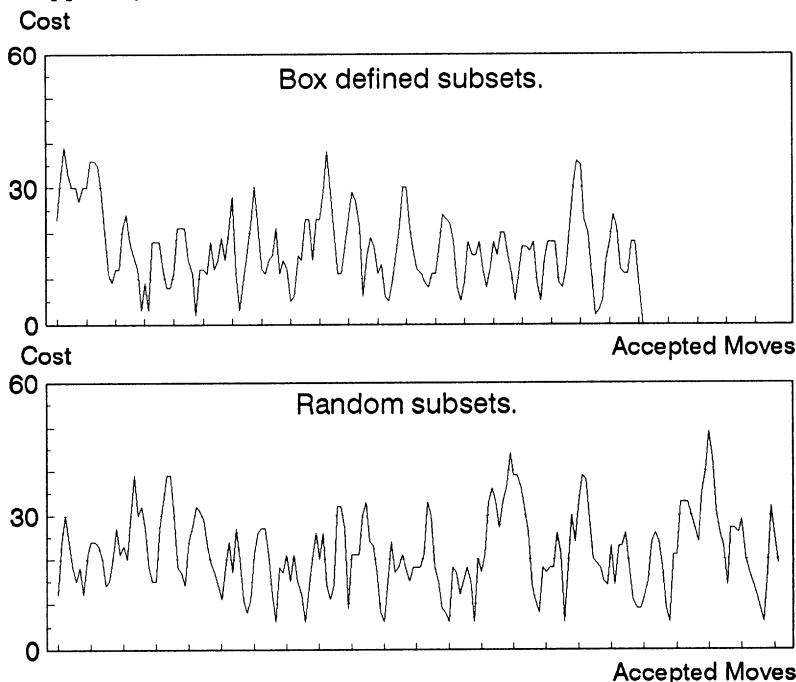


Figure 4: Behaviour of the cost function - BI1 and RI1.

It is also noteworthy that the 100% and 90% sampling results are identical for PI1, PI2 and RI2 for the 10-20 data-set. In the case of PI1 and RI2 this is due to the small subsets, which have between 10 and 20 elements each. The difference between 90% and 100% sampling is therefore just one or two moves, which is unlikely to change the choice of the best sampled move in each subset. The same argument holds for PI2. Although subsets are grouped for sampling, the correct percentage is sampled from each subset in the group, rather than sampling randomly from the group as a whole.

With the exception of a small percentage of the RI options, 50% sampling gave the best overall solution, both in terms of the percentage of problems solved all 5 times and the percentage not solved at all. Figure 5 shows the result of 50% sampling and the screened mixed phase for each of the methods and compares it with the result obtained by running NMSA for the given amount of time. Of the STT methods BI1 scores best on both counts, and closer examination of the data reveals that it generally outperforms BI2 even on problems with long thin boxes for which the latter method was introduced. The argument that BI1 sometimes has difficulty in turning long thin boxes was based on the best selection in each subset. When only 50% of the subset is sampled this argument may not hold. However, it is clear from Table 1, that the overall performance of BI1 is also better than BI2 for 100% sampling. This may be partly due to a fall off in performance on squarer boxes, but it could also be due to the fact that forcing the re-orientation of a piece according to the current subset may cause too many unpromising moves or may be introducing bias towards solutions with equal numbers of horizontal and vertical placements.

Both PI1 and PI2 fail to do as well as BI1, although PI2 is significantly the better of the two. This suggests that the smaller subsets implied by PI1 may be too small. We have not been able to explain the difference between BI1 and PI2, but there is some indication from the graphical experiments that PI2 is still prone to cycling. It is also

possible that sampling 50% from each subset in the group may not be a good idea. It may be better to sample uniformly from the whole set. It is also particularly interesting to note that there is no significant difference between SA and the best version of STT. However, a more detailed comparison is necessary before any definite conclusions can be drawn.

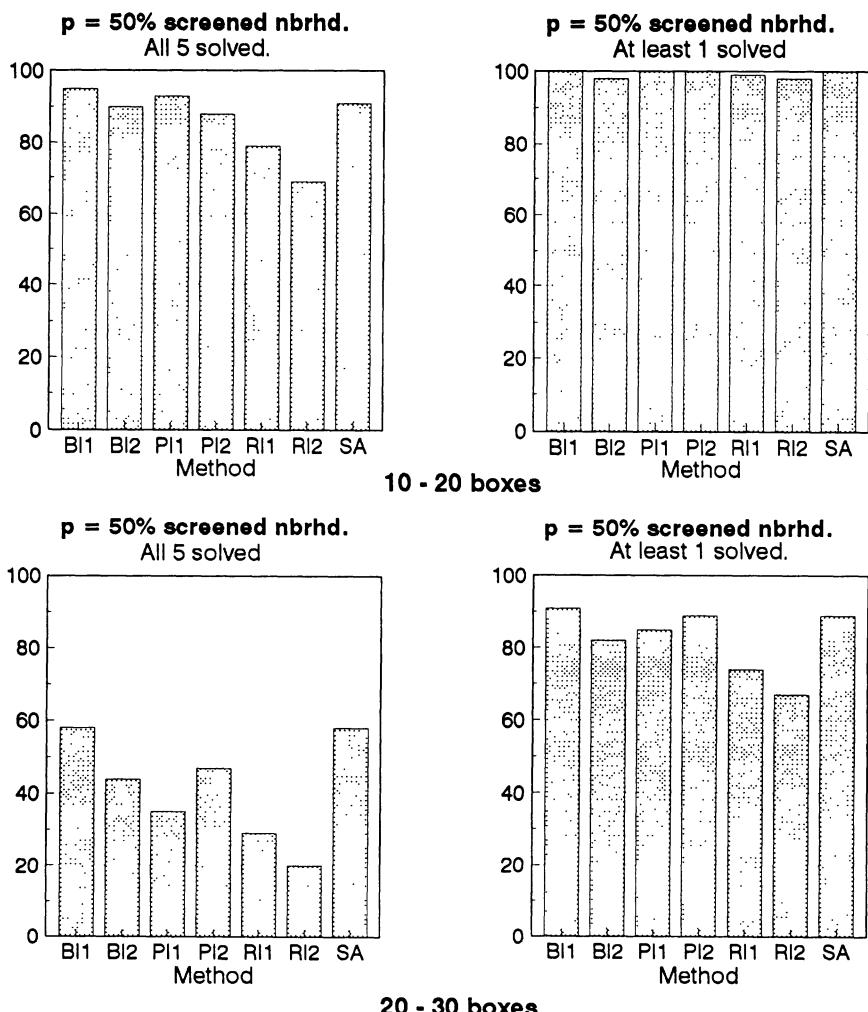


Figure 5: Comparison of Methods.

The results suggest that both methods are reasonably successful at solving problems within the given range. As the search involves a degree of randomness it is not surprising that some problems are only solved some of the time. However, this could also indicate that the search is not diversifying sufficiently to explore enough of the solution space. The relatively low value of $p=50\%$, which gave the best solutions, also suggests that a significant amount of hill-climbing is necessary in order to obtain optimal solutions. These two factors lead us to wonder if the role of the mixed phase in keeping a reasonably low trajectory as the search moves through the solution space may be redundant, and that better, or at least equivalent results may be obtained by repeated application of a descent algorithm from a series of randomly generated starting points.

We therefore implemented this option, repeatedly generating a new random starting solution each time a local optimum was reached and allowing a maximum of 1000 moves per attempt. The results are shown in Table 2. Comparing these with any of the blocks in Table 1, but in particular with the BI1, $p=50\%$ screened option, indicates that STT and therefore NMSA are certainly superior to repeated random descent.

Table 2: Percentage of problems solved optimally - random descent.

solved	10-20 data-set					20-30 data-set				
	5	4	3	2	1	5	4	3	2	1
37	44	55	63	75		3	5	9	13	21
50	56	63	71	85		3	9	15	15	23
62	67	76	81	90		11	15	20	23	32
67	78	80	88	95		14	18	23	29	38

The results were also examined with a view to determining whether or not any of the methods are complementary. The 9 problems which were unsolved by the best STT variant were examined and compared with those problems unsolved by other methods. In all cases a

significant proportion of these 9 problems were unsolved by any method, with the remainder being solved once, or at most twice by some other variant. This suggests that these particular problems give rise to landscapes which are difficult for both STT and NMSA. It is particularly interesting to note that these 9 problems are a subset of the 11 problems which NMSA fails to solve.

Obvious factors affecting the ease or difficulty in solving a given problem are the size of the solution space and the number of global optima. The first is a function of the number of boxes and the number of positions, both of which can easily be measured. The second is more difficult to define as an accurate count would require the generation of all optimal solutions. However it is possible to get a useful estimate from the exact solution approach described in Dowsland (1987). The method utilises a tree search algorithm which is effectively pruned by efficient bounding conditions. For problems within the required range it is not too computationally expensive to run a modified version which continues after finding the first optimal solution and searches for all subsequent solutions of the same quality. Because of the pruning procedures, which make use of symmetries and certain equivalences between patterns which closely resemble one another, this will not be the complete set of optimal solutions. However it will be close enough to indicate whether a particular problem has just one or several thousand global optima. This information was calculated for all 100 problems in the 20-30 data-set. Table 3 shows the 9 unsolved problems, together with the number of pieces to be packed, the number of positions per piece, the size of the neighbourhoods and the number of optimal solutions. Sample means and extreme values for the full dataset are also given. Not surprisingly problems with a large number of optimal solutions tended to solve easily. However, many problems which recorded less than 10 optimal solutions were solved very successfully. Closer inspection revealed that difficult problems have few solutions, tend to be large (all the 10-20 data set solved at least once) and frequently involve long thin boxes, long thin pallets or both.

The significance of problem size and the number of optimal solutions is obvious and the significance of long thin boxes in terms of escaping local optima has already been noted. It therefore remains to comment on the difficulty caused by long thin pallets. Such pallets are more likely to have more than one perfectly fitting combination of dimensions along the longer of its edges. Once a particular configuration is in place it becomes difficult to change as there are no gaps to reduce the amount of overlap caused by placing new pieces in the same region. If there are sufficient low cost moves using the remainder of the pallet area, then this effectively means the region of the solution space containing the global optima is separated from that containing the current solution, by a relatively steep slope. This does not explain why repeated attempts to solve the same problem tend to converge to the 'wrong' region of the solution landscape, unless it is far larger than the area containing the global optima.

Table 3: Difficult problems

Pallet	Box	Pieces	Positions	Nbrs	Optimal Solutions
19x16	5x3	20	118	2360	6
31x19	8x3	24	144	3456	6
30x16	8x3	20	122	2440	1
20x17	7x2	24	158	3792	8
33x26	11x3	26	180	4680	6
19x16	5x3	20	118	2360	6
29x16	7x3	22	132	2904	2
27x25	8x3	28	214	5992	24
36x22	11x3	24	160	3840	1
Mean:		23.7	149.3	3619.5	1.9×10^6
Min:		20	50	1250	1
Max:		29	238	6902	1.6×10^7

Note: the dimensions given are not those originally generated but the reduced dimensions according to the procedure described in Dowsland (1993).

7. Conclusions and suggestions for further research

Although the performance of STT is far from being consistently optimal on many of the test problems, it performs well when measured against other local search strategies. At first glance it does not appear to be very much better than the undulating temperature version of SA but there is some evidence that its scope may extend beyond that of NMSA, especially in tackling larger problems with squarish boxes. Previous research using SA suggests that this level of performance is sufficient to render further research into the use of STT in the solution of more complex cutting and packing problems worthwhile.

The comparison with random descent indicates that even in relatively spiky solution spaces the low trajectory ensured by STT or NMSA is beneficial. Examination of the unsolved problems show that both methods perform well in terms of converging on good local optima, and consistent failure to find global optima tends to occur only when such points are very sparsely scattered in spaces riddled with local optima.

The success of the screening option in the improving phase shows that this can be very effective in improving the performance of the search. Here, as in many satisficing problems, the moves are defined by problem entities, some of which will contribute to the cost of the current solution. As the screening option screens out moves involving non-contributors, the number of feasible moves is likely to be small when the search is close to good local optima. It is also likely that not all screened moves are promising moves. This research has highlighted two factors which should be borne in mind when implementing this type of screening. The first is the possibility of cycling if mixed phase screening is used with an absolute best selection policy. The second concerns the way in which the subsets are defined, and our results suggest that the set of moves should be partitioned so that the screening process eliminates whole subsets, rather than just leaving a few feasible moves in each.

Apart from a more detailed comparison with NMSA, future investigations may take one of two paths. From a theoretical point of view it would be interesting to analyze the landscapes produced by different problems, with a view to determining features which cause particular difficulties for this type of algorithm. However, even simple problems such as this give rise to very large and complex solution spaces and it is not immediately obvious how such an analysis should be carried out.

From a more practical view point, the work could progress in the direction of the work on NMSA and the lessons learned from the pallet loading problem could be used to aid the implementation of STT for the more complex problem of packing different sized pieces, or irregular pieces. In order to get the best out of NMSA, several new features, such as working with two different cost functions in parallel, and the addition of occasional 'illegal' optimisation steps were introduced. It will be interesting to see if STT can find solutions of similar quality without these modifications. If not, incorporating similar features into an STT implementation will provide further challenges.

8. References.

- J. Blazewicz, P. Hawryluk, and R. Walkowiak, Using a tabu search approach for solving the two-dimensional irregular cutting problem, *Annals of Operations Research*, 41 (1993) 313-327.
- K. A. Dowsland, An exact algorithm for the pallet loading problem. *EJOR* 31 (1987) 78-84.
- K.A. Dowsland, Some experiments with simulated annealing techniques for packing problems. *EJOR* 68 (1993) 389-399.
- F. Glover, Simple tabu thresholding in optimization, Internal Report, University of Colorado, Boulder, Colorado (1992).
- M.Lundy and A. Mees, Convergence of an annealing algorithm, *Mathematical Programming* 34 (1986) 111-112.

- H. Lutfiyya, B. McMillin, D.A.P. Poshyanon, and C. Dagli, Composite stock cutting through simulated annealing, *Mathematical Computer Modelling*, 16 (1992), 1, 57-74.
- V.M.M. Marques, C.F.G. Bispo, and J.J.S. Sentieiro, A system for the compactation of two-dimensional irregular shapes based on simulated annealing, *Proceedings of IECON - 91* (IEEE - 1992), 1911-1916.
- J.F.C. Oliveira and J.A.S. Ferreira, Algorithms for nesting problems, *Applied Simulated Annealing*, ed. R.V.V. Vidal, Lecture Notes in Econ and Maths Systems 396, Springer Verlag (1993), 255-274.
- V. Valls, R. Martí and P. Lino, A tabu thresholding algorithm for arc crossing minimization in biparite graphs. *Internal Report. Departamento de Estadística e Investigación Operativa*. Universidad de Valencia, Spain (1993).

**CRITICAL EVENT TABU SEARCH FOR
MULTIDIMENSIONAL KNAPSACK PROBLEMS**

Fred Glover
Graduate School of Business, Box 419
University of Colorado at Boulder
Boulder, Colorado, 80309-0419
E-Mail: fred.glover@colorado.edu

Gary A. Kochenberger
College of Business
University of Colorado at Denver
Denver, Colorado 80217-3364
E-Mail: gkochenberge@castle.cudenver.edu

Abstract

We report a new approach to creating a tabu search method whose underlying memory mechanisms are organized around "critical events." A balance between intensification and diversification is accomplished by a strategic oscillation process that navigates both sides of the feasibility boundary, and serves to define the critical events. Surrogate constraint analysis is applied to derive choice rules for the method. Computational tests show the approach performs more effectively than previous heuristics for multidimensional knapsack problems, obtaining optimal solutions for all problems in a standard testbed.

Introduction.

Considerable progress has been made over the past decade in developing and testing search techniques for combinatorial optimization problems. Various forms of genetic algorithms, simulated annealing, and tabu search have all been reported to perform well in diverse problems settings. Tabu search (TS) has proved highly effective in various implementations across a wide range of applications, and forms the foundation of the approach of this paper. Background on applications of TS and elements responsible for its successes can be found in recent survey papers [8, 10, 11].

The distinguishing characteristic of tabu search is the use of adaptive memory structures which are accompanied by associated strategies for exploiting information during the search. The adaptive memory component of TS typically incorporates recency-based and frequency-based memory defined over varying short term and long term spans of the search history. This memory operates through control mechanisms of tabu restrictions and aspiration criteria, and associated penalties and inducements to modify move evaluations. By such adaptive designs, which abstract certain processes conjectured to operate in human problem solving, this memory is sometimes contrasted with the "rigid" memory structures of branch and bound, and the "memoryless" designs of a variety of other approaches.

The choice rules of tabu search are highly aggressive, seeking "best moves" at each step (from those allowed by its restrictions and made available from its history). To apply this aggressive orientation judiciously, the approach characteristically relies on strategies to isolate subsets of desirable moves (rather than to exhaustively examine all alternatives at each iteration), and utilizes choice criteria that make use of a notion of *influence* (which includes elements of criticality and patterning), in addition to drawing on the outcomes of search history. A principal theme is to create an effective interplay between intensification and diversification, that is, between approaches designed

to identify and reinforce attributes of good solutions and approaches designed to draw the search into promising new regions. A general treatment of tabu search and its principal strategies may be found, for example, in [8] and [9].

In this paper we focus on a special subset of these elements and report on a new method for creating an effective search process based on a flexible memory structure that is updated at *critical events*. The tabu status of a potential move is determined through the integration of recency-based and frequency-based memory information. A balance between intensification and diversification is accomplished by a strategic oscillation scheme that probes systematically to varied depths on each side of the feasibility boundary. These oscillations, coupled with dynamic tabu information, guide the search process toward different critical events.

The approach illustrated here applies to many types of problems. However, we specifically will be concerned with an implementation designed to solve multidimensional knapsack (MK) problems, which take the form:

$$\begin{aligned} \text{MK:} \quad & \text{maximize } c\mathbf{x} \\ & \text{subject to} \\ & A\mathbf{x} \leq b \\ & \mathbf{x} \text{ binary} \end{aligned}$$

The matrix A and the vectors b and c consist of real-valued constants that satisfy

$$A \geq 0 \text{ and } b, c \geq 0.$$

1. Basic Notions.

Our approach is representative of a class of strategic oscillation methods that proceed by alternating between constructive and destructive phases. In the present setting, a constructive phase

corresponds to one that progressively sets variables equal to 1, while a destructive phase corresponds to one that progressively sets variables to 0.

Within this framework, we organize a strategic oscillation as follows. We first sketch the ideas in overview, and then diagram the steps of the approach. We introduce a parameter *span* that indicates the amplitude (or depth) of the oscillation about the feasibility boundary, measured in the number of variables *added* (set to 1) when proceeding from the boundary into the infeasible region and in the numbers of variables *dropped* (set to 0) when proceeding from the boundary into the feasible region. Although the depth in these two directions need not be the same (and in fact asymmetric or even "one-sided" oscillations can be best for certain problems) we focus here on treating them as equal.

We begin with span equal to 1, and gradually increase it to a limiting value. For each value of span, a series of alternating constructive and destructive phases is executed before progressing to the next value. At the limiting point, we begin to gradually decrease span, allowing again a series of constructive and destructive phases for each span value before progressing to the next. When span reaches a value of 1, we begin once more a gradual increase in span. The manipulation of the span parameter may be viewed as an *outer oscillation* that contains the oscillation of constructive and destructive moves (for a given value of span) within it.

1.1 Method in Detail.

Let x^* denote the best solution found so far; that is, the feasible solution that gives a maximum value for cx . Until x^* has been determined, cx^* is taken to be a large negative number. We assume that setting all $x_j = 1$ is infeasible, else the problem is trivially solved. The terms *tabu* and *non_tabu* will be given operational definitions subsequently. Our strategic oscillation method, then, consists of the

following:

Initialization

Step 0: Begin with $x = 0$ or $x = e$ and set $\text{count_span} = 0$. If $x = 0$, set $\text{feasible} = \text{.true.}$ and go to the Constructive Phase. If $x = e$, set $\text{feasible} = \text{.false.}$ and go to the Destructive Phase.

Constructive Phase: (move from feasible to infeasible)

Step C1: (feasible = .true.)

- (1) If no component x_j of x can be increased from 0 to 1 except by violating feasibility, then:
 - (a) if $cx > cx^*$, let $x^* = x$
 - (b) set $\text{feasible} = \text{.false.}$ and go to step C2.
- (2) If condition (1) does not hold, then choose an x_j to increase from 0 to 1, such that the move maintains feasibility, and return to the start of step C1.

Step C2: (feasible = .false.)

Set $\text{count_span} = \text{count_span} + 1$

- (1) If $\text{count_span} > \text{span}$, or if there are no x_j available to change from 0 to 1, go to the Transfer Phase.
- (2) If condition (1) does not hold, choose an x_j to increase from 0 to 1 and return to the start of Step C2.

Transfer Phase:

- Step T1:* Set $\text{count_span} = 0$. Change the value of span if appropriate (by a rule to be identified later).
- Step T2:* Go to the Destructive Phase if the last phase was a Constructive Phase. Else, go to the Constructive Phase.

Destructive Phase: (move from infeasible to feasible)

Step D1: (feasible = .false.)

- (1) Select an x_j to change from 1 to 0.
- (2) If the solution produced by (1) is infeasible, return to the start of Step D1. Otherwise, set feasible = .true. and:
 - (a) if $c x > c x^*$, let $x^* = x$.
 - (b) go to Step D2.

Step D2: (feasible = .true.)

Set count_span = count_span+1

- (1) if count_span > span, or if there are no x_j available to change from 1 to 0, go to the Transfer Phase.
- (2) If condition (1) does not hold, choose an x_j to decrease from 1 to 0 and return to the start of Step D2.

The preceding method terminates whenever a selected number of iterations have been performed without finding an improved x^* , or simply after a total iteration limit (number of Transfer Phase executions) has been reached.

1.2 Surrogate Constraint Choice Rules.

As in a variety of strategic oscillation approaches, we make use of surrogate constraint information to guide the decisions of the method. In particular, the choice of a variable to add (steps C1(2) and C2(2)) and of a variable to drop (steps D1(1) and D2(2)) is determined by a standard surrogate constraint evaluation. (For background on surrogate constraints and their uses in choice rules, see Glover [6, 7].) We dynamically form surrogate constraints, based upon the current solution, as normalized (nonnegative) linear combinations of the problem constraints. At each iteration, we compute ratios of the objective function coefficients to their associated surrogate constraint

coefficients. During a Constructive Phase, we add variables with maximum ratios and during a Destructive Phase we drop variables with minimum ratios. When tabu restrictions are enforced, this ratio information is augmented by recency and frequency information.

The surrogate constraints used to determine the choice rules are formed by first normalizing (or otherwise weighting) and then summing various constraints. We currently employ three different surrogates depending on the feasibility status of the current solution vector and the phase of the search. In all cases we compute

$$b'_i = b_i - \sum(a_{ij} : \text{for } j \text{ with } x_j = 1).$$

As long as the current solution is feasible, the weight w_i for constraint i is chosen to be $1/b'_i$. When the search process enters the infeasible region, we choose weights by one of two methods:

- a. If $b'_i > 0$, set $w_i = 1/b'_i$
If $b'_i \leq 0$, set $w_i = 2 + |b'_i|$
- b. If $b'_i \geq 0$, set $w_i = 0$
If $b'_i < 0$, set $w_i = 1/(|b'_i| + \sum(a_{ij} : \text{for } j \text{ with } x_j = 0))$

We have experimented with both methods and currently use method (a) in the Constructive Phase and method (b) in the Destructive Phase. The rationale for the above procedures is to exaggerate the influence of the most violated constraints and thus encourage searches "near" critical solutions. (Our testing to date has not conclusively demonstrated whether one method is preferred to the other.)

Let $\sum s_j x_j \leq s_o$ denote the resulting surrogate constraint, i.e., $s_j = \sum w_i a_{ij}$ and $s_o = \sum w_i b_i$. The choice rule for the constructive phase then selects the variable x_j to change from 0 to 1 in order to

$$\text{Maximize } (c_j/s_j: x_j = 0)$$

while the choice rule for the destructive phase selects the variable x_j to change from 1 to 0 in order to

$$\text{Minimize } (c_j/s_j: x_j = 1).$$

These rules are modified by the TS restrictions and penalties subsequently indicated. In the next sections, we describe how we accomplish this in our implementation.

2. Tabu Memory.

The essence of the method rests upon the scheme for defining tabu status, and hence that guides the oscillations productively. In our implementation, the tabu status of a potential move (adding or dropping a variable) is determined by recency and frequency information gathered as the search process encounters critical events. For MK problems, we define a critical event to be the construction of a complete (feasible) solution by the search process at the feasibility boundary. That is, critical events correspond to solutions obtained by the Constructive Phase at the final moment before going infeasible, and by solutions obtained by the Destructive Phase at the first moment of regaining feasibility.

At critical events, we also include steps for generating additional trial solutions which are further candidates for x^* . In the Constructive Phase, we proceed as follows. As variables are chosen to be changed from 0 to 1, a move is finally made that drives the search infeasible. When such a step is imminent, a trial solution can be identified if some variable can be found to add (change from 0 to 1) that "fits" within the constraints even though it contributes less to the objective function than the one normally chosen (that produces infeasibility). Thus, at such a juncture, the variables are examined, in order of decreasing

objective function coefficients, for the first such variable that can be added while maintaining feasibility. The trial solution choice does not replace the customary choice (in the standard variant we employ), but simply provides a solution that is recorded if it improves the best one currently known.

The second trial solution (at a critical solution in the Constructive Phase) is generated by retaining the regularly selected move that produced infeasibility (as noted) and searching for a variable to drop (other than the one just selected to add) that will re-establish feasibility. Candidate variables to drop for the purpose of generating this second trial solution are examined in order of increasing objective function coefficients.

Corresponding to the two trial solutions generated at the critical level of the Constructive Phase, we similarly generate two such solutions at the critical level of the Destructive Phase. The rules are the mirror images of the above steps. Note that trial solutions generated in this way represent the use of an aspiration criterion since the solutions are found by temporarily ignoring tabu information.

2.1 Using Recency and Frequency Information.

In order to influence the search by recency information, we record (in a circular list) the last t solutions obtained at critical events, where in our implementation t is a simple function of problem size that takes values in the range of 3 - 12. Using this record of the last t critical solutions, which range from $x(\text{last})$ to $x(\text{last}-(t-1))$, we maintain a short term recency tabu vector (TABU_R) that is the sum of the last t solutions. That is, each time a new $x(\text{last})$ is identified, we set

$$\text{TABU_R} = \text{TABU_R} + x(\text{last}) - x(\text{last}-t).$$

In a like manner, long term frequency information is captured for use in the search process by maintaining another vector (TABU_F)

which is the sum of all critical solutions encountered to date (rather than the last t critical solutions).

As previously noted, during a Constructive Phase, variables are added until the span parameter indicates it is time to switch to the Destructive Phase. The search process then "turns around" and proceeds toward (and past) the feasibility boundary by dropping variables. Eventually, we switch again to the Constructive Phase, where we turn around and move toward infeasibility by adding variables. It is at these *turn around* points that we use the foregoing memory to impel the search process to head in new directions in pursuit of new critical solutions. We sketch our rationale for accomplishing this as follows.

Suppose we have just switched from the Destructive Phase to the Constructive Phase. Our goal is to add variables that have not appeared at a value of 1 in recent critical solutions. Thus, we seek to impose the condition that the first variable added back, x_j , will have $TABU_R(j) = 0$. That is, we may conceive the condition $TABU_R(j) > 0$ as implying that x_j is tabu. However, this is not broad enough for our purpose. More generally, we seek to require that the first k variables added back (after turning around) will have $TABU_R(j) = 0$. Such a requirement may not be strictly possible. Consequently, we attach a large penalty weight, PEN_R , to $TABU_R(j)$ and create a penalty value $PEN_R * TABU_R(j)$. This penalty value is subtracted from the ratio evaluation, indicated earlier whose maximum value is used to choose the preferred move.

Likewise, frequency information is included by subtracting another penalty term, $PEN_F * TABU_F(j)$, from the ratio evaluation. The positive weight, PEN_F , is scaled by the iteration count so that the penalty influence derived from long term frequency information is small compared to that of the short term recency information. In this manner, long term tabu information plays a subtle but useful role of

breaking ties that may otherwise occur if one utilizes only short term tabu information.

Similarly, when we switch from the Constructive to the Destructive Phase, we seek to restrict our choice of variables to drop so that the first k variables chosen are selected from those x_j with maximum entries in the tabu lists TABU_R and TABU_F. Here, the tabu restriction is implemented by creating an inducement to drop a variable by subtracting an associated penalty term from the ratio evaluation and seeking the minimum penalized ratio.

Since the local search is based on a ratio evaluation, we choose the penalty coefficients by reference to this evaluation. This is done as follows. Each original constraint is scaled by its RHS value and the resulting weighted rows are summed to yield a surrogate constraint from which the ratio r_j , for each column j , is computed. Denote the maximum r_j ratio by r^* . Then, our penalties are set as follows:

$$\text{PEN_R} = r^*$$

$$\text{PEN_F} = r^*/s$$

where s is a scale factor given by the product of the iteration count and a large, positive constant(C). Note that since the iteration count is an over estimate of the maximum TABU_F value at any iteration, the above calculation for the long term tabu penalty corresponds to scaling TABU_F by a simple function of the maximum TABU_F value. We experimented with several values (over a wide range) for C . For the results reported later in this paper, C was set to 100,000.

Other approaches to incorporating long term (e.g. TABU_F) information into the search process are available as well. For example, the penalties could be adaptively computed on the basis of the current surrogate constraint instead of relative to an initial surrogate constraint.

Also, TABU_F(j) could be normalized by the sum of the TABU_F(j) values. We comment briefly on this alternative, along with the notion of postponing the use of long term information, later in the paper.

2.2 Penalty Calculations.

The calculations for handling the penalty terms in both the Constructive and Destructive phases are as follows.

Denote the usual evaluation for variable x_j by ratio(j). Let count_var denote the number of variables chosen since the last turn-around and let j^* be the index of the variable to be chosen next. Then in the Constructive Phase, we choose j^* (from the set of all j such that $x_j = 0$) as follows:

If count_var > k, let value(j) = ratio(j)

if count_var $\leq k$, let value(j) = ratio(j) - PEN_R*TABU_R(j)
 - PEN_F*TABU_F(j)

Then j^* corresponds to the variable with the maximum value of value(j). Similarly, in the Destructive Phase, we choose j^* (from the set of all j such that $x_j = 1$) in order to minimize value(j), using the same two calculations of this value shown above.

The parameter k, the number of tabu-influenced adds or drops to be made immediately after a "turn around" is managed in a fashion that fosters additional diversity in the search process. We start with $k = 1$, and after a number of iterations (e.g., $2t$), we set $k = k+1$. We continue in this fashion until k reaches a limit (KMAX) at which point we set k back to 1 and the process repeats. In general, KMAX is a function of problem size. However, over a rather wide range of problems, KMAX = 4 has performed well in our testing.

2.3 Controlling the Outer Oscillation parameter (Span).

The oscillations about the feasibility boundary are shaped by the parameter *span*. Span is fixed for a certain number of iterations and then changed in a systematic fashion. In our implementation, we manage span in the Transfer Phase by the following rules:

Initialization:

Set span=1, choose values of parameters p1 and p2, and designate that span is increasing.

Transfer Phase (increasing span):

For span from 1 to p1: Allow $p_2 * \text{span}$ executions of the Constructive and Destructive Phase and then increase span by 1.

For Span from p1+1 to p2: Allow p_2 executions of the Constructive and Destructive Phases and then increase span by 1. When span is increased beyond p2, set it back to p2 and designate that span is decreasing.

Transfer Phase (decreasing span):

For span from p2 to p1+1: Allow p_2 executions of the Constructive and Destructive Phases, and then decrease span by 1.

For span from p1 to 1: Allow $p_2 * \text{span}$ executions of the Constructive and Destructive Phases, and then decrease span by 1. When span reaches 0, set it back to 1 and designate that span is increasing.

Large values of p1 and p2 foster aggressive diversification while smaller values facilitate a search in a closer neighborhood of the most recent critical solution. The default values we employ are $p_1 = 3$ and $p_2 = 7$. These particular values are supported by the fact that span values of 1 to 3, with a predominant emphasis on the value 3, have

been found effective in the context of labor scheduling problems. We include values of span larger than $p_1 = 3$ in order to expand the range of alternatives examined. Clearly other values for these parameters, and other schemes for managing span may hold promise as well. We undertook to implement easily determined small values, without spending a lot of time to explore alternatives. As it turned out, our simple parameter choices worked well for us, as shown in the following section on computational results.

3. Computational Experience.

Multidimensional knapsack problems have been the object of many research initiatives over the past forty years. Early work, exemplified by that of Lorie and Savage [13], was motivated by applications in capital budgeting. Since that time, many other applications have been discussed and a variety of algorithms have been proposed. This class of problems is known to be NP-hard, and despite the considerable attention these problems have received over the years, optimal solutions remain elusive.

Many researchers have attempted to locate high quality, if not optimal, solutions to MK problems by designing and employing various heuristics (e.g., Senju and Toyoda [15], Loulou and Michaelides [14], and Fréville and Plateau [5]). These approaches showed considerable promise and laid the groundwork for continuing work in designing heuristics for this class of problems.

Building on these foundations, recent papers by Drexel [4], Dammeyer and Voss [3], Aboudi and Jornsten [1], and Glover and Lokketangen [12] have reported computational experience with new heuristic approaches to 57 standard multidimensional knapsack problems taken from a test set due to Drexel [4] which is currently available from Beasley [2]. Since this particular problem set has become the standard by which various approaches are compared, we chose to test our Tabu Search approach on these problems as well.

Starting with initial values ($p_1 = 3$, $p_2 = 7$, and $t = 7$), each of the 57 standard test problems was run for a total of 50 outer SPAN oscillations. For each run (restart), KMAX was set to four. On 42 of the 57 problems, the optimal solution was found on the first run. On the other fifteen problems, optimal solutions were found on runs later than the first. For these runs, the initial parameter values were successively modified as follows. Keeping $p_2 = 7$, t was decreased to 5 and then to 3. Parameter p_2 was then lowered to 5 and t was again allowed to take on the values 7, 5 and 3. In all additional runs, p_1 was kept at the default value of 3. Optimal solutions were found for all of the 57 test problems by this approach, which introduces a maximum of six restarts, each beginning from the same zero solution vector.

None of the other published approaches to these standard test problems report finding the optimal solution for all 57 problems. Drexel [4] obtained his results using two different implementations of a simulated annealing algorithm. Each problem was solved starting from 10 different starting points. That is, each problem was solved 20 times. He reports finding an optimal solution in 25 of the 57 problems tested.

The results reported by Dammeyer and Voss [3] were obtained from three different versions of an add/drop exchange heuristic with *reverse elimination* tabu lists. For each version, each problem was solved from 20 different starting points. The solutions reported are the best solutions found in the 60 attempts for each problem. Across the 57 test problems, they report an optimal solution in 41 cases.

Aboudi and Jornsten [1] report results obtained from their implementation of an LP-based "pivot and complement" heuristic, which they also guide by a form of tabu search. Twenty different versions (variations) of their heuristic were tried on each problem. The results they report are the best solutions found for each problem. Out of the 57 test problems they reported the optimal solution in 49 cases.

(However, no single variation they implemented performed nearly this well.)

Lokketangen and Glover [12] report a tabu search heuristic that also uses adjacent extreme point moves, which correspond to non-tabu pivots to adjacent basic feasible solutions. Each problem was solved twice (using a different move evaluation and choice rule). The results they report are the best solutions found for each problem. For the 57 problems, they report the optimal solution in 54 cases.

Comparisons of the various methods should be made with considerable care. Issues of tuning, coding, quality and number of starting points, and parameter settings make comparisons difficult. In addition, the approach of Glover and Lokketangen [12], is designed for general 0/1 mixed integer problems rather than being a specialized algorithm for multidimensional knapsack problems.

Further Remarks on our Computational testing:

1. The results we report were obtained without the use of any reductions or variable pegging due to pre-processing. The starting solution for each problem was $x = 0$.
2. The importance of generating additional trial solutions by a simple adjustment of variables (that overrides tabu conditions at critical events) is substantial. For the 57 problems reported on here, the optimal solution was obtained from such trial solutions about 82% of the time. This highlights the effectiveness of our search process in generating critical events in the neighborhood of the optimal solution.
3. Several problems required multiple span cycles (complete outer oscillation on both sides of the feasibility boundary) to reach an optimal solution. This highlights the difficulty of these

problems and underscores the effectiveness of our flexible tabu structure and strategic oscillation scheme in mounting a robust search of the solution space. Over the 57 problems, optimal solutions were found in an average of approximately 7 span cycles. Thirty nine of the problems were solved optimally in four or fewer cycles. Four of the problems took more than 25 cycles to locate the optimal solution and one problem took 40 cycles.

4. Many of the 57 problems were also solved optimally utilizing only short term information, that is, by setting PEN_F =0. Generally, long term frequency information appears most useful (leading to optimal solutions in fewer span cycles) on the more difficult problems.

In an effort to explore this further, along with some alternative ways of utilizing long term information, we conducted a variety of additional runs on the 10 most difficult problems. These problems required the largest number of full span cycles to locate optimal solution. Recall that the results reported above were obtained utilizing long term frequency information throughout the search process in the manner depicted in section 2.1. For the purpose of comparison, we solved these 10 problems again under the following conditions: (a) using short term memory only; (b) using long term memory, but suppressing (withholding) the influence of long term memory in the search process for a certain number of iterations; (c) incorporating long term information as a penalty by normalizing TABU_F(j) by the sum of the TABU_(k) values rather than the scheme outlined in section 2.1; and (d) using long term information as described in (c) but suppressing it's impact for a certain number of iterations. For alternatives (b) and (d), long term frequency information was withheld for 100 iterations. In all cases, exactly the same parameter values were

used as those that led to the reference results.

The results obtained from these runs underscore the important role of long term information in the search process. On average, it took 2.25 times as many span cycles to locate the optimal solution when using short term information only. When using long term information, but withholding its influence for the first 100 iterations, (e.g. case (b)), it took an average of 1.875 times as many span cycles as the reference case. Finally, for the conditions of (c) and (d), it took, respectively, 3.125 and 3.375 times as many span cycles on average to locate the optimal as the reference case.

Thus, on the problems considered here, the implementation described in this paper outperformed alternatives (a), (b), (c), and (d) in terms of average performance by a substantial amount. In addition, it produced the individual best performance on 6 out of the 10 problems and was generally close to most preferred on the others. Alternatives (a), (b) and (d) each produced the best individual problem result for one of the ten problems and there was a tie for best performance on the tenth problem.

5. In addition to the standard 57 test problems, we tested our approach on 24 randomly generated (correlated) problems ranging in size up to 500 variables and 25 constraints. Where possible, optimal solutions were obtained by a branch and bound algorithm in order to assess the performance of our heuristic. These problems are fairly difficult and in some cases the branch and bound algorithm ran for more than 4 consecutive days on a 486 machine without terminating with a proven optimal solution. For 23 of these 24 problems, our approach generated solutions that were generally appreciably superior to those given by the branch and bound algorithm. In

the one exception, we reported an objective function value of 4524 compared to the branch and bound generated value of 4525. In all cases, our procedure required a fraction of the time taken by the branch and bound procedure, completing its run in about 28 CPU minutes on average, and 355 CPU minutes in the worst case. (This represents a speed difference of about 50 to 1.) To provide an additional (perhaps more informative) basis of comparison, we note that the time required to *first* reach the best solution found by each of these methods averaged 22 CPU minutes for our TS method and 1250 CPU minutes for the branch and bound (yielding a time advantage for the TS approach of about 55 to 1). The fact that TS additionally found better solutions in 23 of the 24 cases increases the significance of these differences.

4. Conclusion.

In this paper we report an implementation of a new tabu search approach to multidimensional knapsack problems. Our approach employs a flexible memory structure that integrates recency and frequency information keyed to critical events of the search process. The method is enhanced by a strategic oscillation scheme that alternates between constructive and destructive phases, and drives the search to variable depths on each side of the feasibility boundary.

Our approach successfully obtained optimal solutions for each of 57 standard test problems from the literature, a level of performance not matched by other attempts reported in the literature. Moreover, we found best known solutions to 23 of 24 additional multidimensional knapsack problems (correlated, randomly generated). Our reliance upon exceedingly simple parameter values, without conducting extensive exploration of alternatives, suggests that the use of such critical event memory in tabu search may be valuable in other applications.

5. REFERENCES

- R. Aboudi and K. Jornsten (1994) "Tabu Search for General Zero Integer Programs Using the Pivot and Complement Heuristic," *ORSA Journal on Computing*, Vol. 6, No. 1, Winter 1994, pp. 82-93.
- J. Beasley (1995) "OR-Library," available by anonymous ftp to msemga.ms.ic.ac.uk.
- F. Dammeyer and S. Voss (1993) "Dynamic Tabu List Management Using Reverse Elimination Method," *Annals of Operations Research*, No. 41, pp. 31-46.
- A. Drexl (1987) "A Simulated Annealing Approach to the Multiconstraint Zero-One Knapsack Problem," *Computing*, No. 40, pp. 1-8.
- A. Fréville and G. Plateau (1986) "Heuristics and Reduction Methods for Multiple Constraints 0-1 Linear Programming Problems," *European Journal of Operations Research*, No. 24, pp. 206-215.
- F. Glover (1965) "A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problem," *Operations Research*, 13, 879-919.
- F. Glover (1977) "Heuristics in Integer Programming Using Surrogate Constraints," *Decision Sciences*, Vol. 8, No. 1, January, pp. 156-166.
- F. Glover (1994) "Tabu Search Fundamentals and Uses," University of Colorado Working Paper.
- F. Glover (1993) "Tabu Thresholding: Improved Search by Nonmonotonic Trajectories," to appear in *ORSA Journal on Computing*.
- F. Glover and M. Laguna (1993) "Tabu Search," *Modern Heuristic Techniques for Combinatorial Problems*, C. Reeves, ed., Blackwell Scientific Publishing, pp. 70-141.

- F. Glover, M. Laguna, E. Taillard, and D. de Werra, eds (1993) "Tabu Search," special issues of the *Annals of Operations Research*, Vol. 41, J.C. Baltzer.
- F. Glover and A. Lokketangen (1994) "Solving Zero-One Mixed Programming Problems Using Tabu Search," University of Colorado Working Paper.
- J. Lorie and L. Savage (1955) "Three Problems in Capital Rationing," *Journal of Business*, No. 28, pp. 229-239.
- R. Loulou and E. Michaelides (1979) "New Greedy-Like Heuristics for the Multidimensional 0-1 Knapsack Problem," *Operations Research*, no. 27, pp. 1101-1114.
- S. Senju, and Y. Toyoda (1968) "An Approach to Linear Programming With 0-1 Variables," *Management Science*, No. 15., pp. 196-207.

**Solving Dynamic Stochastic Control Problems in Finance Using
Tabu Search with Variable Scaling**

Fred Glover
School of Business
University of Colorado at Boulder
Boulder, CO 80309
E-mail: fred.glover@colorado.edu

John M. Mulvey^{*}
Department of Civil Engineering and Operations Research
Princeton University
Princeton, NJ 08544
E-mail: mulvey@macbeth.princeton.edu

Kjetil Hoyland
Department of Managerial Economics and Operations Research
University of Trondheim
N-7034 Trondheim, Norway
E-mail: kth@iok.unit.no

Abstract

Numerous multistage planning problems in finance involve nonlinear and nonconvex decision controls. One of the simplest is the fixed-mix

* Research supported in part by NSF grant CCR-9102660 and Air Force grant AFOSR-91-0359. We acknowledge the assistance of Michael T. Tapia in preparing this paper for publication.

investment strategy. At each stage during the planning horizon, an investor rebalances her/his portfolio in order to achieve a target mix of asset proportions. The decision variables represent the target percentages for the asset categories. We show that a combination of Tabu Search and Variable Scaling generates global optimal solutions for real world test cases, despite the presence of nonconvexities. Computational results demonstrate that the approach can be applied in a practical fashion to investment problems with over 20 stages (20 years), 100 scenarios, and 8 asset categories. The method readily extends to more complex investment strategies with varying forms of nonconvexities.

Key Words: Dynamic stochastic control, financial modeling, tabu search, variable scaling.

1. Introduction

Efforts to explain financial markets often employ stochastic differential equations to model randomness over time. For example, interest rate modeling has been an active research area for the past decade, resulting in numerous approaches for modeling the yield curve. See Brennan and Schwartz (1982), Hull (1993), Ingersoll (1987) and Jarrow (1995) for details. These equations form the basis for various analytical studies such as estimating the “fair” market value of securities that display behavior conditional on the future path of interest rates. Often, for tractability (Hull 1993), these techniques discretize the planning horizon into a fixed number of time steps $t \in \{1, \dots, T\}$ and the random variables into a finite number of outcomes.

Our goal is to employ a similar discretization of time and randomness. But instead of estimating fair market value, we are interested in analyzing alternative investment strategies over extended planning periods -- 10 or even 20 years. The basic decision problem is called asset allocation. The universe of investments is divided into a relatively

small number of generic asset categories (US stocks, bonds, cash, international stocks) for the purposes of managing portfolio risks. There is much evidence that the most critical investment decision involves selecting the proportion of assets placed into these categories, especially for investors who are well diversified and for large institutions such as insurance companies and pension plans.

In our model, we assume that future economic conditions are presented in the form of a modest number of scenarios. A scenario is described as a complete path of all economic factors and accompanying returns for the assets over the planning horizon. Asset allocation decision are made at the end of each of the T time periods. They cannot be changed during the period, but they can and do respond to the changing state of the world (including the condition of our portfolio) at the beginning of the next period. To keep the problem in a form that can be understood by financial planners and investment managers, we have assumed a decision rule that is intuitive and commonly used -- the fixed-mix rule. Under this rule, the portfolio has a predetermined mix at the beginning of each period -- for instance, 60% stock, 30% bonds, and 10% cash. The portfolio must be rebalanced by selling and buying assets until the proper proportions are attained. This strategy has been used successfully by Towers Perrin (Mulvey 1995) and other long term investment managers. See Perold and Sharp (1988) for a detailed description of the fixed-mix and other dynamic strategies for asset allocation.

In this paper, we show that Tabu Search can be combined with Variable Scaling to solve important cases of the asset allocation problem. These decisions are difficult due to nonconvexities in the objective function. Also, the inclusion of real world considerations, such as taxes and other transaction costs (Mulvey 1993), may cause difficulties for continuous optimization algorithms. In contrast, Tabu Search takes advantage of the discretization of the solution space.

2. The Fixed-Mix Dynamic Control Problem

In this section we provide the mathematical description of the fixed-mix control problem, a dynamic stochastic control model. Other reallocation rules could be employed, such as the life cycle concept; these lead to similar nonconvex optimization problems.

To state the model, we define the following sets:

a set of discrete time steps in which the portfolio will rebalanced:

$$t = \{1, 2, \dots, T\};$$

a set of asset classes:

$$i = \{1, 2, \dots, I\};$$

a set of scenarios, each of which describes the full economic situation for each asset category i , in each time period t :

$$s = \{1, 2, \dots, S\}.$$

Define the following decision variables:

$x_{i,t}^s$: amount of money (in dollars) invested in asset category i , in time period t , for scenario s ;

w_t^s : wealth (in dollars) at the beginning of time period t under scenario s ;

λ_i : fraction of the wealth invested in asset category i (constant across time), $0 \leq \lambda_i \leq 1$, $i = 1, 2, \dots, I$ and $\sum_{i=1}^I \lambda_i = 1$ (not allowing short sales).

Define the following parameters:

w_1 : initial wealth in the beginning of time period 1 (in dollars);

$r_{i,t}^s$: percentage return for asset i , in time period t , under scenario s ;

\bar{r}_t^s : average return in time period t under scenario s ,

$$\bar{r}_t^s = \frac{\sum_{i=1}^I (x_{i,t}^s \cdot r_{i,t}^s)}{\sum_{i=1}^I x_{i,t}^s};$$

p^s : probability that scenario s occurs, $\sum_{s=1}^S p^s = 1$;

t_i : percentage transaction cost for asset class i -- to simplify the presentation, symmetric transaction costs are assumed (i.e. cost of selling equals cost of buying); the implemented model can also handle nonsymmetric transaction costs.

The initial asset allocation will be:

$$x_{i,1}^s = w_1 \lambda_i. \quad (1)$$

The fixed-mix control rule ensures that a fixed percentage of the wealth is invested in each asset category. Wealth at the beginning of the second period will be:

$$w_2^s = \sum_{i=1}^I (1 + r_{i,1}^s) x_{i,1}^s - \sum_{i=1}^I |r_{i,1}^s - \bar{r}_1^s| x_{i,1}^s t_i. \quad (2)$$

The second term of (2) takes the transaction cost into account assuming linear transaction costs. To compute these values, the average return, \bar{r}_t^s , is calculated for each time period t and for each scenario s . A portion of the asset classes with returns higher than the average return is sold, while the asset classes with below average returns are bought.

The equation for rebalancing at the beginning of period t is

$$x_{i,t}^s = w_t^s \lambda_i \quad \forall i \text{ and } s, \text{ and } t = \{2, 3, \dots, T\}, \quad (3)$$

while the equation for wealth at the beginning of period $t+1$ is

$$w_{t+1}^s = \sum_{i=1}^I (1 + r_{i,t}^s) x_{i,t}^s - \sum_{i=1}^I |r_{i,t}^s - \bar{r}_t^s| x_{i,t}^s t_i \quad \forall s \text{ and } t = \{2, 3, \dots, T\}. \quad (4)$$

Typically the model includes linear constraints on the asset mix which we for now assume constant over time:

$$A\lambda - b \leq 0. \quad (5)$$

These constraints can take any linear form, for instance upper and lower bounds on the proportions. A typical example will be for investors to limit the international exposure to some value (say 30%).

The objective function depends on the investor's risk attitude. A multiperiod extension of the Markowitz's mean-variance model is applied for this analysis. Two terms are needed -- the average total wealth, $\text{Mean}(\mathbf{w}_T)$, and the variance of the total wealth, $\text{Var}(\mathbf{w}_T)$, across the scenarios at the end of the planning horizon (at the end of time period T). Other objective functions can also be used. For example, we can maximize the von Neumann-Morgenstern expected utility of the wealth at time T (Keeney and Raiffa 1993). Alternatively, we can maximize a discounted utility function (Ziemba and Vickson 1975). Other objective function forms, such as multiattribute utility functions, can also be used to model multiple investor goals (Keeney and Raiffa 1993).

Our financial planning model is thus

$$\text{Max } Z = \beta \text{Mean}(\mathbf{w}_T) - (1-\beta)\text{Var}(\mathbf{w}_T) \quad (6)$$

$$\text{s.t. } \text{Mean}(\mathbf{w}_T) = \sum_{s=1}^S p^s w_T^s, \quad (7)$$

$$\text{Var}(\mathbf{w}_T) = \frac{1}{S} \sum_{s=1}^S [w_T^s - \text{Mean}(\mathbf{w}_T)]^2, \quad (8)$$

equations (1) - (5), and $0 \leq \beta \leq 1$.

$\text{Mean}(\mathbf{w}_T)$ measures the expected profit from the investment strategy, while $\text{Var}(\mathbf{w}_T)$ measures the risk of the strategy. A tradeoff exists between the expected profit and the risk of an investment strategy. By solving the problem while allowing β to vary between 0 to 1 we obtain

the multiperiod efficient frontier. The quadratic equality constraint (3) causes the fixed-mix model to be nonconvex.

3. Specializations to Tabu Search

An advantage in solving the fixed-mix control problem with Tabu Search is that each application of Variable Scaling effectively discretizes the solution space. We thus define the solution space as a discrete set of possible investment proportions in each asset category. Let us define the following vectors:

$\lambda(i)$: proportion of the total investment in asset category i ;

$l(i)$: lower bound on the investment proportion in asset category i ;

$u(i)$: upper bound on the investment proportion in asset category i .

Restricting the investment proportions in each asset category to be an integer percentage share of the total investment means that we allow $\lambda(i)$ to take the following percentage values:

$$l(i), l(i) + 1, l(i) + 2, \dots, u(i) - 1, u(i),$$

where $0 < l(i) < u(i) < 100$ (not allowing short sale) and $u(i)$ and $l(i)$ are integer.

Tabu Search operates under the assumption that a neighborhood can be constructed to identify adjacent solutions. The fixed-mix problem is constrained such that the sum of all the investment proportions must equal 100%. We define a neighbor solution by choosing two variables, $\lambda(up)$ and $\lambda(down)$ and assigning them new values:

$$\lambda(up) = \lambda(up) + delta, \text{ and}$$

$$\lambda(down) = \lambda(down) - delta,$$

such that $\lambda(up) < u(up)$ and $\lambda(down) > l(down)$, where up is the index of the variable increasing its value, $down$ is the index of the variable decreasing its value, and $delta > 0$ is the stepsize for the neighborhood.

For a given discrete solution space, the following neighborhood search algorithm (Glover and Laguna 1993) is an attempt to solve the

fixed-mix control problem. This approach gives the global optimal solution (in the discrete solution space) if the problem is convex. However, if the problem is nonconvex, as is the case with the fixed-mix control problem, the neighborhood search algorithm will terminate at the first (discrete) local optimal solution. The area of the solution space that has been searched will be limited using this approach.

Algorithm: SIMPLE LOCAL SEARCH

Step 1: Initialization.

Select a feasible starting solution, λ_{start} .

Set $\text{current_solution} = \lambda_{\text{start}}$.

Calculate objective value of this solution, $f(\lambda_{\text{start}})$, and set $\text{best_objective} = f(\lambda_{\text{start}})$.

Step 2: Stopping criterion.

Calculate the objective value of the neighbors of current_solution .

If no neighbors have a better objective value than best_objective , stop.

Otherwise, go to Step 3.

Step 3: Update.

Select a neighbor, λ_{next} , with the best objective value, $f(\lambda_{\text{next}})$ (or select any neighbor that gives an improving objective value).

Update: $\text{current_solution} = \lambda_{\text{next}}$, $\text{best_objective} = f(\lambda_{\text{next}})$

Go to Step 2.

Due to nonconvexities, we need a more intelligent method for finding a global solution. Tabu Search extracts useful information from previous moves to direct the search to productive areas of the solution space. It uses specialized memory structures to characterize moves with certain attributes as tabu, and assigns other moves values or penalties in order to guide the search through the solution space without becoming trapped at local optima.

We define the total neighborhood of the current solution, $N(\text{current_solution})$, as all possible ways of increasing the amount invested in one asset category and decreasing the amount invested in another asset category, while satisfying the constraints. This neighborhood is modified to $N(H, \text{current_solution})$, where H represents the history of the moves in the search process. Therefore, the search his-

tory determines which solutions might be reached from the current point. In short term Tabu Search strategies, $N(H, \text{current_solution})$ is typically a subset of $N(\text{current_solution})$. In intermediate and longer term strategies $N(H, \text{current_solution})$ may contain solutions not in $N(\text{current_solution})$. See Glover (1995) for further details.

3.1 The Memory Structures

Tabu Search makes use of adaptive (recency based and frequency based) memory structures to guide the search. The recency and the frequency memory structures each consist of two parts. The first part of the recency based memory keeps track of the most recent iteration at which each variable received each of the values assigned to it in the past ($\text{tabu_time}(i, \text{value})$). The second part of the recency based memory keeps track of the most recent iteration at which each variable changed its value ($\text{tabu_last_time}(i)$).

The first part of the frequency based memory keeps track of the number of times each variable has been assigned its specific values ($\text{tabu_count}(i, \text{value})$). The second part measures the number of iterations the variables have resided at the assigned values ($\text{tabu_res}(i, \text{value})$). Let $\text{duration}(i)$ measure the number of iterations since the variable received its current value.

3.2 Move Attributes

To make use of these memory structures, Tabu Search uses the concept of move attributes. Generally a move $\lambda_{\text{next}}(\text{up}) = \lambda(\text{up}) + \delta$ and $\lambda_{\text{next}}(\text{down}) = \lambda(\text{down}) - \delta$ has four attributes:

- (1) $\lambda(\text{up})$: value of increasing variable, prior to increase;
- (2) $\lambda_{\text{next}}(\text{up})$: value of increasing variable, after increase;
- (3) $\lambda(\text{down})$: value of decreasing variable, prior to decrease;
- (4) $\lambda_{\text{next}}(\text{down})$: value of decreasing variable, after decrease.

If a move attribute is tabu, this attribute is said to be tabu-active. A

move is classified tabu as a function of the tabu status of its attributes. In the next section we show what combinations of tabu-active move attributes classifies a move tabu.

3.3 Tabu Rules Based on Recency Memory Structure

An important parameter of this process is *tabu_tenure*, that is the number of iterations in which a move attribute is tabu-active. For our approach, *tabu_tenure* will have two values, *t_from* and *t_to*, where *t_from* is the number of iterations an assignment $\lambda_i = value$ is tabu-active, to discourage moving λ_i “from” its present value (*now_value*), and *t_to* is the number of iterations an assignment $\lambda_i = value$ is tabu-active, to discourage moving λ_i “to” a specific value (*next_value*) (Glover and Laguna 1993).

An attribute $(i, value)$ is declared tabu-active when

$tabu_time(i, value) \neq 0, tabu_time(i, value) \geq current_iteration - t$, where $t = t_to$ if $value = next_value$. (This classification will discourage a variable from returning to a value it recently has moved away from.) Also, an attribute i (implicitly, an attribute $(i, value)$ for $value = now_value$) is declared tabu-active when

$tabu_last_time(i) \neq 0, tabu_last_time(i) \geq current_iteration - t$, where $t = t_from$. (This classification will discourage moving a variable from its present value if it recently received this value.)

Both dynamic and static rules can be applied to set *tabu_tenure*. For our purposes, we have used simple static rules, such as setting *t_from* to a constant value between 1 and 3 and *t_to* to a constant value between 4 and 6. Our choice of values for *t_to* and *t_from* is based on preliminary experimentation. We choose $t_to \cong 2\cdot\sqrt{n}$ as a rule of thumb in deciding the size of this parameter (in our test cases $n = 8$).

In the implementation we introduce one other kind of tabu-active classification: an attribute $(i, value)$ is “strongly from tabu-active” if
 $tabu_last_time(i, value) \geq current_iteration - 1$.

For this study, the basic rule for defining a move tabu is if one of the following is true:

- i) either attribute (1) or attribute (3) is strongly tabu-active;
- ii) two or more of the remaining attributes are tabu-active.

3.4 Tabu Rules Based on Frequency Memory Structure

The frequency memory structure is usually applied by assigning penalties and inducements to the choice of particular moves. Let S denote the sequence of solutions generated by the search process up to the present point and define two subsets, $S1$ and $S2$, which contain high quality and low quality solutions (in terms of objective function values), respectively. A high residence frequency in the subsequence $S1$ indicates that an attribute is frequently a participant in high quality solutions, while a high residence frequency in the subsequence $S2$ indicates that an attribute is frequently a participant in low quality solutions. These relationships guide the search in the direction of high quality solutions, by increasing or decreasing the incentive to choose particular moves based on the quality of past solutions that contain attributes provided by these moves. This constitutes an instance of a Tabu Search intensification process. Define another set, $S3$, containing both high and low quality solutions. Assigning a penalty to high frequency attributes in this set pushes the search into new regions, creating a diversification process. A high transition frequency might indicate that an attribute is a “crack filler”, which is an attribute that alternates in and out of the solution to “perform a fine tuning function” (Glover and Laguna 1993).

4. Solution Strategy

Tabu Search methods often do not “turn on” their memory or restrictions until after reaching a first local optimum. In our solution strategy, we take advantage of an efficient method to find the local optimum, namely a Variable Scaling approach. Our Variable Scaling procedure is an instance of a candidate list method, which is the name given to a class of strategies used in Tabu Search (by extension of related procedures sometimes used in optimization) to generate subsets of good moves without the effort of examining all moves possible. In the present setting, where the number of values available to each variable is effectively infinite, we elect a candidate list strategy that "scales" the variables to receive only a relatively small number of discrete values. However, we allow the scaling to be variable, permitting the scaling interval to change each time a local optimum is reached in the restricted part of the neighborhood defined by the current scaling. When no change of scaling (from the options specified) discloses an improving move -- so that the current solution is truly a local optimum relative to the scalings considered -- we apply a recency based Tabu Search approach to drive the solution away from its current location, and then again seek a local optimum by our Variable Scaling approach. The Variable Scaling approach is outlined in section 4.1, while the complete strategy is outlined in section 4.2.

4.1 Variable Scaling Approach

We define a set of stepsizes (or scaling intervals), each of which gives rise to a restricted neighborhood. A local neighborhood search is done over a given restricted neighborhood until no improvement is possible. When local optimality is reached in that neighborhood (for one particular stepsize), we switch to another restricted neighborhood via a new stepsize. We choose a set of stepsizes in decreasing order. (For our purposes, typically the biggest stepsize is 5% and the smallest stepsize is 1%.) The smallest stepsize determines the accuracy of identifying a solution as a local optimum.

For a convex problem, applying such a candidate list approach based on Variable Scaling speeds up the search since we move in bigger steps toward the optimal solution in the beginning of the search, and only as we get “close to the top” (given we are maximizing) do we decrease the stepsize. For a nonconvex problem (like ours) with several local optima, the approach can reduce the number of necessary iterations to get to a local optimum, and it can also move the search away from a local optimum. If one stepsize is “stuck” in a local solution, a change in the stepsize can take us away from the local optimum. All of these ideas are incorporated in *Algorithm*: VARIABLE SCALING.

Algorithm: VARIABLE SCALING

Step 0: Initialize.

Construct a stepsize list: $stepsize_1, stepsize_2, \dots, stepsize_NS$, where NS is the number of stepsizes. Choose stepsizes in decreasing order.

Set $stepsize = stepsize_1$.

Construct an initial solution, λ_start . Let $current_solution = \lambda_start$.

Set $current_iteration = 0$.

Let $best_objective = f(\lambda_start)$, the objective value of λ_start .

Step 1: Calculate neighbors.

Step 2: Evaluate neighbors.

If objective value of at least 1 of the neighbors is better than $best_objective$:

Pick the neighbor with the best objective value, λ_next ;

Update $current_solution = \lambda_next$ and $best_objective = f(\lambda_next)$;

Set $current_iteration = current_iteration + 1$;

Go to Step 1.

If none of the neighbors improves the solution:

If $stepsize \neq stepsize_NS$, use next stepsize from list and go to Step 1;

If $stepsize = stepsize_NS$, check if objective has improved for any of the last NS stepsizes: If the objective has improved, set $stepsize$ to $stepsize_1$, and go to Step 1. If it has not improved, go to Step 3.

Step 3. STOP: Local optimal solution is obtained.

4.2 The Complete Algorithm

The implemented algorithm is:

Algorithm: COMPLETE ALGORITHM

Step 0, Step 1 and Step 2 from the Variable Scaling approach.

Step 3: Incorporate Recency Based Tabu Search (diversifying search).

3.1: Set $stepsize = stepsize_div$ ($stepsize$ used for the diversifying search).

Set $div_it_counter = 0$.

3.2: Set $div_it_counter = div_it_counter + 1$.

For all neighborhood moves do

check if move is tabu.

if tabu, go to next neighbor;

if not tabu, calculate the objective value of the move.

3.3: Pick the (nontabu) neighbor with the best objective value. Denote this neighbor λ_{best_div} , and its objective value $f(\lambda_{best_div})$.

If $f(\lambda_{best_div}) > best_objective$ set $stepsize = stepsize_I$ and go to Step 1 in the Variable Scaling approach.

If $f(\lambda_{best_div}) < best_objective$ go to Step 3.4.

3.4: If $div_it_counter < max_div_it$, update tabu status (as done in Steps 1-3 in memory updating in section 3.1) and go to Step 3.2.

If $div_it_counter = max_it_counter$, go to Step 4.

Step 4: STOP. The solution is equal or sufficiently close to the true (discrete) global optimal solution.

The procedure imposes the Tabu Search approach in conjunction with the candidate list strategy of Variable Scaling. This is just one of many ways that Tabu Search can be coordinated with a candidate list strategy. (More typical is to subject such a strategy directly to the guidance of Tabu Search memory, rather than invoking this guidance only at particular stages and for particular neighborhood instances.) Nevertheless, we have found this approach convenient for our present purposes. Specifically, in this alternating approach, we switch from the Variable Scaling method (over its chosen set of scalings) to the Tabu Search method (over another set of scalings) whenever Variable Scaling no longer improves the current solution. If Tabu Search generates a solution better than the one obtained from the Variable Scaling, we return to Variable Scaling, with the solution from Tabu Search as an initial solution. If Tabu Search fails to generate an improved solution, the algorithm will stop when the maximum number

of iterations is reached. In this procedure the choice of the stepsizes is crucial for the algorithm's success. We typically choose small stepsizes in the range of 0.5% to 5% for the Variable Scaling, while a bigger one, typically 5% to 15% is applied for the Tabu Search.

5. Computational Results

The algorithm described in section 4.2 is applied to an investment problem with $I = 8$ asset categories, $T = 20$ time periods and $S = 100$ scenarios. The 8 different asset categories are cash equivalents, Treasury bonds, large capitalization US stocks, international bonds, international stocks, real estate, government/corporate bond index, and small capitalization US stocks. One hundred scenarios were generated by the technique introduced in Mulvey (1995). Each scenario is given equal probability $p_s = 1/S = 1\%$. Each scenario consists of returns for each asset, in each time period. Hence the total number of returns generated is equal to 16000. The initial wealth is set equal to unity.

In all the experiments, each point on the multiperiod efficient frontier is obtained as explained in section 4.2. The entire efficient frontier is obtained by solving the problem for 22 values of β . Through all the experiments, we have a set of basic test parameters¹.

5.1 Comparison with Global Method

Our solutions are compared with the solutions obtained from the global method described in Androulakis *et al.* (1994). This deterministic global optimization algorithm guarantees finite ε -convergence to the global optimum. In this case we assume no transaction cost or tax. The solutions obtained by our approach are very close to the solutions obtained by the global method and are often slightly better. The opti-

¹ Number of stepsizes for the Variable Scaling approach (NS): 3; Variable Scaling stepsizes $stepsize_1 = 5\%$, $stepsize_2 = 3\%$, $stepsize_3 = 1\%$; stepsize for the Tabu Search part of the algorithm ($stepsize_div$): 10%; maximum number of diversifying steps (max_div_it): 25; tabu tenures: $t_from = 2$, $t_to = 4$.

mal mixes generated by the two methods are indeed practically identical. These results are obtained despite the presence of nonconvexities.

5.2 Including Transaction Costs and Tax

We assumed percentage transaction costs for the asset categories². The tax rate is assumed to be 28%. Figure 1 shows the efficient frontiers obtained by solving the model that takes tax and transaction cost in to account (tax-model) and the one that does not (no-tax model). As expected, the efficient frontier obtained by the tax-model is tilted down. More surprisingly, perhaps, the efficient frontiers cross in the low end of the variance area (for low β 's). Transaction costs and taxes dampen the variance of the expected value and therefore a lower variance can be achieved for the tax-model. From Figure 2 we see that for $\beta > 0.94$ in the model where transaction costs and tax is included, it is optimal to have all the funds in the asset category with the highest expected value. This is in contrast to the case without taxes or transaction costs, where a more diversified optimal solution was found as soon as the variance was assigned a weight in the objective function.

There is an explanation for this phenomenon. When all of the funds are concentrated in one asset category, no buying or selling is necessary to update the portfolio at the end of each time period. When the funds are split between asset categories, however, trading must be done at the end of each time period to reconstruct the portfolio to the predetermined weights of the fixed-mix rule. This implies cash outflows because of the transaction costs and taxes on assets sold for profit. So for $\beta > 0.94$ the gain from reduced variance by diversifying is offset by a larger loss in the expected value.

² Cash equivalents: 0%; Treasury bonds: 0.25%; large capitalization stocks: 0.25%; international bonds: 0.5%; international stocks: 1.0%; real estate: 6.0%; government/corporate bond index: 1.0%; small capitalization stocks: 1.0%.

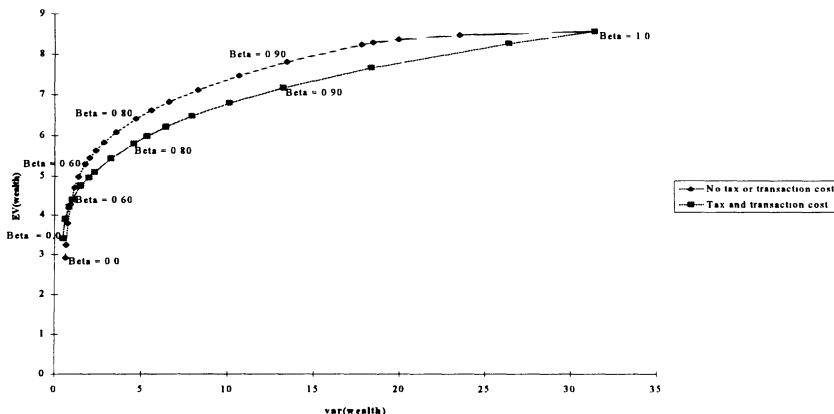


Figure 1: Efficient frontiers for model with and without tax and transaction costs. Notice how the efficient frontier for the tax-model is tilted down (as one would expect). Also see how the efficient frontiers crosses in the low variance end.

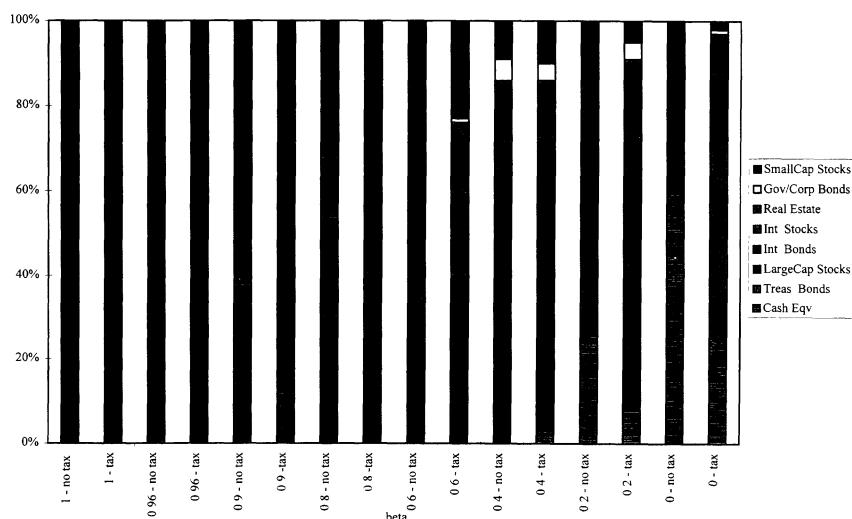


Figure 2: Optimal solutions for model with and without tax and transaction costs. Notice how the solutions obtained from the tax-model are less diversified for high β 's (i.e. in the high variance/high expected value end of the efficient frontier), and more diversified for low β 's.

At the other end of the efficient frontier (low β , i.e. low variance), we see that tax-model recommends solutions that are more diversified than the solutions from the no-tax model. Tax-model recommends a portfolios consisting of all 8 asset categories for $\beta \leq 0.4$, with no dominating asset category (except for $\beta = 0$). The no-tax model, however, recommends a portfolio concentrated in fewer asset categories. There is a simple explanation for this result. With many asset classes, the return in some of asset classes will be close to the average return. Little trading is required in these asset categories to rebalance the portfolio to the prefixed weights. More trading is done in the asset categories with returns far from average, but these asset categories are a fraction of our total portfolio. This is not the case for a portfolio concentrated in fewer asset categories. Consider, for instance, the portfolio recommended by the no-tax model for $\beta = 0.2$ -- approximately 80% in cash equivalents and Treasury bonds. Since it is unlikely that both dominating asset categories have close to average returns, more trading is needed to rebalance the portfolio. This portfolio is likely to have more cash outflows from transaction costs and taxes than the more diversified one -- the gain in reduced variance by concentrating the portfolio in cash equivalents and Treasury bonds is offset by a loss in expected value caused by the increase in cash outflows.

As expected, the solution time increases considerably when transaction costs and taxes are included to the model. The average solution time per β -problem for the model with basic test parameters (see beginning of this section) is 46.7 CPU seconds (just over 17 CPU minutes to obtain the entire efficient frontier) on a Silicon Graphics Iris workstation. The results are highly attractive, particularly given that we are developing a long term projecting system.

5.3 The Effect of Including Tabu Search Restrictions.

To see the effect of the Tabu Search restrictions we consider the tax

problem and compare the solutions of our approach (COMPLETE ALGORITHM) with the solutions of a pure Variable Scaling approach (the candidate list strategy used in our method). The largest stepsize in the pure Variable Scaling approach is set equal to the stepsize of the Tabu Search part of our algorithm (*stepsize_div* = 10%). The other stepsizes are also set equal for both approaches (5%, 3% and 1%). Hence, the sole difference between the two approaches is the recency memory restrictions on the 10% search applied in our algorithm. For $\beta > 0.5$ the solutions are identical. The pure Variable Scaling approach “gets stuck” in a local solution for $\beta < 0.4$, in contrast to the approach that includes Tabu Search memory guidance.

6. Conclusions

Tabu Search is an efficient method for obtaining the efficient frontier for the fixed-mix investment problem. The computational results show that the solutions obtained by the Tabu Search are very close to (often slightly better than) the ϵ -tolerance global optimal solutions obtained by the method of Androulakis *et al.* (1994) for the case with no taxes or transaction costs. In an expanded model, which addresses transaction costs and taxes for greater realism, our approach continues to obtain optimal solutions efficiently. The expanded model is beyond the capability of the global optimization approach, and its complicating features in general pose significant difficulties to global optimization solvers based on currently standard designs.

Some areas for future research are: (1) develop and test related dynamic stochastic control strategies (with nonconvexities); (2) design an approximation scheme for updating information between iterations in order to improve computational efficiencies; and (3) incorporate additional strategic elements of Tabu Search. Since the discretization of the solution space depends upon an investor’s circumstances, research in this area is critical for successful use of this methodology.

7. References

- I.P. Androulakis *et al.* Solving stochastic control problems in finance via global optimization, Working paper, SOR-94-01, Princeton University (1994).
- M.J. Brennan and E.S. Schwartz, An equilibrium model of bond pricing and a test of market efficiency", *Journal of Financial and Quantitative Analysis*, 17 (1982) 75.
- F. Glover, Tabu search: fundamentals and usage, Working paper, University of Colorado, Boulder (1995).
- F. Glover and M. Laguna, Tabu search, in: *Heuristic Techniques for Combinatorial Problems*, ed. C. Reeves (1992).
- J.C. Hull, *Options, Futures and other Derivative Securities*, (Prentice Hall, 1993).
- J.E. Ingersoll, Jr. *Theory of Financial Decision Making*, (Rowman & Littlefield, 1987).
- R. Jarrow, Pricing interest rate options, in: *Finance*, eds: R. Jarrow, V. Madsimovic, and W.T. Ziemba, (North Holland, Amsterdam, 1995).
- R. Keeney and H. Raiffa, *Decisions with Multiple Objectives: Preferences and Value Tradeoffs*, (John Wiley, New York, 1976; reprinted by Cambridge University Press, 1993).
- J.M. Mulvey, Incorporating transaction costs in models for asset allocation, in: *Financial Optimization*, ed: S. Zenios, (Cambridge University Press, 1993).
- J.M. Mulvey, Generating scenarios for the Towers Perrin investment system, Working paper, SOR 95-04, Princeton University (to appear *Interfaces* 1995).
- A.F. Perold and W.F. Sharpe, Dynamic strategies for asset allocation, *Financial Analysts Journal*, (1988) 16.
- W.T. Ziemba and R.G. Vickson (eds.), *Stochastic Optimization Models in Finance*, (Academic Press, New York, 1975).

Comparison of Heuristics for the 0-1 Multidimensional Knapsack Problem

Authors: S. HANAFI, A. FREVILLE, A. EL ABDELLAOUI
Université de Valenciennes, LIMAV
Le Mont Houy - B.P.311 59304 VALENCIENNES CEDEX
e.mail: (hanafi,freville,elabdell@univ-valenciennes.fr)

Abstract:

The multidimensional 0-1 knapsack problem (0-1MKP) is a generalization of the well-known 0-1 knapsack problem. As for any hard optimization problem also for 0-1MKP, a reasonable effort to cope with the problem is trying to derive heuristics which solve it suboptimally and which, possibly, yield a good trade-off between the solution quality and the time and the memory requirements. In this paper, we describe several heuristics for 0-1MKP. The first one is a simple multistage algorithm (SMA) which always maintains feasibility requirements. We also propose variants of the tabu search dealing with infeasibility, called also tunneling effect. We implement these heuristics as C code and compare their performances.

Keywords: 0-1 multidimensional knapsack, heuristics, multistage method, tabu search, tunneling effect.

1. Introduction

The 0-1 multidimensional knapsack problem (0-1MKP) is a knapsack with multiple resource constraints which is a special case of general 0-1 integer programming, first encountered in the areas of capital budgeting and resource allocation. An update and comprehensive survey for 0-1MKP dealing with applications, complexity and heuristics can be found in Fréville (1995). For a survey on further problems in the knapsack area the reader is referred to Martello and Toth (1990).

Formally, 0-1MKP may be defined as follows:

$$\begin{aligned} \text{Max } & \sum_{j=1}^n c_j x_j \\ \text{subject to: } & \sum_{j=1}^n A_{ij} x_j \leq b_i, \quad \forall i \in \{1, \dots, m\}, \\ & x_j \in \{0, 1\}, \quad \forall j \in \{1, \dots, n\}, \end{aligned}$$

where n is a number of items and m is a number of knapsack's constraints with capacities b_i ($i=1, \dots, m$), associated weights A_{ij} ($i=1, \dots, m$; $j=1, \dots, n$) and the profits c_j ($j=1, \dots, n$). The objective is to find a subset of the set of items that yield a maximum profit. We assume that all data c_j , A_{ij} and b_i to be non-negative integer.

As for any hard optimization problem and also for 0-1MKP, a reasonable effort to cope with the problem is trying to derive heuristics or metaheuristics which solve it suboptimally and which, possibly, yield a good trade-off between the solution quality and the time and the memory requirements.

A simple multistage algorithm (SMA) is presented in section 2 which maintains feasibility requirements along the process. Section 3 describes variants of the tabu search dealing with infeasibility. In section 4 computational results are reported on test problems from the literature.

2. A Simple Multistage Algorithm

It's now a common and popular scheme to combine different heuristic principles such that greedy search, sampling method or stochastic

process. It is the case of the method of Fréville and Plateau (1982,1986,1994,1995), called AGNES, for which a lot of numerical experiments shows its efficiency for 0-1MKP instances. Within a reduction scheme, AGNES method is composed of greedy steps bringing into play the concepts of surrogate relaxation, oscillating assignment and strongly determined variables, first introduced by Glover (1977,1978). Additional steps such as complementing and partial implicit enumeration complete the method.

We present in this section a simple multistage algorithm for solving 0-1MKP, which incorporates different heuristic principles in a flexible fashion. The first step of SMA can be viewed, according to tabu search terminology, as a diversification process. It is realized by generating random solutions (or strings) in $\{0,1\}^n$ plus the two "extreme" solutions $(0,0,\dots,0)$ and $(1,1,\dots,1)$. Then there are used as starting points for a neighborhood search. Feasibility is maintained all along the process, hence a projection onto the feasible domain is needed if the initial solution violates at least one constraint.

The neighborhood search can be performed in various ways which allows flexibility to SMA. Numerical experiments reported in section 4 compare the quality of different procedures: a pure "greedy" search based on a dual multiplier u , nonmonotonic ascent methods such as *Simulated Annealing* (SA), or *Threshold Accepting* algorithm (TA) and the *Noising Method* (NM).

As an intensification process, an additional step based on repeated calls of the greedy search tries to improve the current feasible solution. Figure 1 describes the main structure of SMA.

```

procedure SMA
comment: SMA provides a feasible solution  $\underline{x}$ 
Generate a set P with S random binary strings
 $x^s \in \{0,1\}^n$ ,  $s = 1,\dots,S$ ;
Add to P the two special strings  $x^0 = (0,0,\dots,0)$ 
and  $x^{S+1} = (1,1,\dots,1)$ ;
 $\underline{x} := x^0$ ;
for  $s = 0$  to  $S+1$  do
  if  $x^s$  violates at least one of the constraints ( $x^s$  is infeasible)
  then projection( $x^s$ ) endif
  local_search( $x^s$ );
  repeated_drop_add( $x^s$ );

```

```

| if  $cx^S > cx$  then update the best current feasible solution  $\underline{x}$  with  $x^S$ 
| endif
| endfor

```

Figure 1 : Skeleton for SMA

The procedure **projection** is based on the dual greedy heuristic of Senju and Toyoda (1968). At each iteration one variable is fixed at 0 according to a penalty factor. The procedure stops as soon as the feasibility requirements are satisfied.

procedure projection(x)

comment: starting with an infeasible solution x , *projection* generates a feasible solution x according to the greedy mechanism of *Senju-Toyoda*

feasible:= false;

calculate the indicator Δ associated to the resources consumed by x :

for $i = 1$ to m do $\Delta_i = \sum_{j=1}^n A_{ij}x_j - b_i$ endfor;

determine the subset $I(x)$ of constraints violated by x :

$I(x) = \{ i \mid \Delta_i > 0, i = 1, \dots, m \}$;

while not feasible do

choose $j^* = \arg \min \{ \frac{c_i}{v_j} \mid x_j = 1, j = 1, \dots, n \}$ where v_j is a

penalty factor of component j such that $v_j = \sum_{i \in I(x)} A_{ij}\Delta_i$;

$x_{j^*} := 0$;

update Δ : for $i = 1$ to m do $\Delta_i := \Delta_i - A_{ij^*}$ endfor;

update $I(x)$; if $I(x) = \emptyset$ then feasible := true endif

endwhile

Figure 2 : Skeleton for projection

The procedure **add** generalizes the most well-known greedy algorithm used for the knapsack problem by replacing the m constraints A_i by a surrogate constraint uA (figure 3). The nonnegative multiplier u can be derived in a lot of ways. It may be the optimal solution of any dual problem associated to the primal problem (P), or more simply, derived by direct calculations involving structural informations provided by the entries, Fréville and Plateau (1986).

procedure add(\underline{x})

comment: Dealing with 0-1MKP subproblem defined by the variables fixed at 0 in \underline{x} , *add* tries to improve the initial feasible solution \underline{x} by fixing at 1 variables according to decreasing order of the

$\frac{c_j}{uA_j}$ ratios, as far as possible ($uA_j = \sum_{i=1}^m u_i A_{ij}$, $u_i \geq 0$
 $i=1, \dots, m$).

$q :=$ cardinality of $J_0(\underline{x}) = \{ j \mid \underline{x}_j = 0 \}$;

Sort the indices j of $J_0(x)$ such that: $\frac{c_{j1}}{uA_{j1}} \geq \dots \geq \frac{c_{jk}}{uA_{jk}} \geq \dots \geq \frac{c_{jq}}{uA_{jq}}$;

$x := \underline{x}$;

for $k = 1$ to q **do**

if $A_{i,jk} + \sum_{j=1}^n A_{ij}x_j \leq b_i \quad \forall i = 1, \dots, m$ **then** $x_{jk} := 1$ **endif**

endfor

if $c_x > c_{\underline{x}}$ **then** update the best current feasible solution \underline{x} with x **endif**

Figure 3 : Skeleton for add

Several numerical experiments in the literature have shown the usefulness of performing a complementing phase to improve the current feasible solution (e.g. Balas and Martin (1980), Lee and Guignard (1988), Fréville and Plateau (1994). This intensification step is performed by the procedure **repeated_drop_add** described in figure 4.

procedure repeated_drop_add(\underline{x})

comment: *repeated_drop_add* tries to improve an initial feasible solution \underline{x} by applying the procedure *add* starting from neighbors of \underline{x} repeatedly.

Consider the subset $J_1(\underline{x})$ of components fixed at 1 in \underline{x} :

$J_1(\underline{x}) = \{ j \mid \underline{x}_j = 1, j = 1, \dots, n \}$;

$x\# := \underline{x}$;

for all j in $J_1(\underline{x})$ **do**

$x := x\#$; $x_j := 0$;

add(x);

if $c_x > c_{\underline{x}}$ **then** update the best current feasible solution \underline{x} with x **endif**

endfor

Figure 4 : Skeleton for repeated_drop_add

We now briefly describe three metaheuristics which can be used as the local search in SMA.

Drexel (1988) studied *Simulated Annealing* for solving 0-1MKP approximatively. While a greedy algorithm generates a sequence of solutions with increasing objective value, SA allows the objective value to decrease occasionally to avoid getting stuck at a local optimum. To deal with the inequality constraints, Drexel introduces a special 2-exchange random move which maintains the feasibility of all the solutions generated during the process. The computational results demonstrate that such a flexible move is an useful concept for an SA framework. Another strategy, used by Battiti and Tecchiolli (1995), is to introduce a penalty function which transforms 0-1MKP into an unconstrained problem. A suitable way is that the penalty function is proportional to the degree of violation of the constraints. In this case the random move is straightforward and does not preserve feasibility. Numerical experiments report that SA performance varies greatly with 0-1MKP settings and is worse than suitable TS implementation (see also Osman (1995) for a similar study with the Generalized Assignment Problem which is a connected problem).

Recently three general combinatorial optimization techniques, "Threshold Accepting", "Great Deluge" and "Record-to-Record travel", have been developed by Dueck and Scheurer (1989,1990) and applied to various optimization problems. Threshold Accepting is formally very similar to SA. TA accepts a tentative solution as a new temporary solution if it does not deteriorate the current value of the objective function too much. In contrast, SA accepts worse solutions only with rather small probabilities. Dueck and Wirshing (1989) shows that TA is faster and gives better results than SA for 0-1MKP. Figure 5 gives the main common structure of SA and TA.

procedure nonmonotonic_local_search(\underline{x})

comment: *nonmonotonic_local_search* tries to improve an initial feasible solution \underline{x} with repeated moves which are not always improving ones.

Choose initial parameters T and r;

change:= true;

$\underline{x} := \underline{x}_0$;

while change do

 change:= false;

```

repeat r times
  move(x,x');
  acceptance(x,x',T,change);
  if cx > cx then update the best current feasible solution x
  with x endif
  update T and r
endwhile

```

Figure 5: Skeleton for nonmonotonic_local_search

Let us now specify the differences between SA and TA. The parameter T is called the temperature (respectively threshold) in SA (resp. TA). The procedure **move(x,x')** generates a new feasible solution x' in the neighborhood of x . SA uses a stochastic version combining simple change and 2-exchange among x -variables or components, Drexel (1988). In contrast, TA proceeds completely deterministically, Dueck and Wirshing (1989).

TA accepts every new solution x' which is not much worse than the old one x (figure 6):

```

procedure TA_acceptance(x,x',T,change)
if  $\delta = cx' - cx > -T$  then  $x := x'$ ; change := true endif

```

Figure 6 : TA_acceptance

SA accepts worse solution only with rather small probabilities (figure 7):

```

procedure SA_acceptance(x,x',T,change)
if  $\delta = cx' - cx > 0$  then  $x := x'$ ; change := true
else generate random variable  $\gamma$  uniformly distributed in  $]0,1[$ ;
  if  $\gamma < e^{\delta/T}$  then  $x := x'$ ; change := true endif
endif

```

Figure 7 : SA_acceptance

Charon and Hudry (1993a,1993b) have recently proposed a new general heuristic, called *Noising Method*, who applied to two optimization problems, the Clique Partitioning of a graph and the Traveling Salesman Problem. The main idea of NM is to introduce

noise to the data to overcome local optimality. Starting with an initial solution, NM repeats the following operations: firstly add a noise to the data in order to change the values taken by the objective function, secondly apply a descent step to the current solution for the noised data. At each iteration, the extent of the added noise decreases until it reaches 0 at last iteration. The main structure of this method applied to 0-1MKP is described in figure 8.

```
procedure noising_search(x)
comment: noising_search tries to improve the initial feasible solution x
          by applying a greedy search to noised data.

Choose the initial noise  $\delta$  and the maximum number of iterations  $r$ ;
x:=x;
repeat  $r$  times
  add_a_noise( $\delta$ );
  local_search(x);
  if  $c_x > c_{\bar{x}}$  then update the best current feasible solution x
    with x endif ;
  update  $\delta$  such that  $\delta$  decreases towards 0 during the process.
```

Figure 8 : Skeleton for noising_search

In this first variant of the noising method, the noise is just added to the objective function (figure 9). Hence no additional work is needed to maintain feasibility.

```
procedure add_a_noise( $\delta$ )
for  $j = 1$  to  $n$  do
  generate random variable  $\rho_j$  uniformly distributed in  $] -1, 1 [$ ;
   $c_j := c_j + \rho_j \delta$ 
endfor
```

Figure 9: Procedure add_a_noise

The noise δ is initialized with δ° such that $\delta_j^\circ = \alpha c_j$ ($\alpha \in]0, 1[$). At each iteration δ decreases by using the following rule: $\delta := \delta - \frac{\delta^\circ}{r}$.

Hence the noise δ is equal to 0 at the last iteration of the loop. In the following numerical experiments the chosen local_search is repeated_drop_add procedure, but it can be replaced by any other

method. It can be pointed out that when using **repeated_drop_add** procedure, adding a noise to the objective function only is nothing else than changing the initial order by which the variables x_j are fixed at 1. Another characteristic of the current version of NM tested in this work is that at each iteration **local_search** starts with the last generated solution. An alternative is to restart always with the initial feasible solution \underline{x} . The impact of various strategies dealing with infeasibility (which are described in section 3 in the context of TS) is also investigated by Fréville, Hanafi and El Abdellaoui (1995a).

3. Variants of tabu search

The term *Tabu Search* was first coined by Glover (1986,1989,1995) who has derived its modern form. Seminal ideas of the method are also developed by Hansen (1986) in a steepest ascent/mildest descent formulation. This method is certainly one of the most popular metaheuristic and has enjoyed successes in a great variety of problem settings (see e.g. Glover and Laguna (1993) for a comprehensive survey and Thiel and Voss (1994) for combining approaches with other metaheuristics). In contrast with "myopic procedure", a fundamental characteristic of TS is by using flexible memory for taking advantage of history to guide the search in the solution space. Some of the principal features of TS that are most responsible for its successes are examined in Glover (1995).

Dammeyer and Voss (1993) performed a TS version proposed by Glover (1990), called the *Reverse Elimination Method*, for solving 0-1MKP. *REM* defines the dynamic way by which the tabu list is managed. They show in particular that SA may be outperformed by *REM* with respect to various criteria. We present a TS approach, called FTS/HFE, which uses the same multi-attribute move as stated in Dammeyer and Voss (1991). This move is called **feasible_drop_add** (figure 10) and was firstly introduced by Fréville and Plateau (1986).

```

procedure feasible_drop_add( $\underline{x}$ )
comment: feasible_add_drop defines a move from a feasible solution  $\underline{x}$ 
          to a feasible neighbor of  $\underline{x}$ .
choose  $j^* = \arg \min \{ \frac{c_j}{A_{j^*j}} \mid x_j = 1, j=1,\dots,n \}$  with  $i^*$  being a scarce
resource such that  $i^* = \arg \min \{ [b_i - \sum_{j=1}^n A_{ij}x_j] / b_i \mid i=1,\dots,m \} ;$ 
 $x_{j^*} := 0$ ; feasible := true;
while feasible do
  choose  $k^* = \arg \max \{ c_j \mid x_j = 0, j=1,\dots,n, j \neq j^* \}$  with

```

```


$$A_{ik^*} + \sum_{j=1}^n A_{ij}x_j \leq b_i, i=1,..,m;$$

if  $k^*$  exists then  $x_{k^*} := 1$  else feasible := false endif
endwhile

```

Figure 10: feasible_drop_add procedure

The principal features of FTS/HFE method, common to the infeasible variant IFTS/HFE introduced below, are now briefly described:

- Tabu list management is static with a length up to 24 and a circular list implementation. The selected move attributes are the indices of the dropped variable and the last added variable during the **feasible/infeasible_drop_add** procedure.
- A simple aspiration criterion is defined during the add phase of the **feasible/infeasible_drop_add** procedure. A move is deemed acceptable if it can attain a new best possible solution.
- The starting solution for each problem is provided by a greedy local search as the **add** procedure.
- The frequency-based memory is contained into two vectors (H^+ and H^-) of size n . They keep track all the transitions performed during the moves: $H^+(j)$ (respectively $H^-(j)$) is incremented by 1 if the variable x_j is added (resp. dropped).
- Intensification and diversification set up as soon as no improvement occurs during a fixed number of iterations. Intensification strategy performs a local search starting from the best feasible solution. The solution space is then reduced by fixing variables associated to the highest values in H^+ and H^- . In contrast, diversification strategy tries to drive the search into new regions by fixing variables of the current solution associated to the lowest values in H^+ and H^- .
- The stopping criterion is at most $20 \cdot n$ iterations (iteration = call of the **feasible/infeasible_drop_add** procedure).

An other TS method, called *Reactive Tabu Search*, where the move is a simple exchange of one variable, has been tested by Battiti and Tecchiolli (1994). The reported computational experiments with *RTS* demonstrated a satisfactory performance compared with Genetic Algorithms, Neural Networks and Simulated Annealing. All these TS schemes maintained feasibility of the solutions which are generated along the process.

Two other TS approaches were designed with the ability to solve more general models than the 0-1MKP. Both methods relaxed the

integer requirements ($x \in \{0,1\}^n$). For solving general 0-1 integer programming problems, Aboudi and Jörnsten (1994) implemented several elements of strategies proposed by TS method which employed the Pivot and Complement heuristic (Balas and Martin 1980) as a subroutine. Glover and Lokketangen (1994a,1994b) performed a TS approach for solving general 0-1 mixed integer programming problems that exploited the extreme points property of zero-one solutions.

An usual attempt to overcome the difficulty to satisfy resource requirement ($Ax \leq b$) is to introduce a penalty factor in the objective value which transforms 0-1MKP into an unconstrained problem. In general this penalty factor is proportional to the degree of violation of the constraints and then favor feasible solutions (Gendreau, Hertz and Laporte 1995). Yet recent advances summarized in Glover (1995) allow new mechanisms dealing with infeasibility, also called *tunneling effect*. In particular, Glover and Kochenberger (1995) obtained computational results of high quality with a new TS approach employing a flexible memory structure that integrated recency and frequency information keyed to "critical events" of the search process. Their TS method consists in a strategic oscillation scheme which alternates between "constructive" and "destructive" phases and drive the search to variable depths on each side of the feasibility boundary.

We also introduce a such infeasibility TS strategy, called IFTS/HFE, by considering the following modification of the **feasible_drop_add** move (figure 11). Starting with a feasible solution x , a move(x, x') is then defined by the **infeasible_drop_add(x)** procedure alone if the solution x' is feasible, or followed by the **projection(x')** procedure otherwise.

procedure infeasible_drop_add(x)

comment: *infeasible_drop_drop* defines a move from a feasible solution x to a neighbor of x which is not necessarily feasible.

choose $j^* = \arg \min \left\{ \frac{c_j}{A_{j^*}} \mid x_j = 1, j=1,\dots,n \right\}$ with i^* being a scarce

resource such that $i^* = \arg \min \left\{ [b_i - \sum_{j=1}^n A_{ij}x_j] / b_i \mid i=1,\dots,m \right\}$;

$x_{j^*} := 0$; near_feasible := true;

while near_feasible do

```

choose  $k^* = \arg \max \{ c_j \mid x_j = 0, j=1,\dots,n, j \neq j^* \}$  with  $x+e_{k^*}$ 
satisfying near-feasible requirements ( $x+e_{k^*}$  and  $x$  are identical
solutions except the  $k^*$ -component fixed at 1 in  $x+e_{k^*}$ );
if  $k^*$  exists then  $x_{k^*} := 1$  else near_feasible := false endif
endwhile

```

Figure 11: infeasible_drop_add procedure

Various ways are able to control the infeasibility of the generated solutions.

- Supplementary resource: the generated solutions can violate the constraints within a fixed amount ϵ of supplementary resource: choose $k^* = \arg \max \{ c_j \mid x_j = 0, j=1,\dots,n, j \neq j^* \}$ with $A_{ik^*} + \sum_{j=1}^n A_{ij}x_j \leq b_i + \epsilon_i$, $i=1,\dots,m$. The additional term ϵ decreases to 0 along the process.
- Use of a surrogate constraint: infeasibility of the generated solutions is controlled within a surrogate constraint associated to a dual multiplier u (structural, Lagrangean or surrogate): choose $k^* = \arg \max \{ c_j \mid x_j = 0, j=1,\dots,n, j \neq j^* \}$ with $uA_{ik^*} + \sum_{j=1}^n uA_{ij}x_j \leq ub$.
- Violated constraints permutation: the subset of constraints being allowed to be not satisfied is changed in a deterministic fashion periodically, by example every p iterations ($p \geq 1$).

The preliminary numerical results reported in section 4 indicate that the "violated constraints permutation" is the most promising variant.

4. Numerical experiments

All the procedures have been coded in C (using the Turbo-C Compiler) and implemented on a PC 486. The following tables report preliminar computational experiments with a test set of 54 problem instances due to Fréville and Plateau (1982,1990) with well-known optimal solutions. All the running times are given in seconds.

4.1 Multistage heuristics

Table 1 reports the performances of SMA with four variants of local search (GA = Greedy Algorithm; SA = Simulated Annealing; TA = Threshold Accepting; NM = Noising Method) and of AGNES. The results are reported in terms of the relative percentage deviation and CPU time.

SMA is obviously the most effective with TA as local search, both in computational time and deviation from optimality. SMA/TA finds an optimal solution in 39 of the 54 problems tested and the relative percentage deviation does not exceed 1.5% in the worst case. Although better results were reported in the literature with the same set of test problems, SMA/TA is very simple to implement and provides to solve large-scale instances an attractive trade-off between required CPU time and quality of the solution.

AGNES is a more sophisticated multistage heuristic which uses three different initial multipliers. The results shown in Table 1 are the best solution found for each problem. Across the 54 test problems, AGNES reports the optimal solution 52 cases (Fréville 1991). The procedure only fails to find the optimal solution with PET7 as a stress instance, and its reduced form FHP2. In addition, AGNES was tested on several randomly generated problems ranging in size up to 500 variables and 30 constraints. Promising results were obtained within a very attractive computational time (5 CPU minutes in the worst case).

Table 1: Average relative percentage deviation and CPU time of the multistage heuristics.

Multistage Procedures	Average Deviation from Optimality (%)	Average CPU-Time (seconds)	# of Optimal Solutions Found
SMA/GA ¹	0.42	0.14	15
SMA/NM ¹	0.45	30.68	26
SMA/SA ¹	0.49	18.05	27
SMA/TA ¹	0.08	4.27	39
AGNES ²	0.02	1.30	52

1 : C - PC 486 (66 Mhz)

2 : Fortran 77 - SUN 3/50 (68020; 16.6 Mhz)

4.2 Tabu search approaches

In Table 2 the compared algorithms are: the Dammeyer and Voss (1993) heuristic, FTS/DV; the feasible variant of our heuristic, FTS/HFE; the Aboudi and Jornsten (1994) heuristic, RFTS/AJ; the Glover and Lokkentengen (1994a,1994b) heuristic, RFTS/GL; the Glover and Kochenberger (1995) heuristic, IFTS/GK; the infeasible variant of our heuristic, IFTS/HFE. The first two TS approaches maintain feasibility along the process while the four other ones allow infeasibility. For each method and for each problem, the solutions

given in Table 2 are the best solutions found from different versions of the method and/or starting points and/or parameter settings.

The feasible version FTS/HFE finds the optimum in 51 out of 54 problems. These results are at least as good as the three other TS approaches which maintain feasibility along the process. Yet preliminary results performed with IFTS/HFE on these test problems of moderate size confirm the promising results of Glover and Kochenberger (1995) and the attractive behaviour of heuristics using mechanisms dealing with infeasibility. Glover and Kochenberger found optimal solutions for each of the 54 test problems, and also produced feasible solutions of high quality with large-scale randomly generated problems ranging in size up to 500 variables and 25 constraints. As Glover and Kochenberger, IFTS/HFE also successfully obtained optimal solutions for each of 54 test problems.

Table 2: Relative percentage deviation of different TS approaches.

Tabu Search Approaches	Average Deviation from Optimality (%)	# of Optimal Solutions Found
FTS/DV ¹	0.07	43
FTS/HFE ¹	0.06	50
RFTS/AJ ²	0.05	47
RFTS/GL ²	0.28	49
IFTS/GK ³	0	54
IFTS/HFE ³	0	54

1 Feasibility maintained along the process

2 Resource requirements maintained along the process

3 Integer requirements maintained along the process

5. Conclusion

In this paper we report the implementation of multistage algorithms and a new tabu search method. The multistage approach yields an attractive trade-off between the solution quality and the time and the memory requirements. On the other hand, our preliminary numerical results confirm the potentiality of tabu tunneling approaches to generate solutions of high quality on solving 0-1 multiknapsack problems.

6. References

- R. Aboudi and K. Jornsten, Tabu Search for General Zero Integer Programs using the Pivot and Complement Heuristic, *ORSA Journal on Computing*, 6 (1994) 82.
- E. Balas and C. H. Martin, Pivot and Complement - A Heuristic for 0-1 Programming, *Management Science Research* 26, 1 (1980) 86.
- R. Battiti and G. Tecchiolli, Local Search with Memory: Benchmarking RTS, in *OR-Spektrum*, 17 (1995) forthcoming.
- R. Battiti and G. Tecchiolli, The Reactive Tabu Search, *ORSA Journal on Computing*, 6 (1994) 126.
- I. Charon and O. Hudry, The Noising Method: A New Method for Combinatorial Optimization, *Operations Research Letters*, 14 (1993a) 133.
- I. Charon and O. Hudry, La méthode du bruitage: application au problème du voyageur de commerce, Working paper, ENST 93 D003, Paris, France (1993b).
- F. Dammeyer and S. Voss, Application of Tabu Search Strategies for Solving Multiconstraint Zero-One Knapsack Problems, Working paper, Technische Hochschule Darmstadt, Germany (1991).
- F. Dammeyer and S. Voss, Dynamic Tabu List Management using the Reverse elimination Method, *Annals of Operations Research*, 41 (1993) 31.
- A. Drexel, A Simulated Annealing Approach to the Multiconstraint Zero-One knapsack Problem, *Computing*, 40 (1988) 1.
- G. Dueck, New Optimization Heuristics - The Great Deluge Algorithm and the Record-to-Record Travel, Working paper, TR 89 06 011, IBM Heidelberg Scientific Center, Germany (1989).
- G. Dueck and T. Scheurer, Threshold Accepting: A General Purpose Optimization Algorithm, *Journal of Computational Physics*, 90 (1990) 161.
- G. Dueck and J. Wirsching, Threshold Accepting Algorithms for Multi-Constraint 0-1 Knapsack Problems, Working paper, TR 89 10 016, IBM Heidelberg Scientific Center, Germany (1989).
- A. Fréville, Contribution à l'Optimisation en Nombres Entiers, Habilitation à Diriger des Recherches, University of Paris XIII, France (1991).
- A. Fréville, A Comprehensive Survey of Heuristics for the 0-1 Multiknapsack Problem, Working paper, University of Valenciennes, France (1995).
- A. Fréville, S. Hanafi and A. El Abdellaoui, Noising Method for the 0-1 Multiknapsack Problem, Presented at the congress FRANCORO (Mons Belgium), University of Valenciennes, France (1995a).

- A. Fréville, S. Hanafi and A. El Abdellaoui, Efficiency of Tabu Tunneling on Solving 0-1 Multiknapsack Problems, Working paper, University of Valenciennes, France (1995b).
- A. Fréville and G. Plateau, Méthodes Heuristiques Performantes pour les Programmes en Variables 0-1, Working paper, ANO-91, University of Lille, France (1982).
- A. Fréville and G. Plateau, Heuristics and Reduction Methods for Multiple Constraints 0-1 Linear Programming Problems, *European Journal of Operational Research*, 24 (1986) 206.
- A. Fréville and G. Plateau, Hard 0-1 Multiknapsack Test Problems for Size Reduction Methods, *Investigacion Operativa*, 1 (1990) 251.
- A. Fréville and G. Plateau, FPMK90: an Algorithm for Solving the Multiknapsak Problem, Presented at Congress TIMS-SOBRAPO (Rio Brazil), Universities of Paris XIII and Valenciennes, France (1991).
- A. Fréville and G. Plateau, An Efficient Preprocessing Procedure for the Multidimensional 0-1 Knapsack problem, *Discrete Applied Mathematics*, 49 (1994) 189.
- A. Fréville and G. Plateau, The 0-1 Bidimensional Knapsack Problem: Towards an Efficient High Level Primitive Tool, Working paper, Universities of Paris XIII and Valenciennes, France (1995).
- M. Gendreau, A. Hertz and G. Laporte, A Tabu Search Heuristic for the Vehicle Routing Problem, *Management Science*, 40 (1994) 1276.
- F. Glover, Heuristics for Integer Programming Using Surrogate Constraints, *Decision Sciences*, 8 (1977) 156.
- F. Glover, Some Fresh Heuristic Principles for Integer Programming, Working paper, 77-7, University of Colorado, Boulder, USA (1978).
- F. Glover, Future Paths for Integer Programming and Links to Artificial Intelligence, *Computers and Operations Research*, 19 (1986) 533.
- F. Glover, Tabu Search: Part 1, *ORSA Journal on Computing*, 1 (1989) 190.
- F. Glover, Tabu Search: Part 2, *ORSA Journal on Computing*, 2 (1990) 4.
- F. Glover, Tabu Search Fundamentals and Uses, Working paper, University of Colorado, USA (1995).
- F. Glover and G. Kochenberger, Critical Event Tabu Search for Multidimensional Knapsack problems, Working paper, University of Colorado, USA (1995).
- F. Glover and M. Laguna, Tabu Search in: *Modern Heuristic Techniques for Combinatorial Problems*, ed. C.R.Reeves (Blackwell, Oxford, 1993).

- F. Glover and A. Lokketangen, Solving Zero-One Mixed Integer programming problems Using Tabu Search, Working paper, University of Colorado, Boulder, USA (1994a).
- F. Glover and A. Lokketangen, Probabilistic Tabu Search for Zero-One Mixed Integer Programming Problems, Working paper, University of Colorado, Boulder, USA (1994b).
- P. Hansen, The Steepest Ascent, Mildest Descent Heuristic for Combinatorial programming, Presented at the Congress on Numerical Methods in Combinatorial Optimization, Capri, Italy (1986).
- M.Laguna et al, Tabu Search for the Multilevel Generalized Assignment Problem, *European Journal of Operational Research*, 82 (1995) 176.
- J.S. Lee and M. Guignard, An Approximate Algorithm for Multidimensional Zero-One Knapsack Problems - a Parametric Approach, *Management Sciences* , 34 (1988) 402.
- S. Martello and P.Toth, *Knapsack Problems: Algorithms and Computer Implementations*, (Wiley & Sons, Chichester 1990).
- I.H.Osman, Heuristics for the Generalized Assignment Problem: Simulated Annealing and Tabu Search Approaches, in *OR-Spektrum*, 17 (1995) forthcoming.
- S. Senju and Y. Toyoda, An Approach to Linear Programming with 0-1 Variables, *Management Sciences* , 15B (1968) 196.
- J. Thiel and S. Voss, Some Experiences on Solving Multiconstraint Zero-One Knapsack Problems with Genetic Algorithms, *INFOR*, 32 (1994), 226.

Probabilistic Move Selection in Tabu Search for Zero-One Mixed Integer Programming Problems

Arne Løkketangen
Institute of Informatics
Molde College
Britveien 2, 6400 Molde, Norway
Email: Arne.Lokketangen@hiMolde.no

Fred Glover
School of Business, CB 419
University of Colorado at Boulder
Boulder, Colorado 80309, USA
Email: Fred.Glover@colorado.edu

Abstract:

We extend our previous work applying first-level Tabu Search mechanisms to 0/1 MIP problems, introducing probabilistic measures for move selection. These are especially applicable when the move evaluation function is contaminated by noise or contains elements not directly related to the objective function value. We also look at the possibility of replacing tabu memory completely by associated probabilistic guidance. Our approach is designed with the ability to solve general 0/1 MIP problems, and thus contains no problem domain specific knowledge. The outcome improves significantly on the solutions obtained by the first-level Tabu Search mechanisms previously employed. The probabilistic measures are extremely simple to implement, and can be readily incorporated in standard Tabu Search designs.

Keywords: Heuristics, Integer Programming, Probabilistic Tabu Search

1. Introduction

A variety of zero-one mixed integer programming (MIP) problems with special structures have been successfully treated by tabu search (TS). For an overview of TS and its application areas, see for example the application survey in Glover and Laguna (1993) and Glover et al. (1993). Tabu search has also been applied to solving general zero-one MEP problems by superimposing the TS framework on the "Pivot and Complement" heuristic by Balas and Martin (1980) (see Aboudi and Jörnsten (1994)), and by embedding TS within the "Pivot and Complement" heuristic (see, Løkketangen, Jörnsten and Storøy (1994)).

A more direct approach is used in Glover and Løkketangen (1995a), using a standard bounded variable simplex method as a subroutine, to exploit the fact that an optimal solution to the zero-one MIP problem may be found at an extreme point of the LP feasible set.

Building on the direct approach, this paper explores the use of probabilistic measures instead of the usual deterministic ones for move selection. We also explore the notion of abandoning almost completely tabu tenure in its deterministic sense (apart from a one-step move reversal), relying on probabilistic move selection mechanisms to guide our search. Our implementations are based on ideas first put forth in Glover (1989). An extension of this paper, which also treats probabilistic tabu tenure and a linkage to strategic oscillation is reported in Glover and Løkketangen (1995b).

The general outline of the paper is as follows. Section 2 gives the necessary background, while in Section 3 we describe probabilistic tabu search in general. In Section 4 we describe our approach to probabilistic move acceptance. Computational results are given in Section 5 followed by the conclusions in Section 6. The references are in Section 7.

2. Background and Fundamentals

The necessary background for the methods described later in the paper is outlined here, using the notation and definitions of earlier studies of the direct zero-one tabu search approach.

2.1 MIP Problem Formulation.

We represent the MIP problem formulation in the form

$$\begin{aligned}
 & \text{Maximize} && \sum (c_j x_j; j \in N) \\
 & \text{Subject to} && \\
 & && \sum (A_j x_j; j \in N) = b \\
 & && 1 \geq x_j \geq 0 \text{ and } x_j \text{ integer} \quad j \in I \subseteq N \\
 & && U_j \geq x_j \geq 0 \quad j \in C \subseteq N - I
 \end{aligned}$$

The maximizing form is chosen because it provides a natural representation for our multiconstraint knapsack test cases. The first-level TS heuristic for this problem, TS - MIP, may then be outlined as follows.

TS - MIP. Let x^* denote the best MIP feasible solution, and let z^* denote its objective function value.

Step 0: Solve the LP relaxation (i.e. ignoring integer constraints) to obtain an optimal LP basic (extreme point) solution.

Step 1: Consider the feasible pivot moves that lead to adjacent basic LP feasible solutions.

If a candidate move would lead to an MIP feasible solution x whose associated z value yields $z > z^*$, record x as the new x^* and update z^* .

Step 2: Select the pivot move with the highest move evaluation, applying tabu restrictions and aspiration criteria.

Step 3: Execute the selected pivot, updating the associated tabu search memory and guidance structures. Return to Step 1.

Note that the method may not necessarily visit a best MIP feasible neighbor of the current solution, since the move evaluation of Step 2 (which is not yet fully specified) depends on factors in addition to the objective function value.

2.2 Neighborhood Structure,

Let $x(0)$ denote a current extreme point solution, let $\{x_j; j \in NB\}$ denote the current set of non basic variables and let

$\{x_j; j \in B\}$ denote the set of basic variables ($B = N - NB$). The extreme points adjacent to $x(0)$ have the form

$$x(h) = x(0) - D_h \theta_h \quad \text{for } h \in NB$$

where D_h is a vector associated with the non basic variable x_h , and θ_h is the change in the value of x_h that moves the current solution from $x(0)$ to $x(h)$ along their connecting edge.

One useful view of the solution space is shown in Figure 1, where the axes denote the total amount of *integer infeasibility* (see the following discussion of Move Evaluation), and the associated *objective function value*. We thus start the search integer infeasible, and may also spend large parts of the search visiting integer infeasible solution states. The usefulness of moving in infeasible space is a common theme of tabu search (particularly its strategic oscillation component), and has been demonstrated in a variety of studies; e.g., Glover and McMillan (1986), Kelly, Golden and Assad (1993) and Gendreau, Hertz and Laporte (1994). An example of encountering difficulties when trying to stay feasible all the time is shown by Connolly (1992), where movements in the solution space are severely restricted for tightly constrained problem.

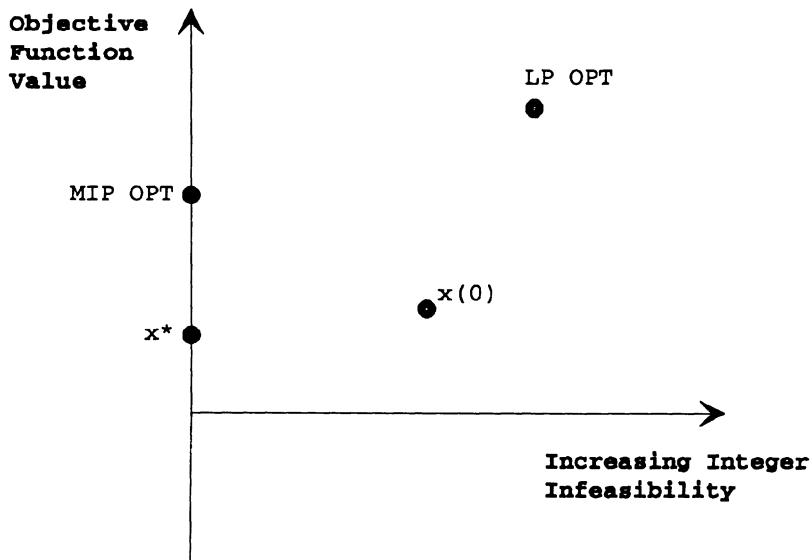


Fig 1. One view of the solution space.

2.3 Candidate List Strategy

Due to the moderate size of our test problems, we explore the full neighborhood, NB, evaluating all the neighboring extreme points, without making use of a more sophisticated design to extract a subset of the neighborhood for consideration. Candidate list strategies are described in Glover, Taillard and de Werra (1993) and Glover (1994).

2.4 Move Evaluation

Following the design that led to the successful outcome of our previous study, we employ a composite move evaluation function, based on two independent measures. The first measure is the change in *objective function value* when going from $x(0)$ to $x(h)$, and the second measure is the associated change in *integer infeasibility*. Let $\text{near}(x_j(h))$ denote the integer nearest to the value $x_j(h)$. Then we define $u(h)$ to be the amount of integer infeasibility for the solution $x(h)$, where

$$u(h) = \sum |x_j(h) - \text{near}(x_j(h))| \quad j \in I$$

Restricting consideration to $h \in \text{NB}$, we define

$$\begin{aligned} \Delta z(h) &= z(h) - z(0) \\ \Delta u(h) &= u(0) - u(h) \end{aligned}$$

Note that it is not necessary to execute a pivot to identify $x(h)$ or the values $u(h)$ and $z(h)$, since only the vector D_h , the scalar θ_h and the current solution $x(0)$ are required to make this determination.

2.5 Move Types

Likewise drawing on the foundations of our previous study, we classify our moves into four distinct move types, according to the values of $\Delta z(h)$ and $\Delta u(h)$:

- | | |
|----------------|--------------------------------------------------------------------------------------------------------------------------|
| Move Type I: | $\Delta z(h) < 0, \Delta u(h) > 0$ |
| Move Type II: | $\Delta z(h) > 0, \Delta u(h) < 0$ |
| Move Type III: | $\Delta z(h) \geq 0, \Delta u(h) \geq 0$ |
| Move Type IV: | $\Delta z(h) < 0, \Delta u(h) < 0$, or
$\Delta z(h) = 0, \Delta u(h) < 0$, or
$\Delta z(h) < 0, \Delta u(h) = 0$ |

Figure 2 shows how this move classification relates to the view of the solution space depicted in Figure 1.

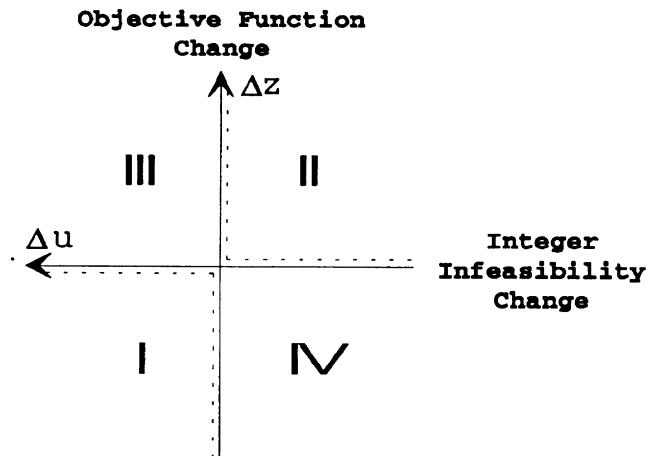


Fig 2. Classification of move types

2.6 Choice Rules

The classification of move types is made according to the degree to which they may be considered to be favorable. Moves of type III are generally the most favorable, improving in both measures, while moves of type IV are the least favorable, worsening in both measures. Moves of types I and II are improving in one of the measures, and worsening in the other.

The ranking of moves within each move group is as follows, with the best move in the group being indicated by the formula:

- | | |
|----------------|-----------------------------------------|
| Move Type I: | $\text{Max}(\Delta z(h)/\Delta u(h))$ |
| Move Type II: | $\text{Max}(\Delta u(h)/\Delta z(h))$ |
| Move Type III: | $\text{Max}(\Delta u(h) * \Delta z(h))$ |
| Move Type IV: | $\text{Min}(\Delta u(h) * \Delta z(h))$ |

When move evaluations tie within a group, we choose the move with the largest value of the positive component for moves of type I and II. For moves of type III and IV we choose the move which is most balanced in its components, i.e. where the Δu and Δz components are most equal.

Motivated by our earlier findings, the ranking of different move type groups is done by always considering type III moves first, and type IV moves last, with move types I and II grouped together in the middle. To be able to properly rank move types I and II, their values need to be normalized.

We first define the ratio R by

$$R = \frac{\sum |\Delta z(h)|}{w \sum |\Delta u(h)|}$$

where the summations are taken over the currently available moves of type I and II, and where w , the skew factor, can be used to change the emphasis between the two measures making up the move evaluation function. The normalized move values are

$$\text{Move Type I: } R1(\Delta h) = (\Delta z(h)/\Delta u(h))/R$$

$$\text{Move Type II: } R2(\Delta h) = (\Delta u(h)/\Delta z(h)) * R$$

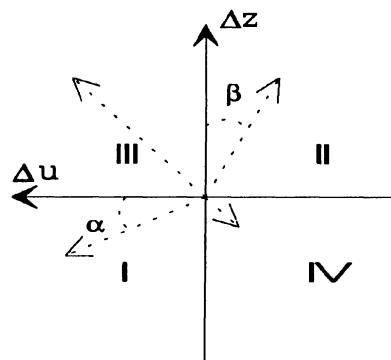


Fig 3. Ranking of move types

The preferred move selection from group I and II is then
 Move Type I and II: $\text{Max}(\text{R1}(h), \text{R2}(h))$

With reference to Figure 3, this means that we are comparing the angles α and β , choosing the smallest. Values of w other than 1 can be viewed as a skewing of the axis, causing these angles to shift, as indicated in Figure 4.

2.7 Aspiration Criteria

We have found it valuable for these 0-1 MIP problems to use two aspiration criteria; by *objective function levels*, and by detecting a *new current best* solution. In aspiration by objective function levels, we divide the range of possible LP feasible objective function values into L slots, and whenever a move would lead to an integer infeasibility value better (lower) than the best encountered so far for this slot, this new value is registered in the slot, and the move is selected. There is no need to aspire for objective function values smaller than that of the incumbent.

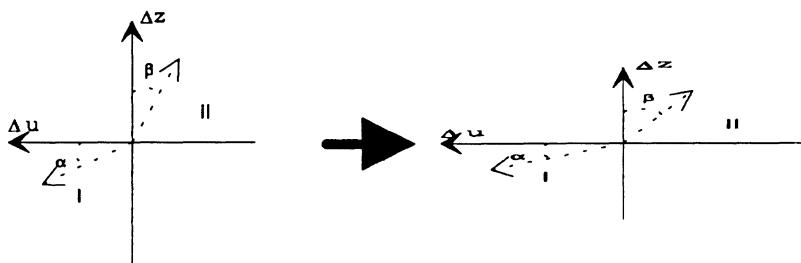


Fig 4. Skewing of axis.

As mentioned in Section 1, a new best integer feasible solution may be encountered during a move evaluation without subsequently selecting the move leading to it. This is due to the process for evaluating and ranking the moves. Such a solution is picked by the new best solution criterion, but with a threshold value, so that moves leading to poor integer feasible solutions are not selected.

2.8 Rejection of Degenerate Moves.

By degenerate moves, we mean type III moves where both $\Delta z(h)$ and $\Delta u(h)$ are zero. These moves often occur in bunches, and our experience is that they should be accepted only rarely. The rejection of (most) degenerate moves does not in general lead to better solutions, but the number of iterations required is reduced, and the tabu list is not clotted up with ineffectual moves.

3. Probabilistic Tabu Search

As sometimes noted in the TS literature, "controlled randomization" (which uses the biases of probability) may be viewed as a substitute for memory-when we are ignorant of how memory should advantageously be used. But just as there are multiple kinds of memory that can supplement each other, probabilities may find their best uses in various supplementary roles. The ideas that are tested in this paper originate in part from Glover (1989), and are adapted to our specific problem area.

For clarification, since more than one interpretation is possible, what is generally regarded as *Probabilistic TS* is usually applied to the move acceptance function, and is intended here to have the following design:

- A. Create move evaluations (for a current candidate list of moves examined) that include reference to tabu status and other relevant biases from TS strategies-using penalties and inducements to modify an ordinary objective function evaluation.
- B. Map these evaluations into positive weights, to obtain probabilities by dividing by the sum of weights. The highest evaluations receive weights that disproportionately favor their selection.

Among conjectures for why these probabilistic measures may work better than the deterministic analog, one may guess that move evaluations have a certain "noise level" that causes them to be imperfect-so that a "best evaluation" may not correspond to a "best move". Yet the imperfection is not complete, or else there would be no need to consider evaluations at all (except perhaps from a thoroughly local standpoint - noting the use of memory takes the evaluations beyond such a local context). The issue then is to find a way to assign probabilities that somehow compensates for the noise level.

The application of Probabilistic TS as outlined above can be guaranteed to yield an optimal solution under certain easily controlled conditions, if allowed to run for an infinite number of iterations (see Glover (1989)). This means that one may use Probabilistic TS as an escape hatch to hedge against persistent wrong moves in the absence of reliable knowledge to guide the search.

One may also view controlled randomization as a means for obtaining diversity without reliance on memory. In this respect it represents a gain in efficiency by avoiding the overhead otherwise incurred in the use of long term memory. However, it also implies a loss of efficiency as potentially unproductive wanderings and duplications may occur, that a more systematic approach would seek to eliminate.

There are also some similarities between Probabilistic TS and Simulated Annealing (SA), as both use probabilistic measures for move acceptance. One major difference is that SA samples the neighborhood (either randomly or systematically), and accepts any improving move encountered, and non-improving moves are accepted with a decreasing probability as the search progresses. In contrast, PTS collects the evaluated moves in a candidate list whose members receive probabilities of selection as a monotonic function of their attractiveness, strongly biased in favor of more attractive candidates, whether improving or not. Furthermore, the measure of attractiveness is generated to take account of penalties and inducements based on tabu restrictions and aspiration criteria. PTS is thus much more aggressive, and gives more guidance to the search process. For a description of SA that clarifies such differences, see Dowsland (1993), and see Connolly (1994) for a description of using SA as a general search tool component for pure ILP problems.

Hart and Shogan (1987) describe the use of probabilistic measures for move selection in greedy heuristics. Here the move is selected with uniform probability either among the n best moves, or among the moves better than some threshold (compared to the best move), but only considering improving moves. We tried the ideas of Hart and Shogan in our PTS framework (generalizing them to accept non-improving moves in order to go beyond the limitation of terminating at local optima). These tests were conducted both with and without the use of TS memory, but produced inferior results compared to the

methods reported in Section 5, which give a probabilistic bias toward selecting the presumably best moves.

4. Probabilistic Move Acceptance

To be able to apply the general probabilistic move acceptance approach as outlined in the previous section, we seek a move evaluation function that appropriately reflects the merit of the moves in the candidate list.

However, in the present setting we do not have a comparable measure for all the different move types identified in Section 1. Also, the move evaluation within each move type is quite noisy with respect to the overall goal of maximizing the objective function value, as it also contains a component designed to reduce the amount of integer infeasibility, and the two move evaluation components are combined rather dramatically by multiplication or division. It is therefore difficult to assign a good comparable numerical measure which reflects the *true merit* of all the moves. However, as shown in Glover and Løkketangen (1995a), there is good reason to believe that the *relative ranking* of the moves in the candidate list is quite good, as is the relative ranking of the move type groups, and that a move should be selected among the first few in the candidate list, if possible.

We therefore propose to introduce controlled randomization in the move selection process by exponentially decreasing the move acceptance probability when traversing the candidate list, thus relying solely on the individual move rankings.

4.1 Exponentially Decreasing Move Acceptance

The general outline of the method is as follows. Let p be the probability threshold for acceptance of a move, and let r be a randomly generated number (both in the range 0 - 1). Only the basic move selection core of the method is shown.

Step 1. Generate the candidate list in the usual way.

Step 2. Take the first (potentially best) move from the candidate list.

Step 3. Subject the move to the following sequence of tests:

- Accept if the aspiration criterion is satisfied.
- Reject if Tabu, and place the move at the end of the candidate list, removing its tabu status for the current iteration.

- Apply degenerate move rejection.
- Generate r . If $r > p$, reject. Go to Step4.
If $r \leq p$, accept move, exit.

Step 4. Select the next move on the candidate list. Go to Step 3.

If all acceptable moves are rejected, select the last (or first) nontabu move.

The method is intriguing because of its simplicity, and can easily be implemented in most deterministic TS frameworks. The value of p should not be too small, as we would usually like to usually select one of the top few moves. Testing will show good values for p , but as an example, consider $p = 1/3$. The probability of selecting each of the first d moves is then (disregarding aspiration criteria, tabu status, etc.):

$$1/3, 2/9, 4/27, 8/81, \dots, 2^{d-1}/3^d.$$

The probability of not choosing one of the first d moves is $2^d/3^d$. So $p = 1/3$ gives a very high probability of picking one of the top moves: about .87 for picking one of the top 5, and about .98 for picking one of the top 10.

The effective value of p can also be viewed as a function of the quality of the move evaluation function. The better the move evaluation function, the higher the expected value of p for which the best solutions are obtained.

5. Computational Results

To test our heuristics, we employ the same set of test problems as used in Glover and Løkketangen (1995a), but restrict our reporting to the cases where the first-level tabu search mechanisms of this previous study (described in Section 1) failed to find the optimum. (We also verified that the other problem cases were easy for our new approach as well.) The test problems are of the multiconstraint knapsack type, collected from the literature by Drexel (1988), and employed extensively in the literature as a standard test case portfolio.

The heuristics in this paper were implemented in FORTRAN, incorporating subroutines provided by ZOOM/XMP whenever

possible. The ZOOM/XMP software package was also used to solve the initial LP relaxation. See Marsten (1989a, 1989b).

5.1 First-level Tabu Search Parameters

The following test parameters were used, except when explicitly stated otherwise. All tests were done with a dynamic tabu list length size starting at 5 or $\lceil \text{SQRT}(N) \rceil$, whichever is larger, where N is the number of integer variables. The list size is then varied randomly between this number and its double. Aspiration is by Objective Function Levels, and new best solution (see Section 1), with a threshold at 0.9 LP*. Degenerate moves are accepted with a probability of 0.1. The axis skew factor, w , was set to 1. The value of w was halved and the method was restarted if a long run of at least $2*N$ integer infeasible iterations occurred. Up to 10^*N iterations were allowed per problem.

Table I. Probabilistic TS for PB4

<i>p</i>	10*N	20*N
1.0	90909	91935
0.9	90909	91935
0.8	90615	90909
0.7	91935	91935
0.6	90909	90909
0.5	89659	90909
0.4	90858	93118
0.35	92506	92506
0.3	90008	94965
0.25	90909	90909
0.2	87937	88724
0.1	90615	90615

5.2 Probabilistic Move Acceptance

The first objective was to find a good value or range of values for p , the probability threshold for accepting a move, as outlined in Section 3.1. The results for varying p over the range from 0 to 1 in 0.1 intervals are shown in Table 1. The chosen test case is PB4, for which the first-level TS fails to find the optimum. (The first-level test is represented in the table by $p = 1$). We show the best achieved values for 10^*N iterations and 20^*N iterations, since the probabilistic TS should need more time than the first-level TS, being less focused. The

values for p of 0.25 and 0.35 were added to give extra resolution around the best area.

As can be seen from the table, good values for p for this problem lie in the range 0.3 to 0.4. Larger values for p lead to inferior solutions when compared to the deterministic variant. This is probably because too many potentially good moves are thrown away, while at the same time the choices do not create enough diversification. At the other end of the scale, with values of p from 0.2 and down, too few good moves are chosen to focus the search properly. Figure 5 compares the search progress for the best ($p = 0.3$) probabilistic case for PB4 vs. the deterministic case. The plot shows the best feasible solutions visited in each implementation.

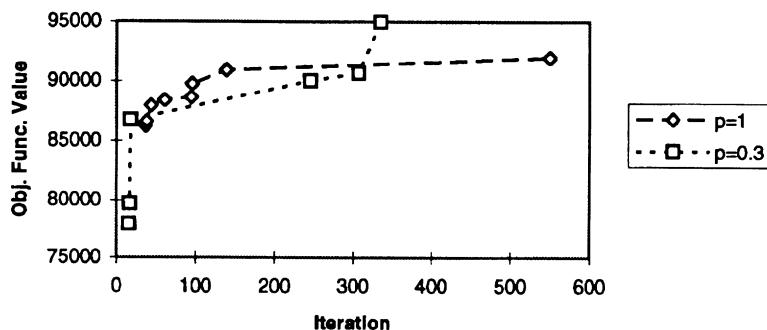


Fig 5. Search progress for PB4 with $p=0.3$ and $p=1$

Similar tests for other problems showed the same pattern, with good values for p being between 0.3 and 0.5, so an overall value of $p = 0.4$ was chosen for further testing.

One important finding from our preliminary testing was that low values of p make it more difficult to obtain integer feasibility. This is due to the fact that good moves are too often rejected that contribute

to the reduction of integer infeasibility. Searches that experience this problem could either use a larger value for p , or a lower value for w .

It is also more important in a probabilistic TS approach based on ranking to have good aspiration criteria, since good solutions in the neighborhood are even less likely to be visited than in the deterministic case, and should be actively pulled out.

5.3 Probabilistic Move Acceptance with no Tabu Memory

We also investigated the conjecture that the probabilistic move acceptance scheme should work without any tabu memory, solely relying on biased randomization as guidance. Table II shows the results for PB4 for various values of p with a tabu list length of 1 (chosen to avoid direct move reversals). As can be seen, for values of p in the range 0.4 to 0.6, this scheme obtains the same results as the first-level tabu search approach, though inferior to the probabilistic tabu search approaches with memory. It is also evident that the best

relative results are obtained for few iterations, as the extra search effort expended when going from 10^*N to 20^*N iterations only yields marginally better outcomes.

Table II. No tabu tenure for PB4

p	10^*N	20^*N
0.9	84730	84730
0.8	84730	84730
0.7	89659	89659
0.6	90909	90909
0.5	90909	90909
0.4	90909	90909
0.3	90615	90615
0.2	90909	90909
0.1	88030	90615

This method is extremely simple and efficient to implement, and can be used in settings where the goal is to obtain good solution in a small number of iterations, or where the tabu attributes might be difficult to identify or separate.

We note that the tabu tenure of 1 for the "almost memoryless" case may be a stronger restriction in the present setting than in many others, since preventing the reversal of a single pivot step may cause the only paths for reaching the preceding solution to consist of multiple pivots, depending on the polyhedral structure of the problem. Consequently, a larger tenure may be appropriate to obtain similar results with probabilistic tabu search in other applications.

5.4 General Test Results

We ran two series of tests for each test case, one running for 10^*N iterations, to compare the results with those obtained in Glover and Løkketangen (1995a), and the other for 20^*N iterations to check the assumption that probabilistic measures need more search time, as the search tends to be less focused. The results are reported in Table III.

Table III. General test results

Problem	M*N	LP	IP	First-level TS	PMA - 10*N	PMA0 - 10*N	PMA - 20*N	PMA0 - 20*N
PET5	10*28	12462.1	12400	12370	12380	12390	12400	12390
PET7	5*50	16612	16537	16507	16468	16494	16508	16524
PB4**	2*29	99622.7	95168	90909	93118	90909	93118	90909
PB6**	30*40	843.3	776	723	765	729	765	729
PB7**	30*37	1086.2	1035	1033	1035	1011	1035	1011
WEISH07*	5*40	5601.9	5567	5525	5542	5541	5567	5541
WEISH08	5*40	5631.6	5605	5603	5605	5603	5605	5603
WEISH16	5*60	7314.02	7289	7287	7287	7288	7287	7288
WEISH18	5*70	9603.7	9580	9565	9565	9565	9580	9565
WEISH19	5*70	7756.9	7698	7674	7698	7674	7698	7674
WEISH22	5*80	9004.2	8947	8851	8947	8908	8947	8908
WEISH25	5*80	9964.7	9939	9923	9923	9923	9928	9923
WEISH26	5*90	9641.6	9584	9532	9584	9584	9584	9584
WEISH27	5*90	9849.7	9819	9811	9819	9811	9819	9811
WEISH29	5*90	9429.03	9410	9410	9354	9378	9354	9410
WEISH30	5*90	11194.5	11191	11160	11191	11191	11191	11191
SENTO1*	30*60	7839.3	7772	7772	7719	7728	7719	7772
SENTO2	30*60	8773.2	8722	8711	8701	8722	8701	8722

The tests for probabilistic move acceptance use $p = 0.4$, and are reported under the columns PMA - 10*N and PMA - 20*N. The columns PMAO - 10*N and PMA - 20*N report the probabilistic move acceptance without tabu memory, also with $p = 0.4$. The symbol '*' indicates that $w = 0.5$, while the symbol '**' indicates $w = 0.25$. Optimal values are indicated in bold. (The slight discrepancy between the results obtained here and those reported in Glover and Løkketangen (1995a) are due to a change in the random number generator used.)

As can be seen from the tables, the use of probabilistic measures for the move selection function improves the general solution quality.

The tables also disclose the outcome (to be expected) that solution quality is clearly correlated with the number of iterations. The tables do not reflect the observation, however, that the probabilistic move acceptance methods find good solutions very quickly compared to the first-level TS approach. This is especially true when no tabu memory is employed, but the method without memory has difficulty in improving on those early good solutions.

Since the "no memory" method actually employs a tabu tenure of 1, these outcomes suggest the relevance of progressively varying the tenure for probabilistic tabu search, starting with a very small tenure to find high quality solutions rapidly, and then gradually enlarging the tenure. A periodic return to very small tenures, particularly at the conclusion of diversification steps, likewise would appear useful based on our findings.

We also checked our heuristics against some of the presumably easier cases solved by the first-level TS approach, and found results similar to those reported above.

6. Conclusions

We have shown how simple probabilistic measures can be used to improve a first-level TS significantly, enhancing diversifying aspects of the search. The use of probabilistic move acceptance is especially important if the move evaluation function is contaminated by noise or contains elements not directly related to the objective function value. We have also shown how a simple ranking scheme can give superior results when the move evaluation function is difficult to compare numerically for different parts of the search neighborhood. Our

approach, which does not depend on any "deep formula," is appealing in its simplicity, and is easily incorporated into other Tabu Search designs.

It is also noteworthy that we obtained relatively good results from the "degenerate" version of the probabilistic move acceptance method, where we abandoned tabu memory except for the single iteration memory that prohibits an immediate move reversal. This approach appears useful when it is imperative to find good solutions quickly, or when tabu attributes might be difficult to identify or separate. These outcomes further suggest the value of combining the probabilistic design with a variable tabu tenure that periodically receives much smaller values than customarily envisioned to be relevant. Our findings indicate that an additional stipulation deserves to be added to the hypothesis that probability, appropriately used, may partly substitute for the role of memory. Apparently, the right use of probability can also enhance the effectiveness of memory.

7. References

- R. Aboudi and K. Jörnsten, Tabu Search for general Zero-One Integer Programs using the Pivot and Complement Heuristic. *ORSA Journal on Computing*, Vol 6, No. 1 (1994), pp 82-93.
- E. Balas and C. Martin, Pivot and Complement - Heuristic for 0-1 Programming. *Management Science* 26, No. 1 (1980), pp 86-96.
- D. Connolly, General Purpose Simulated Annealing. *Journal of the Operational Research Society*, Vol. 43, No. 5, (1992), pp 495-505.
- K. A. Dowsland, Simulated Annealing, in: *Modern Heuristics for Combinatorial Problems*, ed. C. R. Reeves, (Blackwell, Oxford, 1993).
- A. Drexel, A Simulated Annealing Approach to the Multi-constraint Zero-One Knapsack Problem. *Computing* 40 (1988), pp 1-8.

- M. Gendreau, A. Hertz and G. Laporte, A Tabu Search Heuristic for the Vehicle Routing Problem. *Management Science*, Vol 40, No. 10, (1994), pp 1276-1290.
- F. Glover, Tabu Search - Part I. *ORSA Journal of Computing*. Vol 1, No. 3 (1989), pp 190-206.
- F. Glover, Tabu Thresholding: Improving Search by Nonmonotonic Trajectories. Working paper, University of Colorado at Boulder (1994). (To appear in *ORSA Journal on Computing*).
- F. Glover and M. Laguna, Tabu Search, in: *Modern Heuristics for Combinatorial Problems*, ed. C. R. Reeves, (Blackwell, Oxford, 1993).
- F. Glover, M. Laguna, E. Taillard and D. de Werra. *Annals of Operations Research*, Vol 4 1. (J. C. Baltzer AG, 1993).
- F. Glover and A. Løkketangen. Solving Zero-One Mixed Integer Programming Problems using Tabu Search. Working Paper, Institute of Informatics, Molde College, Norway, (1995a).
- F. Glover and A. Løkketangen. Probabilistic Tabu Search for Zero-One Mixed Integer Programming Problems. Working Paper, Institute of Informatics, Molde College, Norway, (1995b).
- F. Glover, E. Taillard and D. de Werra. A User's Guide to Tabu Search in: *Annals of Operations Research*, Vol 41, (1993), pp 3-28.
- F. Glover and C. McMillan. The General Employee Scheduling Problem: An Integration of Management Science and Artificial Intelligence. *Computers and Operations Research*, Vol. 15, No. 5 (1986), 563-593.
- J. P. Hart and A. W. Shogan, Semi-Greedy Heuristics: An Empirical Study. *Operations Research Letters*, Vol. 6, No. 3, (1987), pp 107-114

- J.P. Kelly, B. Golden and A.A. Assad, Large-Scale Controlled Rounding Using Tabu Search with Strategic Oscillation. *Annals of Operations Research*. Vol. 41 (1993), pp 69-84.
- A. Løkketangen, K. Jörnsten and S. Storøy. Tabu Search Within a Pivot and Complement Framework. *International Transactions of Operations Research*. Vol. 1, No. 3, (1994), pp 305-316.
- R. E. Marsten. XMP Technical Reference Manual, XMP Software. (1989a).
- R. E. Marsten. User's Manual for ZOOM/XMP, XMP Software. (1989b).

A Star-Shaped Diversification Approach in Tabu Search

Lutz Sondergeld and Stefan Voß

*Technische Universität Braunschweig, Institut für
Wirtschaftswissenschaften, Abt-Jerusalem-Straße 7,
D - 38106 Braunschweig, Germany
E-mail: Stefan.Voss@tu-bs.de*

Abstract:

Diversification is among the most versatile strategic components of tabu search. In this paper we propose and investigate a star-shaped approach providing multiple disjoint search trajectories scattering over the solution space. The approach is based on a dynamic strategy that explicitly incorporates logical interdependencies between multiple lists representing these trajectories. Numerical results are presented for some benchmark instances derived from the problem of balancing hydraulic turbine runners, which may be formulated as a quadratic assignment problem.

Key Words: Diversification, quadratic assignment problem, tabu search, turbine runner balancing.

1. Introduction

Tabu search as an exponent of recent metastrategies for overcoming local optimality has become one of the success stories within solving hard combinatorial optimization problems. Whereas its basic prescriptions have already led to reasonably good solutions simple ideas of intensification and diversification proved to be successful further-on (see, e.g., Skorin-Kapov (1990) for an early reference). Recently, research has been conducted focusing on broader aspects of intensification and diversification (cf. Glover and Laguna (1993), Kelly et al. (1994), Soriano and Gendreau (1993), Voß (1995)). Usually being utilized in longer term strategies diversification aims at creating solutions within regions of the search space not previously inve-

stigated. On the other hand intensification tries to aggressively lead the search into promising regions by hopefully incorporating good attributes or elements.

In this paper we propose a specific diversification approach that strategically seeks a diversified collection of solutions by scattering multiple search trajectories over the solution space. Based on the reverse elimination method, a tabu search approach observing logical interdependencies between attributes or moves encountered throughout the search (see Section 2), these ideas are specified in more detail in Section 3. The problem for exemplification is the turbine runner balancing problem (**TBP**, see Section 4) which is a special case of the quadratic assignment problem. Section 5 gives numerical results for our approaches. Some benchmark problems from the literature are solved or closely solved to optimality whereas some of our solution values have not previously been known in the literature. In the final section some conclusions are provided.

2. Tabu Search - Reverse Elimination Method

Tabu search is a metastrategy to overcome local optimality. Basically, this strategy aims at solving optimization problems as, e.g.,

$$\text{Minimize} \quad f(x) \quad \text{subject to} \quad x \in S \quad (1)$$

where S is a discrete set and f a real-valued function defined on S . Starting from an initial solution a local search approach iteratively transforms a current solution into a neighbourhood solution (i.e. performs so-called moves by means of one or more attributes). To prevent from cyclic repetitions of solutions previously encountered during the search at each iteration a so-called tabu list of prohibited attributes or moves is maintained. We assume that the reader is well acquainted with the basic concepts of tabu search as they can be found in respective surveys especially of Glover (see, e.g., Glover and Laguna (1993)). For ease, however, we introduce the notion of a running list which is used to store all attributes (e.g. the indices of those variables of a mathematical programming formulation changing their values) of all moves performed during the search.

More advanced concepts of tabu search involve dynamic concepts for the tabu list management as, e.g., the reverse elimination method (**REM**). Based on observing logical interdependencies between the attributes of the moves performed throughout the search, the REM provides a necessary and sufficient condition to prevent

from re-visiting previous solutions (cf. Glover (1990), Dammeyer and Voß (1993), Voß (1995)). The basic proceeding of the REM and its underlying motivation may be explained as follows.

In the tabu list management of the REM in any iteration the tabu list is built anew. To prevent the search from re-visiting solutions that had already explicitly been explored the basic idea of REM is to trace back the actual running list (from the last attribute up to the first) to determine those attributes whose reversion within the next iteration/move would lead the search back to a previously encountered solution. While backtracing, a so-called residual cancellation sequence (RCS) is built up stepwise by processing all attributes. This sequence reveals those attributes that have to be reversed to lead the search trajectory from the current solution back to one previously explored. Initializing the sequence as being empty it is successively modified by either adding or dropping attributes. An attribute is added, if its complement is not within the sequence. Otherwise this complementary attribute is dropped, i.e. cancelled. If at any state of the backtracing all attributes of the sequence represent a feasible move then its complementary move obtains a tabu tenure (usually for exactly one iteration).

Determining all tabu attributes according to the REM at iteration $iter$ may be visualized as follows (for simplification we assume that each move consists of exactly one attribute as well as $t := 1$):

```

for all attributes  $k$  do  $status(k) = .free$ .  

                                (* designate all attributes 'not tabu' *)  

RCS :=  $\emptyset$                                 (* initialize RCS *)  

for  $i = iter, 1, \text{step } -1$  do      (* backtracing of the running list  $RL$  *)  

    if  $\overline{RL(i)} \in RCS$  then  

         $RCS := RCS - \{\overline{RL(i)}\}$  (*  $RL(i)$  cancels its complement on RCS *)  

    else  

         $RCS := RCS \cup \{RL(i)\}$           (*  $RL(i)$  is added to RCS *)  

    endif  

    if  $|RCS| \leq t$  then  $status(\bar{k}) := .tabu$ . for all  $k \in RCS$   

                                (* designate tabu tenure *)

```

3. Diversification

In general search diversification may be explained as a long term memory to penalize often selected assignments where the neighbourhood search can be guided into not yet explored regions to restart the tabu list operation. A simple way to diversify the search with the

REM is to relax the necessity of the method in that each attribute which is supposed to stay tabu for the next iteration gets a tabu tenure of more than one iteration. For instance, we may assume a tabu tenure of ten iterations instead of one, if a diversification approach of this kind is performed (abbreviated by **REM/10** opposite to **REM/1** for the original version of the REM).

The REM provides another interesting modification for search diversification. For simplicity assume that a move consists of exactly one attribute. If at any tracing step the RCS includes exactly t moves (with t being a positive integer, cf. Section 2) then it is possible to define a tabu tenure for the complements of all these moves for the next iteration. The case $t = 2$ seems to be appealing because a larger number of solutions is encountered than with REM in the same number of iterations. That is, all common neighbours of the current solution and of an already explored one are forbidden. These neighbours were implicitly investigated during a former step of the procedure (due to the choice of a best non-tabu neighbour) and need not be looked at again (see, e.g., Voß (1993)).

In the sequel we provide some further ideas for different means of diversification that utilize the special structure of the REM.

3.1. Star-Shaped Diversification Approaches - SSA1

The first star-shaped diversification approach (**SSA1** for short) uses parallel running lists (and tabu lists) instead of single lists in conventional tabu search concepts. 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 any iteration a current solution referring to a specific running list maintains a given minimum distance to current solutions of 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 (cf. Figure 1). That is, diversification of trajectories using simple distance measures should produce collections of solutions, which are disjoint with high probability.

3.2. Star-Shaped Diversification Approaches - SSA2

With respect to our idea of a star-shaped diversification approach more elaborate implementations may be investigated, too. In this

section we consider a different multiple list approach where the disjointness of solutions can be assured in any case.

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 may successively be connected to the current running list in reverse order. That is, pseudo-trajectories are constructed representing fictitious search paths from the final (i.e. 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 or not. 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 (cf. Figure 2).

Note that this so-called exact star-shaped approach (**SSA2**) only works with tabu search methods operating on dynamically controlled running lists, whilst diversification using distance measures even works with conventional static methods. This is one motivation for us to concentrate on the reverse elimination method in this paper.

4. The Turbine Runner Balancing Problem (TBP)

A hydraulic turbine runner consists of a cylinder around which a number n of blades are welded at regular spacings. These runners, which are used, e.g., in electricity generation, should be manufactured such that the center of mass of the blades and the geometric center of the cylinder coincide (cf. Mosevich (1986)). If this is not possible, due to weight deviations of the blades the problem is to determine the location of the blades around the cylinder so as to minimize the distance between the geometric center and the center of mass. With weights w_i of the blades and $loc(i)$ corresponding with the location of blade i ($i = 1, \dots, n$) this deviation between geometric center and the center of mass is defined as follows, where r is the radius of the cylinder:

$$d^2 = (r / \sum_{i=1}^n w_i)^2 \sum_{i=1}^n \sum_{j=1}^n w_i w_j \cos(2\pi(loc(i) - loc(j))/n) \quad (2)$$

Furthermore, all blades have to be bijectively assigned to locations (see loc 1 ... loc n in Figure 3). With binary variables x_{ip} which take

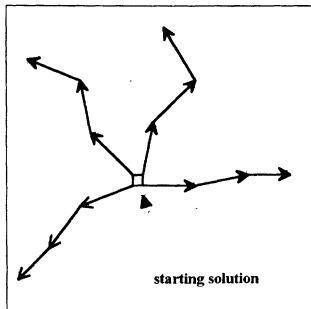


Fig. 1 : Diversified search trajectories

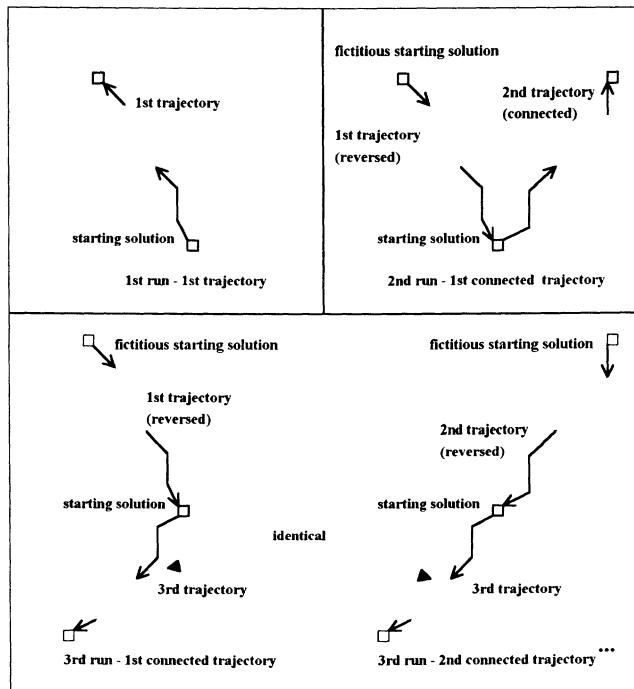


Figure 2 : Diversified search trajectories

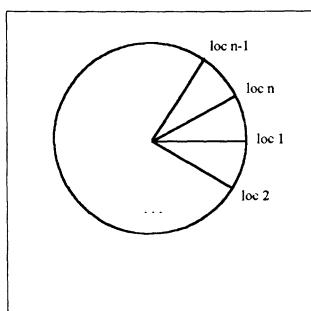


Fig. 3 : Turbine balancing problem

the value 1 if blade p is assigned to a specific location i and take the value 0 otherwise, the following restrictions apply:

$$\sum_{i=1}^n x_{ip} = 1 \quad p = 1, \dots, n \quad (3)$$

$$\sum_{p=1}^n x_{ip} = 1 \quad i = 1, \dots, n \quad (4)$$

From this it becomes obvious that the TBP provides special instances of the quadratic assignment problem (QAP). The QAP is well-known to be NP-hard and hence there is an obvious need for good heuristic approaches including improvement procedures. Starting from a given feasible solution the basic principle of a local search approach is to successively interchange single assignments as long as improvements in the objective function are obtained.

Considering the above formulation a transition, i.e. a move, from one feasible solution to another one may occur by swapping or exchanging two objects such that each is placed on the location formerly occupied by the other. The main drawback of local search algorithms is their inability to continue the search upon becoming trapped in a local optimum. This suggests consideration, for instance, of recent tabu search techniques for guiding known heuristics to overcome local optimality, see, e.g., Battiti and Tecchiolli (1994), Chakrapani and Skorin-Kapov (1993), Taillard (1991), Voß (1995). For additional references on the QAP see, e.g., Pardalos and Wolkowicz (1994) and the contributions therein.

The TBP itself has been heuristically solved by several authors (Fathi and Ginjupalli (1993), Laporte and Mercure (1988), Sinclair (1993)). Before giving detailed results some of these algorithms from the literature will be briefly sketched.

The first is a well-known greedy heuristic for the QAP (**GH**, cf. Fathi and Ginjupalli (1993)). Without loss of generality, assume that the weights w_i ($i = 1, \dots, n$) are ordered according to their respective weights and that the heaviest blade is placed into location 1. In each of the remaining $n - 1$ iterations, say iteration k , the k -th blade is placed into that not yet occupied location l^* such that $\sum_{i=1}^{k-1} w_i \cos(2\pi(l^* - loc(i))/n)$ is minimum among all still available locations.

Contrary to this basic greedy heuristic where at each iteration k the k -th blade is permanently assigned to its final location, two extensions **GHL1** and **GHL2** involve a look ahead mechanism by

just temporarily placing the k -th blade and further exploring the placement of the remaining $n - k$ blades.

In iteration k of GHL1 the k -th blade is successively placed in each of the not yet occupied locations, whereas procedure GH determines the placement of the remaining blades. The assignment of the k -th blade to a location yielding a configuration with a minimum deviation value for all temporary placements is now taken as permanent. GHL2 as a further and more complex look ahead extension of the basic greedy heuristic determines the placement of the remaining blades in iteration k by using GHL1.

The rotational heuristics (**RH**) are based on partitioning the blades into subsets of equal cardinality. That is, a rotational heuristic with k sets may be referred to as RH k . For each of these subsets a heuristic subconfiguration is determined by procedure GHL2. The blades from the first subconfiguration are placed on positions $1, 1 + k, 1 + 2k, \dots$. The remaining configurations are located in positions inbetween those already occupied. Thereby the procedure successively changes the direction of insertion: first clockwise, then counterclockwise and so on. Finally, a number of distinct configurations is obtained by fixing one subconfiguration while rotating the remaining $k - 1$ configurations. That configuration with the lowest deviation gives the overall solution. (Note that RH k is only applied when n is divisible by k . Below we restrict ourselves to the best from $k = 2, 3$, and 4.)

5. Numerical Results

In this section numerical results for the algorithms discussed above will be presented. All computations have been performed on a Pentium PC with 90 MHz.

Table 1 gives the results for various TBP instances with $5 \leq n \leq 80$ blades where the mass values are fixed as follows: $w_i := i$. This way of problem definition, i.e., these instances have been treated as benchmark problems by several researchers (see, e.g., Fathi and Ginjupalli (1993) and the references therein).

Note that for the numerical results the objective function values are given as d instead of d^2 . The first column of Table 1 gives the problem size and all other columns refer to specific algorithms. The greedy based approaches are re-implementations, whereas our results for GHL2 seem to outperform the previous results given in Fathi and Ginjupalli (1993). Column RH gives results for the ro-

	n	GH	GHL1	GHL2	RH	REM/1	REM/10	SSA1/I	SSA1/I/10	SSA2/I	SSA2/I/10	STS
	5	29.93520	29.93520	29.93520	-	29.93520	29.93520	29.93520	29.93520	29.93520	29.93520	29.93520
	6	0.00000	0.00000	0.00000	0.00	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	7	30.41534	9.54478	2.73179	-	2.73179	2.73179	2.73179	2.73179	2.73179	2.73179	2.73179
	8	30.06645	3.64767	30.06	3.64767	3.64767	3.64767	3.64767	3.64767	3.64767	3.64767	3.64767
	9	13.94019	11.02367	3.80432	20.48	0.63675	0.63675	0.63675	0.63675	0.63675	0.63675	0.63675
	10	22.47396	2.05269	0.00000	0.00	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	11	2.09723	2.09723	1.71645	-	0.0320	0.15422	0.00320	0.01979	0.00352	0.00352	0.38198
	12	0.00000	0.00000	0.00	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000	0.00000
	13	8.0605	2.74062	0.37669	-	0.02662	0.11233	0.01265	0.03961	0.02662	0.08478	0.07347
	14	0.00000	0.00000	0.00	0.00000	0.00000	0.05121	0.04107	0.05121	0.02842	0.00000	0.00000
	15	15.68122	0.84705	0.08474	0.00	0.07179	0.06047	0.05346	0.11678	0.03447	0.04337	0.00760
	16	7.49699	0.54434	0.06279	0.37	0.03249	0.03696	0.03696	0.06618	0.035249	0.04241	0.08977
	17	5.03234	0.44695	0.04534	-	0.04821	0.06750	0.03335	0.02346	0.021190	0.021192	0.03191
	18	4.06195	0.18669	0.00000	0.00	0.02409	0.02955	0.03798	0.08379	0.020409	0.03097	0.04464
	19	0.73397	0.71934	0.10893	-	0.00333	0.03529	0.01156	0.02682	0.003333	0.00170	0.08835
	20	1.84156	0.30135	0.00000	0.00	0.02307	0.02307	0.02791	0.01618	0.04285	0.02307	0.07649
	21	3.05081	0.28061	0.03526	0.00	0.00366	0.00720	0.01803	0.01604	0.00366	0.00518	0.0169
	22	2.94692	0.37932	0.02769	0.00	0.01768	0.01520	0.00980	0.00630	0.00442	0.01478	0.03931
	23	1.63104	0.13240	0.02200	-	0.01335	0.02075	0.01590	0.02638	0.00723	0.00331	0.03984
	24	2.46123	0.15537	0.01659	0.00	0.00340	0.00045	0.00920	0.00768	0.01546	0.00886	0.01399
	25	1.52628	0.14276	-	0.00119	0.01272	0.00920	0.00844	0.01848	0.00658	0.04438	0.04438
	26	1.13434	0.11564	-	0.00749	0.00325	0.00749	0.00325	0.00774	0.00548	0.00806	0.01809
	27	1.93152	0.11564	-	0.01285	0.00260	0.00279	0.00755	0.00367	0.00631	0.06531	0.06531
	28	0.89992	0.12765	-	0.00918	0.00438	0.00340	0.00879	0.01672	0.00383	0.03071	0.03071
	29	1.23343	0.06200	-	0.00765	0.00199	0.00685	0.00199	0.00560	0.00199	0.02539	0.02539
	30	2.23265	0.04384	-	0.00889	0.00496	0.00855	0.00298	0.01224	0.00347	0.02905	0.02905
	40	0.07992	-	-	0.00161	0.00084	0.00301	0.00662	0.00330	0.00257	0.00030	0.00030
	50	0.15375	0.03112	-	0.00862	0.00113	0.00285	0.00047	0.00673	0.00088	0.00466	0.00466
	60	0.24642	0.01014	-	0.00191	0.00120	0.00081	0.00055	0.00177	0.00115	0.00526	0.00526
	70	0.36894	-	-	0.00827	0.00707	0.00339	0.00037	0.00352	0.00352	0.00177	0.00177
	80	0.21927	-	-	0.00269	0.00262	0.00015	0.00379	0.00211	0.00144	0.00144	0.00144
	sum(dv(0))				37.28685	37.55715	37.72012	37.47627	37.30784	37.30163	38.03781	
	sum(best)				13	11	11	14	11	11	10	

Table 1 : Numerical results for TBP data

tational heuristic as they are taken from the same source (with a restricted precision of up to two decimals). The remaining columns refer to different tabu search methods. Table 2 gives overall computation times for some selected problems.

n	GH	GHL1	GHL2	REM/1 (REM/10)	SSA1/1 (SSA1/10)	SSA2/1 (SSA2/10)	STS
10	0	0	6	49	50	705	20
20	0	9	648	110	115	741	86
24	0	22	2341	151	153	825	145
30	0	70	-	221	224	865	203
40	0	325	-	360	364	1066	350
50	1	827	-	586	593	1266	570
60	2	1982	-	823	833	1517	800
70	3	-	-	1140	1146	1828	1101
80	4	-	-	1501	1507	2176	1485

Table 2: CPU times for some TBP data (in seconds)

In order to apply REM and the star-shaped diversification approach for the TBP the following adaptation seems to be suitable. First of all a move refers to the transition from any solution to a neighbourhood solution by swapping or exchanging two objects (i.e. blades in the TBP). Note that a move is described by more than one attribute in this case. For SSA1 a simple distance measure between two solutions of the same iteration can be defined by the number of objects assigned to different locations. With a given lower bound for this number the search trajectories referring to the running lists are led into different regions of the solution space. Preliminary numerical tests proved a value of about 60% for this lower bound to be convenient.

For comparison we re-implemented a static tabu search approach known for the QAP (see column **STS**, cf. Taillard (1991, 1994)). At iteration k of this approach a move is tabu if both blades would be assigned to locations they occupied within the last u iterations. Furthermore, if an object has not been placed at a location during the last v iterations, all moves not performing this particular assignment are also tabu. Whereas u as a short term memory parameter prohibits the search trajectory from revisiting formerly encountered solutions, v forces specific assignments and therefore has to be seen as a long term memory diversification parameter. Although the optimal choice for both parameters depends on the problem character and therefore could not be provided in general, in Taillard (1994) a

random modification of u around n of about 10% is said to be fairly robust, while for v a value between $2n^2$ and $5n^2$ seems to be convenient for most problems. In our implementation we choose fixed values $u = n$ and $v = 3n^2$.

Note that the starting solutions for all tabu search approaches (i.e. all variants of the REM as well as STS) are the permutation $(1, \dots, n)$, i.e. assignments with $loc(i) = i$ for all $i = 1, \dots, n$. For each of the different versions of REM as well as STS up to 10000 iterations have been performed. For our implementation of SSA1 we assume the exploration of ten trajectories starting from the same starting solution. Correspondingly, the number of iterations is divided among 'ten runs' (representing ten parallel lists) each of them with up to 1000 iterations.

The results show that REM is superior to the placement heuristics for nearly all TBP instances. Moreover, the behaviour of the computation times for REM increases more smoothly, so that even for larger instances REM produces results in acceptable time, whilst for problems with $n \geq 30$ especially GHL2 is too time-consuming. The results provided by the rotational heuristics are in some cases better than those of REM, however, the maximum deviation of the results may be much larger. REM proves to be quite robust in this respect, i.e., it produces consistently good results for all instances. Since the rotational heuristics refer to a placement algorithm computation time is a critical point all the more.

Special attention should be given to changes in solution quality with respect to the different diversification approaches and techniques. For both REM and SSA1 the elongation of tabu tenure from 1 to 10 iterations results in an improvement for most of the larger instances. The same applies for SSA2.

For the smaller instances represented in Table 1 the main conclusion with respect to diversification by using distance measures is as follows: The more diversification, the worse are the results provided that a limited number of 10000 iterations is performed. However, this picture is changing for larger instances, i.e. for $n \geq 25$, where better results are obtained for a more diversified search.

From these results it can be deduced that with this simple diversification approach the performance of the REM can be improved, provided that the size and structure of the solution space requires more diversification than given by a single trajectory.

In contrast to this the second and more complex star-shaped

approach with disjoint collections of solutions does not show a significant improvement with respect to its performance. In search of an explanation for this discrepancy it should be mentioned that even disjointness of the collection of solutions does not automatically lead to a spread of the trajectories over the solution space in the desired form. As a worst case the different trajectories could extend very close to each other only ensuring that no duplication of solutions occurs. One possibility to avoid such situation where the solutions do not appear to be sufficiently scattered is the additional use of distance measures as discussed above. Applying REM_t could be recommended as another appropriate method for achieving scattered trajectories in case that the conventional REM does not fulfil the desired search diversification.

Although there is no unique picture for most problems STS is not able to outperform the REM and its variants. The two lower lines of Table 1 show some aggregated results, i.e. the sum of all deviation values as well as the number of instances where the specific method obtained the best result out of all improvement methods under consideration. These values underline the above mentioned findings. Note, however, that our implementation of STS starts at the same solution without any randomization.

A remark deserves the number 10000 of iterations performed for REM especially when thinking about the relatively large number of iterations reported in the literature to solve larger problem instances (see, e.g., Voß (1995) or some references in Pardalos and Wolkowicz (1994)). The relatively even picture of the results be it with or without diversification is a clear indicator that more elaborate testing with respect to larger iteration numbers may be helpful. Especially for problem instances with $n \geq 30$ diversification pays when the number of iterations is considerably increased. It should be mentioned that the results confirm the findings of other studies (see, e.g., Voß (1993)) in that the REM by itself incorporates an intensification nature that may be overcome by some simple ideas as provided by SSA1 and SSA2 especially for larger instances. In this respect it seems to be worth to take the best found solution as a starting solution for another, say, 100000 iterations. With this approach for the price of large CPU times, most REM approaches and STS are able to produce improved solutions for some of the problems under consideration.

6. Conclusions

In this paper we have applied simple heuristic solution schemes including different tabu search approaches to the turbine runner balancing problem, which may be modelled as a special case of the quadratic assignment problem. For some benchmark problems taken from the literature we have provided better solutions than previously reported.

The main conclusion is that for the REM two simple and easy to implement methods for scattering the search trajectories over the solution space - (1) elongation of tabu tenure and (2) using distance measures - prove to be more effective than a more complex and even more time-consuming approach.

Further research with respect to the greedy heuristic and its modifications may be devoted to a more systematic investigation of look ahead methods, not only for the TBP and the QAP but in a general context. Further ideas in this respect concern ingenious ways of storing intermediate results to curtail the computational burden for further iterations. In addition, one should think about the possibility of approximating the calculations needed for the look ahead approach in a reasonable way, i.e., trying to find a good compromise between computational accuracy and CPU time.

A final remark refers to the obvious suitability of the star-shaped approach to be implemented in parallel.

Acknowledgement

The helpful comments of two referees with respect to the numerical results and additional references are greatly appreciated.

7. References

- R. Battiti and G. Tecchiolli, The reactive tabu search. *ORSA Journal on Computing* 6 (1994) 126-140.
- R.E. Burkard, S. Karisch and F. Rendl, QAPLIB-a quadratic assignment problem library. *European Journal of Operational Research* 55 (1991) 115-119.
- J. Chakrapani and J. Skorin-Kapov, Massively parallel tabu search for the quadratic assignment problem. *Annals of Operations Research* 41 (1993) 327-341.
- F. Dammeyer and S. Voß, Dynamic tabu list management using the reverse elimination method. *Annals of Operations Research* 41 (1993) 31-46.

- Y. Fathi and K.K. Gajjupalli, A mathematical model and a heuristic procedure for the turbine balancing problem. *European Journal of Operational Research* 63 (1993) 336-342.
- F. Glover, Tabu search - part II. *ORSA Journal on Computing* 2 (1990) 4-32.
- F. Glover and M. Laguna, Tabu search. In: *Modern Heuristic Techniques for Combinatorial Problems*, ed. C.R. Reeves (Blackwell, Oxford, 1993) 70-150.
- J.P. Kelly, M. Laguna and F. Glover, A study of diversification strategies for the quadratic assignment problem. *Computers & Operations Research* 21 (1994) 885-893.
- G. Laporte and H. Mercure, Balancing hydraulic turbine runners: A quadratic assignment problem. *European Journal of Operational Research* 35 (1988) 378-381.
- J. Mosevich, Balancing hydraulic turbine runners - A discrete combinatorial optimization problem. *European Journal of Operational Research* 26 (1986) 202-204.
- P.M. Pardalos and H. Wolkowicz (eds.), *Quadratic assignment and related problems*. (DIMACS Series Discr. Math. and Theoret. Comp. Science, American Math. Society, Providence, 1994).
- M. Sinclair, Comparison of the performance of modern heuristics for combinatorial optimization on real data. *Computers & Operations Research* 20 (1993) 687-695.
- J. Skorin-Kapov, Tabu search applied to the quadratic assignment problem. *ORSA Journal on Computing* 2 (1990) 33-45.
- P. Soriano and M. Gendreau, Diversification strategies in tabu search algorithms for the maximum clique problem. Working Paper CRT-940, Centre de recherche sur les Transports, Montreal (1993).
- E. Taillard, Robust taboo search for the quadratic assignment problem. *Parallel Computing* 17 (1991) 443-455.
- E. Taillard, Comparison of iterative searches for the quadratic assignment problem. Working Paper CRT-989, Centre de recherche sur les Transports, Montreal (1994).
- S. Voß, Tabu search: Applications and prospects. In: *Network Optimization Problems*, eds. D.-Z. Du and P.M. Pardalos (World Scientific, Singapore, 1993) 333-353.
- S. Voß, Solving quadratic assignment problems using the reverse elimination method. In: *The Impact of Emerging Technologies on Computer Science and Operations Research*, eds. S.G. Nash and A. Sofer (Kluwer, Dordrecht, 1995) 281-296.

Communication Issues in Designing Cooperative Multi-Thread Parallel Searches

Michel Toulouse^{1,2}

Teodor G. Crainic^{1,3}

Michel Gendreau^{1,4}

¹ Centre de Recherche sur les Transports
Université de Montréal

C.P. 6128, Succursale Centre-Ville
Montréal, Canada, H3C 3J7

² École Polytechnique

Campus de l'Université de Montréal

³ Département des sciences administratives
Université du Québec à Montréal

⁴ Département d'informatique et de recherche opérationnelle
Université de Montréal

Email: toulouse, theo, michelg@crt.umontreal.ca

Abstract:

Roughly speaking, parallel local search techniques can be divided into three categories: low-level parallelization strategies (e.g., master-slave schemes), solution-space partitioning methods and multi-thread procedures in which several processes explore concurrently the same search space. The multi-thread technique can be further subdivided into independent and cooperative search thread algorithms. In this paper, we focus on cooperative multi-thread heuristics applied to methods such as tabu search, and attempt to identify the key questions to be addressed in the design of any algorithm in this class. In particular, we show that questions related to inter-agent communications are a central element of the algorithmic design of these methods.

Key words: Combinatorial optimization, cooperative heuristics, iterative methods, parallel algorithms design.

1. Introduction

One of the simplest and most often used techniques to parallelize iterative heuristic search methods such as tabu search (TS) performs many concurrent independent search threads on the same problem. This approach have been used to parallelize the TS method applied to the QAP [1, 15], the Job Shop Scheduling problem [16], the location-allocation problem with balancing requirements [6]. Parallelization based on independent search threads is computationally equivalent to a sequential approach where repeated executions of the same algorithm are performed with different initial solutions and eventually different search parameters. The behavior of these algorithms approximates an exponential distribution with respect to the probability of failure $p(t)$ to find the optimal solution before iteration t [1, 15]. As reported in the taxonomy of Crainic, Toulouse and Gendreau [5], the parallelization of a method such as TS can also be realized based on knowledge sharing and cooperation between the different search threads. Huberman [10] conjectures a performance improvement when concurrent search threads are engaged in a cooperative strategy rather than performing independent searches. Experimental observations with concurrent asynchronous interactive TS [4] and simulated annealing procedures [9] seem to confirm the probabilistic model of Huberman.

Cooperative heuristics seem to be a promising parallelization paradigm for iterative search techniques such as TS. One can conceive the design of cooperative heuristics as based on independent threads, plus the specification of a coupling among the independent search threads to realize the sharing of knowledge. The term *coupling* refers to a theoretical concept used in the fields of software engineering and complex system theory. This concept measures the interaction between modules of a program and subsystems of complex structures. Transposed in the domain of standard parallelizations (e.g., based on data or program partitioning), this term refers to the data and control dependencies between concurrent tasks.

The definition of a coupling in standard parallelization based on data dependencies concerns what data are exchanged, when and among which processes. These dependencies between concurrent tasks are never explicitly expressed in the design of a standard parallel algorithm. They arise implicitly from the granularity of the

parallelization, and the data and function partitioning of the sequential algorithm. Parallelizations based on independent search threads do not involve any partition of the solution space or of the sequential program, and the granularity of the parallelization is at the sequential algorithm grain size. Therefore, one cannot specify the coupling of cooperative heuristics based on implicit dependencies between independent search threads. Consequently, in designing cooperative TS based on independent search threads, significant attention and effort should be devoted to the specification of the information sharing process among threads.

The present paper discusses a set of key issues that should be addressed to create cooperative algorithms based on independent search threads. We have organized the presentation of these issues around the explicit design formulation of the three main aspects of the coupling related to data dependencies in standard parallelizations: the definition of *what* information to share, *when* it should be shared, and between which independent threads (*where*) the information should be shared. The explicit statement of these three parameters in the design of a particular cooperative TS algorithm constitute the specification of a coupling between the independent search threads. Each coupling defines the interaction between threads which aims to create a cooperative behavior in the resolution of a combinatorial optimization problem.

This paper is organized as follows. In section 2, we restate aspects of the sequential tabu search method. Section 3 presents an independent search thread parallel algorithm for a location-allocation problem. Sections 4, 5 and 6 discuss the three main issues addressed in the paper. Section 7 presents the result of a cooperative multi-thread algorithm and finally, section 8 concludes the paper.

2. Sequential Tabu Search

Tabu Search (TS) is an iterative, high level meta-heuristic aimed to determine good approximate solutions to combinatorial optimization problems of the form:

$$\min\{c(s)|s \in X\}$$

where X is a set of feasible solutions and c a cost function.

At each TS iteration, a neighborhood $N(s)$ of the current solution s is defined by the application of a transformation rule t to the

solution s . $N(s)$ is the set of solutions that can be reached directly by the application of the transformation rule to s . One TS iteration performs the following operations:

1. Evaluate the neighborhood $N(s)$;
2. Select a solution $s' \in N(s)$ according to some heuristic evaluation criterion;
3. Apply the corresponding transformation (move) to s ;
4. $s = s'$, whether $c(s')$ is better than $c(s)$ or not.

The execution of the tabu program defines a directed graph where the solutions s and their associated neighborhoods $N(s)$ are the nodes, while moves define the arcs. The sequence of solutions chosen at each iteration forms an oriented path, the *search path*.

Generally, TS implementations do not keep track directly of the search path. Rather, indirect records of the search path are kept through different *memories*. More specifically, memories store information on attributes of the solution s and the neighborhood $N(s)$ of each tabu iteration. Records of the search path are not kept permanently. In fact, some of them only last for few iterations, such as those which forbid specific moves. Indeed, the choice of the move applied to s can be restricted at each iteration to a proper subset $V^* \subseteq N(s)$. This prevents the risk of cycling by excluding from $N(s)$ some of the most recently visited solutions in the search path. The solutions in $N(s) \setminus V^*$ are said to be *tabu* and the set of these solutions is referred to as (or through) the *tabu list*. Tabu lists are also referred to the short-term memory of the TS method.

We use the expression *heuristic evaluation criterion* in step 2 of a tabu iteration to emphasize the fact that the choice of s' is not entirely related on descent evaluation schemes. Tabu search is a guided iterative method where constraints like short term tabu lists not only prevent cycling but also determine the direction of the search. Secondly, typical TS algorithms display a variety of moves that are outside the realm of any descent evaluation scheme. For example, the choice of a “restarting” solution s' to begin an intensification or a diversification sequence depends mainly on memories responsible for collecting information on the medium and long term behavior of the search path. This is nevertheless an important feature of the method and it strongly influences the search path.

A TS execution starts with a predetermined setting of decision variables of the problem application. We name *initial solution* the solution s that results from this predetermined setting. A *search strategy* corresponds to the combination of an initial solution and a particular setting of the tabu search parameters (the various tabu list sizes, seeds for random functions, the stopping criterion and the segment exiting criteria, etc.). For a more complete description of the tabu search method see ([7, 8]).

3. An Independent Search Thread Parallelization

The following parallel algorithm is an example of a tabu search parallelization using independent search threads, each thread corresponding to a different search strategy:

- Create as many processes as there are known interesting search strategies or processors available for the concurrent execution of the algorithm;
- Run all processes concurrently;
- If s^a is the best solution of process a , $s^* \in \arg \min c(s^a)$, $a = 1, \dots, n$.

We have implemented this algorithm for a location-allocation problem using the sequential tabu search procedure [6]. To illustrate, Table 1 summarizes the performance results for 12 (rows P1 to P12) instances problem using 8 of the best know search strategies (columns S1 to S8) for the sequential tabu search procedure. The entries (P_i, S_j) show the relative gap (in %) between the solution obtained by process S_j for the instance P_i and the optimal solution of the same instance. The first column (Seq) displays the results obtained for these instances by the sequential tabu search procedure with the best reported search strategy for the problem [3]. These results are compared with those of the parallel algorithm based on independent search threads (column IST). All tests have been run for 300 tabu iterations.

Clearly, the parallelization based on independent search threads largely outperforms the sequential implementation. This is remarkable with respect to the fact that the parallel setting is obtained without any effort. In the three next sections we describe the design step that transform an independent search thread algorithm like this one into a cooperative algorithm.

Problem	Seq.	IST	S1	S2	S3	S4	S5	S6	S7	S8
P1	0	0	.41	0	0	0	.41	0	0	0
P2	.42	.21	.21	.33	.38	.99	.26	.23	.59	.29
P3	.01	0	.03	.63	.57	.57	0	.94	.03	.56
P4	.9	.32	.61	.58	.61	1.12	.32	.37	.39	.88
P5	.77	0	1.25	1.25	.57	.50	0	.90	1.10	1.06
P6	.42	0	.63	.15	.15	.05	.44	0	.05	.15
P7	.07	0	.13	3.69	.73	.73	0	.07	2.19	.07
P8	2.15	0	0	1.32	2.53	2.75	2.68	2.68	0	.85
P9	0	0	1.27	.82	.82	0	0	1.44	0	.82
P10	1.49	.37	1.58	1.59	2.50	1.16	2.33	2.37	2.47	.37
P11	1.52	0	.61	0	3.52	1.38	0	3.22	.61	1.52
P12	.11	0	2.59	1.80	1.46	1.79	0	1.09	0	.11

Table 1: Computation results with independent search threads

4. What Information to Share Between Search Threads

We see three possible gains from information sharing among search threads:

- Heuristic evaluation mechanisms may access memories of many threads, therefore they may be “better informed”;
- Due to the opportunities offered by a coupling between threads, new heuristic evaluation mechanisms may be used to improve a method like TS;
- Additional information relative to the solution space could be induced from cross-examination of memories of different threads.

4.1 Increasing the Knowledge of Heuristic Evaluation Mechanisms

Many heuristic evaluation mechanisms designed for sequential or independent multi-thread TS algorithms depend upon their knowledge of the solution space to be effective. In sequential implementations, these mechanisms increase their effectiveness as the search progresses. In a cooperative scheme, one can easily see how the sharing of memories associated to such heuristic evaluation mechanisms will increase their effectiveness. Let $T = \{t_1, t_2, \dots, t_p\}$ be a set of p threads involved in a multi-thread tabu search computation.

When each of the p independent threads has executed t iterations, each thread has accumulated in its memories a knowledge equivalent to no more than t sequential iterations. In the corresponding cooperative computation, after t iterations, the knowledge accumulated by each thread through the shared memories is equivalent to $p*t$ iterations of a single independent search thread. The increase in the effectiveness of the associated heuristic evaluation mechanisms should follow a similar relation.

The previous relation is a crude approximation since many variables that affect the effectiveness of heuristic evaluations or the global benefit of cooperative computation are not considered. In this section, we examine two of these variables. One of them concerns the communication costs inherent to the sharing of memories between threads, which independent thread computations do not pay. For example, memory contents that last for a relatively short period of time, such as short term tabu lists, will request frequent inter-processor communications. Let it and ct be the times required to perform t iterations by p independent and cooperative threads, respectively. The following relation can be used to evaluate the usefulness of shared memories:

$$it \leq ct \leq \max\{it, \frac{\text{best independent search solution}}{\text{best cooperative search solution}} * it\} \quad (1)$$

This constraint might be overrestrictive for some applications, and it can be relaxed based on experimental observations. Nevertheless, cooperative algorithms based on sharing short term tabu lists will almost certainly not meet this constraint. On the other hand, the contents of shared memories which become obsolete slowly, such as medium or long term memories, should request fewer computer network transfers, and are better suited for cooperative algorithms.

A second variable one has to consider refers to the evaluation of how *meaningful* the memory contents of a thread t_i are to several other threads $t_j \in T$, e.g., could the memory contents of thread t_i be of any use to the decision process of a thread $t_j \neq t_i$? A simple case concerns the short term tabu lists, which are hardly useful to any other threads (there is no interest in a normal implementation of a tabu mechanism to transfer the tabu status of a move to the context of another search path). Unfortunately, in general, the establishment of how meaningful the memory contents

of a particular thread are for many other threads is not easy to represent through a quantitative model. We rather use a qualitative model such as: *the memory contents can be interpreted as describing the problem fitness surface*. Take, for example, the medium term memories aimed to define regions of the solution space where intensification sequences can be performed. Quite often, their contents is correlated with interesting values of the cost function $c(s)$, and hence they say something about the fitness surface of the solution space. They may, therefore, be used in the context of any search path to initiate intensification sequences.

A comparison of Table 1 (section 3) and Table 2 (section 7) illustrates how better informed heuristic evaluation mechanisms, based on information sharing, can improves the independent multi-thread computation. Variations in the respective degree of success of the independent threads may be observed. For example, in Table 1, the best solution found by thread $S6$ has a gap of 3.22% for problem instance $P11$, while the gap of thread $S2$ is 0%; also for problem instance $P7$, thread $S6$ found a best solution at .07% of the optimal solution while thread $S2$ stopped with local optimum at 3.69% of the optimal solution. The gap differences indicate how appropriate were the initial solutions and search parameters to the different problem instances. Given that there were relatively few (300) tabu iterations allocated to the computation, threads that started with inappropriate initial conditions could not overcome their handicap and concluded with poor local optima. Meanwhile, as show in Table 2, unfit initial conditions loose rapidly their influence when information from more successful threads overtake the control of threads with unfit strategies. The results emphasize that information sharing may remedy to the problem of discrepancies in the quality of solutions found by different threads.

4.2 Design of New Heuristic Evaluation Mechanisms

Given the existence of a coupling between threads, one can think of modifications to the TS method to include new heuristic evaluation mechanisms or distributed versions of existing ones. Similar ideas are reported for iterative search methods such as genetic algorithms [14] and simulated annealing procedures [11] where design transformations are introduced to take advantages of the coupling of otherwise independent search threads.

In the TS context, independent search threads lose efficiency due, for example, to the inter-thread cycling phenomenon, where the same move is performed by several threads. The addition of a distributed tabu list mechanism to the independent search threads could control this problem. The idea is to use a distributed tabu list to make tabu to all threads in T a move m that has already been performed by another thread. One should observe, however, that a complete prohibition of inter-thread cycling will over-constrain search paths. Therefore, each thread could, for example, broadcast periodically the moves it has performed, and the moves become tabu for a duration specified by the tabu list size. The periodicity of the broadcast is a parameter used to prevent over-constraining search path and to control the variable ct such that it respects the constraint (1).

Each such heuristic evaluation mechanism requires particular information. However, in all cases, this information has to be considered as part of “what” information should be shared in the design of cooperative TS algorithms.

4.3 Creation of Information Relative to the Solution Space
Additional information relative to the solution space can be obtained by performing cross-examination processes on the shared memories. For example, some parallel genetic algorithms use descent methods to find local optima and then apply a crossover operator to these local optima [13]. One can think of the local optima found by the genetic algorithm threads as shared memories, and interpret the crossover operator as a process to create additional information on the solution space: the offsprings. This particular utilization of the crossover operator constitutes an example of a class of cross-examination processes that can be used by cooperative TS to create additional information. Other potential classes of cross-examination processes could be constituted by the study of the distributions of attributes of the search paths, such as the local minima found, the frequency of particular moves or blocks of moves in good or bad solutions, the average effect of a move on the objective function, etc. The various memories of the TS method offer a great potential to gain extra knowledge relative to the solution space, at the cost of few inter-processor communication and processing operations. This additional knowledge can then be used

by the heuristic evaluation mechanisms.

5. When Information Exchanges Should Take Place

Cooperative TS algorithms with unrestricted access to shared knowledge may experience serious premature convergence problems as do genetic algorithms when the offsprings of the best fit individuals compromise the diversity of the population. Genetic algorithms control this problem by using probabilistic techniques aimed at increasing the diversity of the population. The premature convergence problem of cooperative TS algorithms is somewhat different, since search parameters already possess the capacity to differentiate the search exploration. Rather, the problem arises when the contents of shared memories becomes stable and biased by the best moves executed by the different search threads, due to frequent accesses by the heuristic evaluation mechanisms to the shared knowledge which disseminate the same information in all the memories, including the non-shared ones. This uniformization of the information used by the heuristic evaluation mechanisms has a detrimental effect on the capacity of local search parameters to significantly influence the exploration of the search space.

In order to prevent the collapse of the cooperative search into similar search paths, we should seek an approach that achieves a balance between the influence of shared information and local search parameters. Therefore, the explicit statement of what information to exchange between threads has to be followed by algorithmic design decisions in order to define and control the “when” search threads may access the shared knowledge.

Access conditions to shared knowledge can be implemented for each particular heuristic evaluation mechanism, or they could be mechanism-independent and govern a whole set of heuristic evaluations. Moreover, the parameters to these conditions could be either predefined and fixed for the search, or they could vary dynamically according, for example, to the evolution of the differentiation of the search paths. To illustrate a condition defined specifically for a particular heuristic evaluation mechanism, assume that the best solution s^* ever found by one of the threads in T is considered to be shared; hence, s^* is a shared variable. Let s_i be a non-shared variable representing the best solution ever found by thread t_i . Suppose

that the heuristic evaluation mechanism h decides on the regions of the solution space where diversification sequences should begin based on a solution s . Access to the shared knowledge s^* by h could be constrained by the following relation for each thread t_i :

$$s = \begin{cases} s^* & \text{if } (\frac{i}{\alpha} * s^*) < s_i \\ s_i & \text{otherwise.} \end{cases} \quad (2)$$

where the constant α is problem dependent. The relation (2) states that the heuristic evaluation mechanism h will access the shared knowledge s^* only if the benefit for thread t_i and the cooperative computation is worth a change in the course of the search path of thread t_i set by its local search parameters. Conditions like this one are easy to implement, and have been proved to be sufficient to control the influence of shared information on search paths.

It is noteworthy that dynamically modifying the conditions to access shared knowledge may be used to induce specific collective behaviors among the search threads. For example, if rapid convergence to some local optimum is seek, access to shared information should be relaxed. This will drive many processes towards the same region of the solution space, and thus simulate an intensification sequence. If on the contrary, a smoother descent is desired, the criteria should be tightened. It is also possible to have a collective movements of diversification by preventing any access to shared information which returns the complete control of the search exploration to the local search parameters. If the restrictions to shared knowledge are changed alternatively, one can drive the exploration of the solution space toward phases of collective intensification and diversification.

6. Between which Threads Information Exchanges Take Place

In this section, we elaborate on aspects of the coupling that concern the *graph of sharing*, i.e. the logical links which connect threads to each other and control the propagation of shared information. But first we introduce the concept of a communication structure to support the notion of a graph of shared information.

Synchronous execution of standard (e.g., based on data or program partitioning) parallel iterative algorithms is divided into phases

by rendez-vous points that have to be reached by all processes before being permitted to continue their processing [2]. Assume, as in section 4.2, that T is a set of p threads performing a standard parallel iterative algorithm in synchronous mode. As thread t_i reaches a rendez-vous point k , it reads data from a subset of threads $S_{i,k} \subseteq T$, data needed by t_i to pursue the computation past the rendez-vous point. A less strict version of data exchanges between threads is possible when the algorithm is performed in asynchronous mode. A thread t_i may pursue its computation past a rendez-vous point with data from some threads $t_j \in S_{i,k}$ being out-of-date, because, threads t_j lagged behind t_i and did not update the data when thread t_i was ready to proceed to the next step.

One can define the *communication structure* of the synchronous execution of a standard parallel iterative algorithm as the set of threads involved in each data exchange during the execution. Assuming that the parallel computation is divided into m consecutive phases by $m - 1$ rendez-vous points, the communication structure \mathcal{S} of a parallel iterative algorithm may be expressed by:

$$\mathcal{S} = \{S_{i,k} : i = 1, \dots, p; k = 1, \dots, m - 1\} \quad (3)$$

Assuming that the reading of out-of-date data does not account for a meaningful data exchange, the communication structure \mathcal{A} of asynchronous computation is given by:

$$\mathcal{A} = \{A_{i,k} : i = 1, \dots, p; k = 1, \dots, m - 1\} \quad (4)$$

where $A_{i,k} \subseteq S_{i,k}$. The sets $S_{i,k} \subseteq T$ which communicate data to thread t_i at a rendez-vous point k are implicitly defined by the sequential algorithm and its standard parallelization. From the relation $A_{i,k} \subseteq S_{i,k}$, one can see however that the communication structure depends to some extend on the synchronization mode for standard parallelizations of iterative algorithms.

In cooperative algorithms, the sets $S_{i,k}$ are replaced by accesses to shared information. As mentioned in the introduction, there is no implicit definition of the coupling between independent threads that would define which threads access the shared knowledge at a given synchronization rendez-vous point k . Therefore, one can not assume an implicit communication structure between the cooperative threads: one has to state explicitly this communication structure as part of the cooperative algorithmic design. However,

similarly to standard parallelization, a relationship exists between the synchronization mode and the communication structure, and one still faces the decision to run the computation in either one of the two synchronization modes.

A natural and problem independent design development strategy is to fix the communication structure based on the synchronization mode. To illustrate, consider two very typical synchronization modes. In a synchronous mode, all threads synchronize at rendez-vous points, which results in the following communication structure \mathcal{TS} :

$$\mathcal{TS} = \{S_k = T, k = 1, \dots, p\} \quad (5)$$

where S_k is the set of threads which access the contents of the shared knowledge at rendez-vous point k . All threads write and read the shared memories at every rendez-vous point before being permitted to resume their processing. In an asynchronous mode, there is no rendez-vous point to access shared information. Read or write of shared memories proceeds according to the internal logic of the individual threads. The communication structure is given by \mathcal{TA} :

$$\mathcal{TA} = \{A_k, k = 1, \dots, t\} \quad (6)$$

where $A_k \subseteq T$ is the set of threads that access the shared knowledge at their iteration k . Obviously, if the parallel cooperative execution proceeds according to communication structure \mathcal{TS} , every thread receives the shared information at each rendez-vous point. On the other hand, according to the communication structure \mathcal{TA} , shared information propagation is based on a thread by thread information exchange (since the computation is asynchronous, there is no guarantee that iteration k will be performed at the same time by different threads). As observed in section 5, there is a complex interaction between the shared knowledge and the search parameters. It is interesting to look into this interaction relatively to the communication structure between search threads.

6.1 Interaction Between Shared and Non-shared Information

We now examine how shared knowledge and search parameters interact to control the exploration of the solution space by cooperative algorithms. Assume that s^* , s_i and h are defined as in section 5. Let

l_i be a non-shared data structure representing the short term tabu list of thread t_i . The access to shared knowledge for each thread is constrained by relation (2). Finally, assume that the parallel computation proceeds in synchronous mode, divided into m phases by $m - 1$ rendez-vous points Π . At a rendez-vous point Π_k , each thread $t_i \in T$ writes the contents of s_i into the shared variable s^* , if $s_i < s^*$. Before each thread can resume its computation, it reads the shared variable s^* into a temporary local non-shared variable.

During the first computation phase of the cooperative algorithm, the progress of the computation is identical to an independent search thread parallel algorithm. In the second phase, depending on the local search parameters which control the diversification move, a subset of threads $D \subseteq T$ will perform a diversification move. According to equation 2, for a subset $U \subseteq D$, $s = s^*$, the proposed solution from the shared knowledge, while $s = s_i$, the local best solution, for threads in $D \setminus U$. For $t_i \in U$, the tabu iteration following the diversification move is identical for each thread:

- $s_i = s^*$,
- The short term tabu list l_i will register s^* as a tabu move,
- The local search parameters will all be applied to the same solution s^* .

As the search progresses towards Π_2 , search paths from threads in U will diverge under the influence of their different local search parameters, as well as the contents of their non-shared memories not directly affected by solution s^* . As for the threads $t_i \in T \setminus U$, they will continue to behave like independent search threads during the sequence of tabu iterations between Π_1 and Π_2 .

It follows from this description that, for each rendez-vous point Π_k , the value of s^* depends on the information accumulated by the search threads in their non-shared memories s_i and l_i between Π_{k-1} and Π_k . On the other hand, the contents of each non-shared memory s_i between the rendez points Π_k and Π_{k+1} is partially determined by the shared variable s^* at synchronization point Π_k . This is a specific example of the dynamic interactions by which global knowledge and the non-shared information obtained by search threads influence one another. We refer to that phenomena as the *feedback loop* between shared knowledge and the search paths.

6.2 Comparisons Between Communication Structures

The impact of the feedback loop on parallel cooperative computations is as important as the initial solutions and the local search parameters. The communication structure is one of the factors that influence most directly the behavior of the feedback loop. We have performed an extensive set of tests to compare the impact of the communication structures based on synchronous and asynchronous cooperative search thread algorithms. These tests indicate that asynchronous information exchanges improve the performance of cooperative search thread algorithms over independent search thread algorithms [4]. On the contrary, synchronous information exchanges seem to have a negative impact on the performance of cooperative tabu for the location-allocation problem we studied. The parallel synchronous computation approach could not do as well as the independent search threads [6]. Two complementary hypotheses have been advanced to explain the disappointing results of synchronous cooperative algorithms. Both hypotheses are based on the relations between the communication structure and the feedback loop.

The first hypothesis is based on the control capacities of feedback loops in a system as a possible source of system self-organized behaviors [12]. Very briefly, cooperative TS will experiment self-organized behavior whenever two threads t_i and t_j interact such that

- at iteration x of thread t_i , the move chosen by t_i is based on shared knowledge relative to a state of thread t_j ,
- at iteration y of thread t_j , the move chosen by t_j is based on shared knowledge produced by thread t_i at its iteration x .

The exploration of the solution space performed by cooperative TS self-organizes since decisions to choose a move depend on these interactions between threads in addition to the initial solution and the search parameters. The interaction among threads is dynamic, triggered by events external to the search paths that modify the contents of shared memories. Rendez-vous points of synchronous computation are the main events that trigger interactions among threads. Since synchronization strategies are problem independent, generally either related to the elapsed time or to the number of iterations, they develop self-organized search strategies that concentrate on synchronization events. In asynchronous mode, on the

other hand, improvements to the value of the objective function are the events that trigger interactions among threads. Hence, the search strategies of synchronous computations are potentially less responsive to problem instance structures than those developed by asynchronous computations where read/write of shared memories proceed according to the internal logic of the threads. Therefore, according to this hypothesis, asynchronous computation would perform better than independent search threads because it has dynamic system capabilities to adapt the exploration of the solution space to the conditions of the problem instance and it perform better than synchronous computation because its dynamic system capabilities are activated by the solution space structure rather than the logic of the synchronization.

The second hypothesis is derived from cooperative multi-thread parallelizations of genetic algorithms [13] and simulated annealing [11] based on the *island model* from evolution theory. According to the island model, a population (the set of search threads) has a spatial structure which conditions the interaction between individuals in the population. The communication structure of asynchronous cooperative algorithms can be loosely portrayed as possessing the island structure. Referring to the model of section 6.1 but in asynchronous mode, if a thread t_i communicates (\Leftrightarrow) with thread t_j and $s_i < s_j$, the feedback loop will bring s^* to t_j . If $t_j \Leftrightarrow t_k$, and $s^* < s_j < s_k$, then one can say that s_i has propagated from t_i to t_k by a transitive movement involving t_j . The spatial organization of the communication structure of asynchronous computation is defined by the number of transition threads t_j used to mediate the propagation of s^* between threads. At the opposite, the broadcasts of the synchronization mode derive an uniform communication structure which pictures the set of p threads as a “single panmictic population” (in terms of the genetic theory formalization) since at a rendez-vous point, s^* is instantaneously known to every threads $t_i \in T$. According to Mühlenbein [13], there are reasons to believe that parallel genetic algorithms based on structured populations are more effective than synchronous parallel genetic algorithms which assume a single panmictic population.

In order to test this second hypothesis for the cooperative TS method, we have used a less typical approach to thread synchro-

nization in the design of a cooperative TS. In this new design, each thread t_i synchronize only with threads in $\mathcal{N}_{t_i} \subset T$, called the thread neighborhood of t_i . A thread has only one thread neighborhood but can be in the neighborhood of many other threads. Globally, the cooperative computation is synchronized but the shared information exchanges take place directly only between threads belonging to the same neighborhood. Since neighborhoods overlap each other, this new synchronization mechanism induces a communication structure in which shared information propagation corresponds to a diffusion process. Shared knowledge may have to go across many neighborhood layers before it can reach all search threads and, it may even never reach some of them. Following the model of section 6.1, at rendez-vous point Π_k , if $s_i < s_l \forall t_l \in \mathcal{N}_{t_i}$, then $s^* = s_i$ for all threads in \mathcal{N}_{t_i} . At rendez-vous point Π_{k+1} , neighborhoods \mathcal{N}_{t_j} that include at least one thread of \mathcal{N}_{t_i} will be such that if $s_i < s_l \forall t_l \in \mathcal{N}_{t_j}$, then $s^* = s_i$ for threads of neighborhood \mathcal{N}_{t_j} . This communication structure integrates to some extent the spatial structure constraints of the island model, and it approximates using the synchronous mode the shared information propagation of asynchronous cooperative algorithms. Performances of cooperative computations using this communication structure have been even better than classical asynchronous computation, finding the optimal solution for all the problem instances in Table 1.

Our tests have not confirmed the uniform communication structure as a successful approach to cooperative algorithms. Obviously, the asynchronous computation derives a communication structure far more complex although it does not perfectly correspond to the island model. These experimentations are an indication, however, that the “where” issue, which is the statement of a communication structure between threads, is another determinant factor to the design of cooperative algorithms. This issue is probably the one that allows one to gain the deepest insight into the issues of designing cooperative algorithm. Although we might find clues from theories such as the island model or from connectionist approaches such as neural network models, we believe that models which accurately capture the dynamics between shared and non-shared information could provide a more comprehensive representation of the behavior of cooperative parallel TS heuristics.

Problem	IST	P.C.	S1	S2	S3	S4	S5	S6	S7	S8
P1	0	0	0	0	0	0	.41	0	0	0
P2	.21	0	.39	.39	.39	0	.30	.30	.31	.39
P3	0	0	0	0	0	0	0	0	0	0
P4	.32	0	0	0	0	0	0	0	.27	0
P5	0	0	.49	.49	.49	.48	0	0	0	.14
P6	0	.05	.19	.05	.05	.05	.05	.15	.05	.05
P7	0	.07	.07	.07	.07	.07	.10	.07	.10	.07
P8	0	0	0	.79	0	.79	0	.79	.79	0
P9	0	0	0	.82	0	0	1.26	0	0	0
P10	.37	.14	.14	.75	.75	.14	.14	.14	.82	.60
P11	0	0	0	.33	0	0	0	.33	0	0
P12	0	0	.18	1.14	0	1.14	1.38	1.09	0	1.38

Table 2: Computation results of the cooperative search threads

7. An illustration of cooperative threads

To illustrate our discussion, we present the results of an asynchronous cooperative search thread algorithm, build by setting the three design issues in the following way:

- Threads share their best solution ever found, by using a pool of size one;
- A solution from the pool is considered useful if its value is lower than the best solution of the local thread;
- Information exchanges take place in asynchronous mode.

Table 2 summarizes the performance of this algorithm. The third column (P.C.), display the results obtained when threads exchange their best solutions. Similarly to the independent search threads (IST) algorithm, it found the optimal solution for problems P1, P3, P5, P8, P9, P11 and P12. On problem instances P2, P4 and P10 the P.C. algorithm did better than the IST one, while for P6 and P7 it did a little bit worst. Overall, the new algorithm improves over the IST approach for the value of the best solutions found. The results obtained for strategies S1 to S8 in Table 2 show a significant difference compared to those in Table 1. For 11 of the 12 problem instances, the gaps in Table 2 are lower and more uniform than the gaps in Table 1 for the independent search threads algorithm.

8. Conclusion

We have discussed in this paper three major issues that have to be addressed when using parallel cooperative search processes to resolve optimization problems: the information that is shared among threads, the access conditions to the shared knowledge, and the definition of a communication structure which controls the propagation of information.

Cooperative search opens up new design possibilities for the parallelization of heuristic search methods such as the TS method. In this paper, we have shown how distributed tabu lists could control repeated search patterns by many search threads. We have also shown how changes in the tightness of the cooperation among threads could induce different organized global behaviors like global intensification or diversification phases. We have some reasons to believe that the communication structure among threads and its influence on the feedback loop could be exploited to improve the efficiency of parallel heuristic computations.

Finally, all along this work, we have found that the cooperative schemes of TS, parallel genetic algorithm and parallel simulated annealing possess very similar characteristics. It would probably be very productive to examine a possible unification of these schemes for the three iterative methods.

Acknowledgments

Funding for this project has been provided by the Natural Sciences and Engineering Research Council of Canada, through its Research and Strategic Grant Programs, and by the Fonds F.C.A.R. of the Province of Québec.

References

- [1] R. Battiti and G. Tecchiolli. Parallel Biased Search for Combinatorial Optimization: Genetic Algorithms and TABU. 16(7):351–367, 1992.
- [2] D.P. Bertsekas and J.N. Tsitsiklis. *Parallel and Distributed Computation, Numerical Methods*. Prentice Hall, 1989.
- [3] T.G. Crainic, M. Gendreau, P. Soriano, and M. Toulouse. A Tabu Search Procedure for Multicommodity Loca-

- tion/Allocation with Balancing Requirements. *Annals of Operations Research*, 41:359–383, 1993.
- [4] T.G. Crainic, M. Toulouse, and M. Gendreau. Parallel Asynchronous Tabu Search for Multicommodity Location-Allocation with Balancing Requirements. Publication 935, Centre de recherche sur les transports, Université de Montréal, 1993.
 - [5] T.G. Crainic, M. Toulouse, and M. Gendreau. Towards a Taxonomy of Parallel Tabu Search Algorithms. Publication 933, Centre de recherche sur les transports, Université de Montréal, 1993.
 - [6] T.G. Crainic, M. Toulouse, and M. Gendreau. Synchronous Tabu Search Parallelization Strategies for Multicommodity Location-Allocation with Balancing Requirements. *OR Spektrum*, 17(2/3), 1995.
 - [7] F. Glover. Tabu Search - Part I. *ORSA Journal on Computing*, 1(3):190–206, 1989.
 - [8] F. Glover. Tabu Search - Part II. *ORSA Journal on Computing*, 2(1):4–32, 1990.
 - [9] D.R. Greening. Parallel Simulated Annealing Techniques. *Physica D*, 42:293–306, 1990.
 - [10] B.A. Huberman. The Performance of Cooperative Processes. *Physica D*, 42:38–47, 1990.
 - [11] P. S. Laursen. Problem-Independent Parallel Simulated Annealing using Selection and Migration. pages 1–10, 1994.
 - [12] A. Mees. Chaos in feedback systems. In Arun V. Holden, editor, *Chaos*. Princeton University Press, 1986.
 - [13] H. Mühlenbein. Evolution in time and space - the parallel genetic algorithm. In G. Rawlins, editor, *Foundations of Genetic Algorithm & Classifier Systems*. Morgan Kaufman, 1991.
 - [14] H. Mühlenbein, M. Gorges-Schleuter, and O. Krämer. New Solutions to the Mapping Problem of Parallel Systems - the Evolution Approach. *Parallel Computing*, 6:269–279, 1987.
 - [15] E. Taillard. Robust Taboo Search for the Quadratic Assignment Problem. *Parallel Computing*, 17:443–455, 1991.
 - [16] E. Taillard. Parallel taboo search techniques for the job shop scheduling problem. *ORSA Journal on Computing*, 6(2):108–117, 1994.

**A Study on Algorithms for
Selecting r Best Elements from An Array**

Fan T. Tseng

University of Alabama in Huntsville

Department of MIS/ECN/FIN

Huntsville, AL 35899

E-mail: FTseng@asb1.asb.uah.edu

Abstract:

The issue of identifying r best (smallest or largest) elements from a data set is relevant in a variety of combinatorial search settings, for example, some strategies of tabu search (Glover 1989, 1992, 1994, Lokketangen and Glover 1995), semi-greedy heuristics (Hart and Shogan 1987), and GRASP (Laguna, Feo, and Elrod 1994, Feo and Resende 1995). This is particularly true of aggressive search strategies such as probabilistic tabu search, which strongly bias the choice among elements to favor those with preferred evaluations. The identification of r best elements is also relevant for applying candidate list selection strategies within combinatorial search (e.g. Glover 1992, 1994). This occurs notably where compound moves are screened by decomposing them into components, as where exchange moves are broken into "add moves" and "drop moves," and r best options for each type of component are identified (and then combined) to restrict the number of compound moves examined. Additional applications to candidate list strategies occurs where distribution of the r best elements is monitored as a supplementary guideline for determining the suitability of the candidate list composition. In these instances, the most appropriate value for r is often relatively small, e.g. typically less than 20.

Because the need to identify r best elements occurs repetitively within such contexts, it is important to isolate these elements efficiently. The computer science literature proposes many algorithms for sorting data files, and a few have become widely adopted due to their convenience and efficiency. Yet, by comparison, little attention has been given to selecting r best elements from a list, and the issue of selecting such elements for relatively small values of r has received even less consideration. Conceivably, The worst case emphasis that pervades the computer science literature has served to draw the focus away from such issues, in spite of their potential practical interest.

In this paper, we have adapted a few of the more prominent sorting algorithms to the function of selecting r best elements and compare their performance. Experimental results show that a *hardwired* threshold algorithm (based on simple binary sorting) performs surprisingly well in selecting a relatively small number of the best elements from a larger collection.

Key Words: Sorting, heuristics, search methods.

1. Introduction

Given a list of n records or elements indexed by the set $N = \{1, \dots, n\}$ and an integer r , our goal is to find r best elements represented by indices $k \in N$ that yield smallest (or largest) values of a function $v(k)$. That is, we seek to identify a subset $R \subset N$ such that $|R| = r$ and $v(k) \leq v(h)$ for all $k \in R$ and $h \in N - R$. We also sometimes prefer to know a sequence of the elements of R that will allow the $v(k)$ values to be accessed in ascending order over this set. In the following, we refer to this problem as *selecting r best out of n* . A closely related problem is finding the r -th best out of n elements. Special cases for this problem include finding largest, smallest, and the median of the n elements. These related problems are often discussed in textbooks of computer algorithms, while the problem of finding r best out of n is seldom treated. It is not difficult to see that any internal sorting algorithm can

be tailored to solve the problem. Since only a portion of the elements is required, it should not take more time than sorting the whole list of elements.

2. Selecting r Best out of n

Blum *et al* (1972) presented a linear-time selection algorithm for finding the r -th smallest out of n elements. This algorithm is a variant of quicksort and has been described as probably the fastest algorithm to find the r -th out of n (Aho, Hopcroft, and Ullman 1983). Heapsort algorithm also can easily be modified for finding r best out of n .

Both heapsort (Williams 1964) and quicksort (Hoare 1962) have been proved to be very efficient in sorting n elements internally. Both have average computational complexity of order $n \log n$, while several other simple sorting algorithms such as selection sort, bubble sort and insertion sort, have complexity of order n^2 (see Knuth (1973)). For small n it is more efficient to use one of the simple algorithms due to their low overhead (e.g. Knuth 1973). For large n quicksort has the best average performance, although its worst-case performance is n^2 . Heapsort also performs very well and it requires only $n \log n$ running time in the worst case.

All these algorithms involve comparison and move (at times swap) operations. Here we briefly discuss some of them for selecting r out of n elements. For convenience we will use informal terminology where more rigorous descriptions are easily inferred from the context.

2.1 Selection Sort

Selection sort finds the k -th smallest element at iteration k . Since $k - 1$ smallest elements have been found at iteration $k - 1$, the approach simply finds the smallest of the remaining elements, requiring $n - k + 1$ comparisons to find the k -th smallest and one swap to place it at the k -th position at the end of the iteration. The length of the list examined is reduced by one in each iteration.

For k = 1 to r Do
 Find a smallest element from positions k through r;
 Swap to kth position;
 End

Figure 1: Selection sort

2.2 Heapsort

Heapsort creates a heap structure in which the kth element is not greater than $(2 * k)$ -th and $(2 * k + 1)$ -st elements. A procedure to create such a structure out of the data array is to arrange the k-th element and its descendants as a heap, for each k starting from $(\text{size}/2)$ down to 1. The smallest element is found at the top (first position) of the heap structure at the end of heaping. This first element is then swapped with the last element in the heap and excluded from consideration. The remaining structure now may not be a heap and is "reheaped", which takes only a relatively small number of comparisons and swaps. The smallest of the remaining elements will appear on the top of the heap after the reheap. It is then swapped with the last element of the remaining elements, which is the second to the last element in the entire data array. The process repeats until all r smallest elements have been found in order and placed at the end of the array in reverse order.

Create a heap for the list;
 Do k times
 Swap the root with the last in the heap;
 Exclude the element in the last of the heap;
 Reheap the remaining elements;
 End

Figure 2: Heapsort

2.3 Quicksort

Quicksort selects an arbitrary element called a *pivot* at each iteration. All elements under consideration in the iteration are partitioned into two sets, where the first set contains all elements smaller than the pivot and the second set contains all remaining elements. If the first set contains r elements, or contains r elements by including the pivot, the procedure is complete. Otherwise, if it contains more than r elements, the second set is discarded and the process repeats, while if it contains fewer than r elements, then these and the pivot are set aside as part of the final desired set, and the process repeats with the goal of finding r smallest elements of the second set, where r is now reduced by the number of elements set aside.

The algorithm described above is designed to partition the array into two sets, which is efficient for data with distinct values. For data with some equal values, a variant of the algorithm may be designed to partition the array into three sets, each of which includes those elements smaller than, equal to, and larger than the pivot, respectively. Both versions of the algorithm were compared in the experiments described later in this paper.

```

Find a pivot;
Partition using the pivot;
Let s = size of first partition;
IF s = r THEN quit
ELSE IF s > r THEN do Quicksort(first partition, r)
    ELSE let r := r - s
        do Quicksort(second partition, r);

```

Figure 3: Quicksort

The issue of selecting a pivot is also important. Many researchers have suggested that a pivot be determined from a small sample (instead of simply using one randomly selected element). It has been reported

that the adoption of a sample approach reduces the possibility of worst-case behavior and also gives a slight improvement to the average running time (Knuth 1973). In our algorithm, the median of three randomly selected elements is used as a pivot in each iteration.

The variant of quicksort described here is an accelerated version that does not necessarily identify the ordering of the r best elements (we save time by eliminating recursive calls required to sort the best elements.) Such a streamlined version is acceptable for some search methods that do not require information about the order of elements. For example, Some versions of probabilistic tabu search generate probabilities directly from positive values of the r best elements (mapping these values monotonically into positive values which are divided by this sum), and hence do not require knowledge of the ordering. The GRASP procedure simply chooses an element at random from the r best (during an iterative constructive phase), and so likewise is indifferent to the ordering. However, for methods that require an ordered list, additional steps to sort the r best elements have to be performed, reducing the speed of quicksort.

2.4 Hardwired Threshold Sort

Hardwired threshold sorting is based on the idea that in each iteration current r best elements are sorted. A new element is inserted into its proper position by a binary search-like procedure. Since only the best r elements are desired, a threshold is maintained equal to the current r -th element and all elements larger than the threshold can be discarded by a single comparison. The routine of maintaining the current r best has a resemblance to the insertion algorithm, which is a very inefficient sorting algorithm in general. (This latter type of sort inserts the element being considered into its position in the sorted sublist by shifting elements larger than the element in the sublist.) However, there is a key difference between the embedded threshold routine and the insertion algorithm. In the hardwired threshold sorting algorithm, a binary search scheme for finding the proper position for an element

is designed precisely for a particular value of r . For an r value not an integral power of 2, a search scheme which is a sharpened ("lopsided") version of binary search may be specially

```

Let the starting threshold value and the initial (dummy)
value for each of the best  $r$  elements be a large number;
For each element < threshold Do
    Insert element into the best  $r$  array by binary search with
    fixed parameters;
    Drop the largest element in the best  $r$  array;
    Update threshold as the new largest in the array;
End

```

Figure 4: Hardwired Threshold Sort

designed. We will use the notation hardwire- r for the specially designed hardwired algorithm for finding r out of n . For example, hardwire-16 is the hardwired sorting algorithm designed for $r = 16$. In the following we briefly sketch the algorithm, using a simplified notation whose meaning should be clear.

3. Experiments

To compare the algorithms discussed above for selecting the best r out of n , we make the following experiments. Twenty data files for each of the sizes 100, 200, and 400 and 1000 are created with elements randomly generated from a uniform distribution generator between 1 and 1000. For each $r = 8$ and 16, the four algorithms discussed above are tested with the specified problems. The algorithms are coded in Turbo Pascal and run on a 486 PC. Table 1 shows the results of the experiments for $r = 8$. Each run time shown in the table is the average time obtained by running the same problem thirty times.

Apparently, hardwire-8 runs somewhat faster than the other algorithms. As previously noted specialized variants can similarly be designed for values such as $r = 6$ and $r = 13$, or the structure of the

Table 1: Run time (in hundredths of a second)
for $r = 8$.

	100	200	400	1000
Hardwire-8	.03	.07	.16	.19
Quicksort(2-partition)	.14	.12	.23	.47
Heapsort	.17	.25	.30	.55
Selection	.20	.25	.40	.95

hardwired threshold method for $r = 16$ (for example) can be slightly "relaxed" to apply to other r values smaller than 16. Since a method such as probabilistic tabu search will often have a single preferred r value, we have selected specific values to indicate the performance of the approach.

Table 2 shows the results of the experiments for $r = 16$. Each run time shown in the table is the average time obtained by running the same problem thirty times.

Hardwire-16 has the best overall performance. When file sizes are 100 and 200, hardwire-16 and quicksort are about equal in performance. However, as previously noted, quicksort does not automatically yield the r selected elements in the sorted order (as can be valuable in some forms of probabilistic tabu search). As the file size gets larger, hardwire-16 leads significantly. One could expect hardwire-16 to lead for even larger files.

Several other types of data distributions are also tested. The results are shown in Tables 3 and 4. The data used in the experiment are described as follows. The right-skewed data files in group 1 are

Table 2: Run time (in hundredths of a second)
for r = 16.

	100	200	400	1000
Hardwire-16	.09	.12	.19	.25
Quicksort(2-partition)	.09	.12	.25	.48
Heapsort	.20	.19	.26	.52
Selection	.19	.34	.78	1.82

generated from a population distribution between 1 and 1000 with 90% of the values between 1 and 100, whereas the left-skewed data in group 2 have 90% of the values between 901 and 1000. Both are unimodal. The bimodal data files in group 3 are generated from a bimodal distribution between 1 and 1000 with modes at 200 and 800. The data in group 4 are generated from a uniform distribution between 1 and n, where n = file size / 10. On average, there are ten identical values generated in a sample for each element between 1 and n in the population. The data in group 5 are generated similarly except n = file size / 20, thus on average there are 20 identical values in the sample for each element between 1 and n.

As in the experiments above, each run time in the tables is the average of thirty runs on each of the twenty problems generated in the category. The hardwired threshold method is the winner in all types of data distributions considered. Noteworthy is the comparison between the two-partition and three-partition versions of the quicksort. The three-partition version is better only when there are many duplicates (for example, 20 duplicates per element) and file size is relatively small (100 to 200). The general effectiveness of the two-partition version may be due to an efficient approach used in managing the

movement of the indices in partitioning the file into two sets. The additional operations required by the three-partition version take more time than the saving that results from isolating all the identical values of the pivot at each iteration.

4. Conclusions

The need to select the best r out of n elements arises repetitively in a variety of applications of combinatorial search, therefore calling for an examination of methods to achieve this goal efficiently. In this paper we conducted an empirical study on several algorithms tailored from their sorting counterparts. Experimental results indicate that supposedly efficient sorting algorithm may not work as well as expected for finding r out of n in some situations. For example, quicksort, a method predominantly used in practice, does not emerge the winner when the desired value of r is relatively small. On the other hand, the hardwired threshold method, whose " n out of n " sorting is rather inefficient, proves to work exceedingly well for small values of r . Moreover, unlike the streamlined quicksort variant we employed in our testing, the hardwired threshold approach also orders all of the r selected elements. Finally, the coding for such a procedure is quite straightforward. The result is a useful tool that deserves consideration in applications commonly encountered in certain search procedures for optimization problems.

Table 3: Run time (in hundredths of a second)
for r = 8, using various distributions.

	Data type	File size	HW8	QS2	QS3	HP
1	Right-skewed	100	.06	.07	.14	.12
		200	.09	.14	.16	.25
		400	.11	.25	.27	.22
		1000	.22	.48	.65	.59
2	Left - skewed	100	.04	.11	.13	.13
		200	.10	.20	.13	.25
		400	.17	.18	.37	.37
		1000	.18	.27	.70	.66
3	Bimodal	100	.02	.12	.14	.13
		200	.06	.11	.21	.18
		400	.14	.22	.28	.25
		1000	.26	.55	.67	.64
4	Equal values (10)	100	.03	.10	.09	.10
		200	.07	.10	.16	.24
		400	.13	.18	.26	.28
		1000	.25	.47	.62	.57
5	Equal values (20)	100	.06	.13	.06	.08
		200	.08	.14	.19	.20
		400	.12	.23	.27	.24
		1000	.25	.52	.66	.65

HW8 : Hardwire-8

QS2 : Quicksort (two-partition)

QS3 : Quicksort (three-partition)

HP : Heapsort

Table 4: Run time for r = 16.

	type	File size	HW16	QS2	QS3	HP
1	Right-skewed	100	.09	.10	.15	.17
		200	.11	.21	.25	.13
		400	.17	.24	.33	.24
		1000	.27	.43	.68	.53
2	Left - skewed	100	.10	.16	.13	.20
		200	.07	.09	.20	.14
		400	.18	.26	.31	.29
		1000	.36	.51	.61	.58
3	Bimodal	100	.06	.07	.17	.14
		200	.14	.20	.27	.23
		400	.16	.18	.32	.25
		1000	.23	.44	.64	.69
4	Equal values (10)	100	.07	.10	.08	.16
		200	.15	.17	.12	.12
		400	.21	.22	.30	.32
		1000	.41	.49	.68	.61
5	Equal values (20)	100	.12	.23	.10	.11
		200	.09	.21	.12	.24
		400	.22	.28	.27	.28
		1000	.26	.55	.65	.72

5. References

- A.V. Aho, J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, (Addison-Wesley, 1983).
- M. Blum, R.W. Floyd, V.R. Pratt, R.L. Rivest, and R.E. Tarjan, Time bounds for Selection, *J. Comput. Syst. Sci.* 7 (1972) 448.
- T.A. Feo and M. G. Resende, Greedy Randomized Adaptive Search Procedures, *Journal of Global Optimization* (1995).
- F. Glover, Tabu Search: Part I, *ORSA Journal on Computing* 1, (1989) 190.
- F. Glover, Tabu Thresholding: Nonmonotonic Strategies for Combinatorial Optimization, Working paper, School of Business, University of Colorado (1992), to appear in *ORSA Journal on Computing*.
- F. Glover, Tabu Search Fundamentals and Uses, Working paper, School of Business, University of Colorado (1994).
- J.P. Hart and A.W. Shogan, Semi-Greedy Heuristics: An Empirical Study, *Operations Research Letters* 6 (1987) 107.
- C.A.R. Hoare, Quicksort, *Comp. J.* 5 (1962) 10.
- D.E. Knuth, *The Art of Computer Programming*. Vol. 3: Sorting and Searching (Addison-Wesley, Reading, MA 1973).
- M. Laguna, T.A. Feo and H.C. Elrod, A Greedy Randomized Adaptive Search Procedure for the Two-partition Problem, *Operations Research* 42 (1994) 677.
- J.W.J. Williams, Algorithm 232: heapsort, *Communications of ACM* 7 (1964) 347.

A Modified Tabu Thresholding Approach for the Generalised Restricted Vertex Colouring Problem

Vicente Valls

Dpto. Estadística e Investigación Operativa

University of Valencia

Dr. Moliner, 50. Burjasot, Spain

E-mail: vicente.valls@uv.es

M. Ángeles Pérez; M. Sacramento Quintanilla

Dpto. Economía y Matemática

University of Valencia

Avda. Blasco Ibáñez, 30. 46021 Valencia, Spain

E-mail: perezma@econom.uv.es

E-mail: Maria.Quintanilla@uv.es

Abstract:

We present a modification of the Tabu Thresholding (TT) approach and apply it to the solution of the generalised restricted vertex colouring problem. Both the bounded and unbounded cases are treated. In our algorithms, the basic TT elements are supplemented with an evaluation function that depends on the best solution obtained so far, together with a mechanism which reinforces the aggressive search in the improving phase, and new diversification strategies which depend on the state of the search. The procedure is illustrated through the solution of the problem of minimising the number of workers in a heterogeneous workforce.

Keywords: Generalised restricted chromatic number, graph colouring, tabu thresholding, workforce assignment.

1. Introduction

The problem of colouring the vertices of a graph with the smallest number of colours so that no two adjacent vertices receive the same colour (VCP) has been used to model diverse practical applications: Course Scheduling (Welsh & Powel (1967), Wood (1969)); School Timetable (Neufeld & Tartan (1974), Mehta(1981), de Werra (1985)); Cluster Analysis (Hansen & Delattre (1978)); Group Technology in Production (Chams et al. (1987)); etc.

Due to the variety of applications, many different methods have been presented to calculate the chromatic number. These include both exact approaches (Zykov (1952), Hammer & Rudeanu (1968), Brown (1972), Christofides (1975), Wang (1974), Korman (1979), Brélaz (1979), Kubale & Jakowsky (1985), Chu (1991)) as well as heuristics ones (Welsh & Power (1967), Wood (1969), Matula et al. (1972), Brélaz (1979), Leighton (1979), Dutton & Brigham (1981), Chams, Hertz & de Werra (1987), Hertz & de Werra (1987), Turner (1988); Morh (1988); de Werra(1990); Johnson et al. (1991)).

However, when certain restrictions are imposed, variants of these problems appear which are no longer equivalent to the VCP. In some cases, these variants can be modelled by introducing constraints on the colours available for each vertex of the graph. This gives rise to the restricted VCP, or RVCP (Neufeld & Tartan (1974), de Werra (1985), Kubale (1992)).

Kubale (1992) demonstrates that the RVCP is NP-complete even on some graph structures in which the VCP is polynomial, as in bipartite graphs. As far as we know, no solution procedure for the RVCP has been published to date.

In this paper, we consider a generalisation of the RVCP (GRVCP) that has not previously been dealt with in the literature. The classical RVCP assumes that there is a single unit for each colour (i.e. all the colours are different), and that two vertices linked by an arc (2-clique) cannot be assigned the same colour. In the GRVCP, this constraint is relaxed in the following way. For each colour a finite or infinite number of colour units are available and, therefore, two vertices joined by an arc can be painted with the same colour, though not with the same colour unit. In this case, the number of units available for each colour determines the maximum number of vertices of a clique that can be painted with a particular colour.

We define the RVCP as being a special case of the GRVCP in which the number of units available of each colour is equal to 1.

We distinguish two versions of the GRVCP depending on whether the number of units of each colour is or is not bounded (Bounded_GRVCP and Unbounded_GRVCP, respectively).

Numerous real-life problems can be formulated as GRVCPs, for example: the Number of Workers Minimisation Problem (NWMP) when considering a heterogeneous workforce; minimising the number of vehicles of different capacities needed to carry out a set of trips; minimising the number of different computers needed to carry out an engineering project; minimising the number of interpreters, trained in different languages, required for a set of parallel sessions; solving a timetable with restrictions, etc.

To give a specific example, in the NWMP we start from an established plan of jobs which must be carried out on certain machines. Each job must be undertaken by a single worker, but not all the workers can operate all the machines. As a result of being a heterogeneous workforce, the workers are partitioned into k types T_1, \dots, T_k ; each type T_i has a set of machines $M(T_i)$ it can be assigned to (i.e. the set of machines each worker of type T_i can operate) and $\exists i, j / M(T_i) \cap M(T_j) \neq \emptyset$. The aim is to assign the workers using a minimum number of them.

To formulate the NWMP as a GRVCP suppose each vertex represents a job; an arc between each pair of vertices signifies jobs which are carried out simultaneously; a colour is assigned to each type of worker, with as many colour units as there are workers available for each worker type. Furthermore, each vertex can only be coloured according to the type of worker capable of operating the machine needed for the job represented by the vertex. The number of colour units in the optimal

solution of the GRVCP coincides with the optimum of the NWMP for both the Unbounded_GRVCP and Bounded_GRVCP, in the context of infinite or finite availability of workers of each type, respectively.

In this paper we present an algorithm based on the Tabu Thresholding (TT) technique for both the bounded and unbounded GRVCP. To measure the efficiency of the algorithm a random set of instances of the NWMP has been generated, and the optimal solution compared with the solution calculated by the TT algorithm, once each problem has been transformed to a GRVCP. The optimal solution was obtained using the Branch and Bound algorithm for the NWMP presented in Quintanilla (1994).

TT is a variant of the basic Tabu Search (TS) proposed by Glover (1986). TS is a technique for solving difficult combinatorial problems based on general principles of intelligent problem solving. In essence it is a metaheuristic approach which may be used to guide any local search procedure in the active search for a global optimum using memory structures to avoid getting trapped in local optima.

Glover proposed the basis for the TT method. This procedure implement the fundamental concepts of TS without explicitly using memory structures and is easier to implement. Basically, TT combines elements of probabilistic tabu searching with candidate list procedures derived from network optimisation studies. This combination produces an implicit tabu threshold effect which emulates the functions of the memory structures in TS procedures. A more detailed description can be found in Glover (1992).

In our algorithms, the basic TT elements are supplemented with an evaluation function that depends on the best solution obtained so far, together with a mechanism which reinforces the aggressive search in the improving phase, and new diversification strategies which depend on the state of the search.

The paper is structured as follows. In section 2 we define the Unbounded and Bounded_GRVCPs. Section 3 provides a detailed description of the algorithm employed in the solution of the unbounded version, with particular reference to the modifications made to the usual TT approach. In section 4 we comment on the differences in the algorithm for the bounded case. Finally in section 5 we present a computational study and in section 6 we offer some concluding remarks.

2. The Generalised Restricted Vertex Colouring Problem

Let $G=(V,A)$ be a non-directed graph with vertex set V and arc set A . The elements which define the GRVCP on G are:

- A set of colours $\text{COL}=\{c_1, c_2, \dots, c_k\}$.
- A set of available colour units AVL_COL formed by a finite or infinite number of copies of each colour. Each one of these copies of a colour c_i will be called a **unit of colour c_i** .
- An application of permitted colours $P: V \rightarrow \mathcal{P}(\text{COL})$, where $\mathcal{P}(\text{COL})$ represents all possible subsets of COL . It is then said that $v \in V$ can be coloured or is colourable by the colours of $P(v)$.

Starting from these elements we define the following concepts:

- An application $C:V \rightarrow AVL_COL$ is a **COLOURING** of the vertices of G .
- An application $C:V \rightarrow AVL_COL$ is a **FEASIBLE COLOURING** of the vertices of G if $C(v) \in P(v)$, $\forall v \in V$ and $C(v) \neq C(w)$, $\forall (v,w) \in A$.
A feasible colouring C is said to use $|C(V)|$ colour units.
- A feasible colouring C of the vertices of G is an **OPTIMAL COLOURING** if there is no feasible colouring that uses fewer colour units. The number of colour units used in an optimal colouring is called the **GENERALISED RESTRICTED CHROMATIC NUMBER** of the graph G .

The **unbounded** version (**Unbounded_GRVCP**) assumes that AVL_COL is formed from an infinite number of units of each colour. In the **bounded** version (**Bounded_GRVCP**), the number of units of at least one colour is bounded.

Figure 1 depicts the differences among modelling and solving a problem as VCP, RVCP or GRVCP.

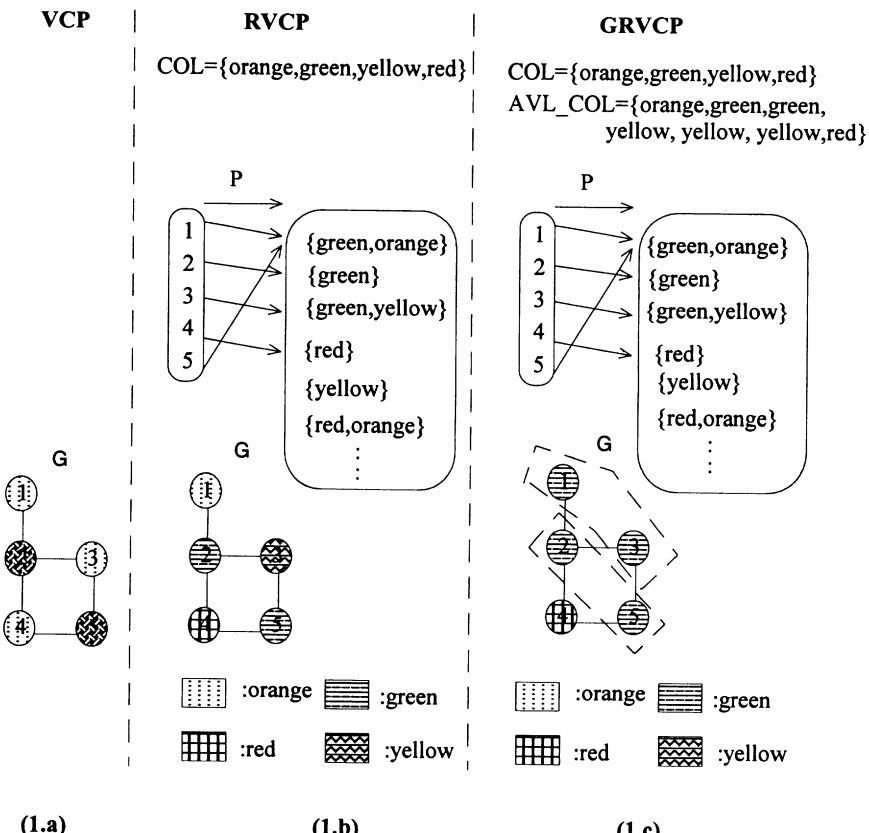


Figure 1: VCP, RVCP, GRVCP.

In a VCP, the vertices of G may be painted in any colour. In this case, G contains a clique of cardinality 2 and the colouring shown in figure 1.a is feasible. Then the colouring is optimal and the chromatic number of G is 2.

In a RVCP there is a set of colours available, and a set of colours allowed for each vertex. If we assume the set $\text{COL}=\{\text{orange, green, yellow, red}\}$ and the application P of the permitted colours of figure 1.b, we have the following situation. Vertices 2 and 4 can only be painted green and red respectively. No other vertex is colourable in red. Only green can be used to colour vertex 5 (besides vertex 2), since vertices 1 and 3 are adjacent to 2. Vertices 1 and 3 have only the colour green in common, already used in the colouring, therefore the colours orange and yellow are required, respectively, for their colouration. Thus the colouring shown in figure 1.b is optimal and the restricted chromatic number of G is 4.

Finally, in a GRCVP, there is a set of colours, of which one or more units of each colour are available. If we assume, besides, a set $\text{AVL_COL}=\{\text{orange, green, green, yellow, yellow, yellow, red}\}$, then as in the previous case, red and green will have to be used to paint vertices 2 and 4, and the same colour unit used for vertex 2 can be also be used to uniquely colour vertex 5. However, in this case two green colour units are available, giving one more green to colour vertices 1 and 3. The colouring shown in figure 1.c is therefore optimal and the generalised restricted chromatic number is 3.

3. Algorithm for the Unbounded_GRVCP

As is usual in the TT approach, the procedure alternates the Mixed and Improving phases until a number of times IM without improving the best solution:

GLOBAL ALGORITHM

```

1.- Initial Solution Construction.
2.- times=0.
3.- while(times<IM).
    3.1.- Improving Phase.
    3.2.- Mixed Phase.
    3.3.- if Improving Phase or Mixed Phase has improved the best
          solution:
              times=0.
    else:
        times=times+1.

```

3.1. The Search Space

A **solution** S of the algorithm presented below consists of a partition of the vertices of G: $S=\{B_1, B_2, \dots, B_n\}$ / $B_i \cap B_j = \emptyset, \forall i,j=1,\dots,n$ and $\bigcup_{i=1,\dots,n} B_i = V$. Each one of the elements of the partition will be called a box.

Each feasible colouring of the Unbounded_GRVCP defines, in a natural way, a solution $S=\{B_1, B_2, \dots, B_n\}$ whose boxes are formed from the vertices that have the same colour unit assigned. On the other hand, if for each box B_i of a solution S there

exists a colour c which can colour all the vertices belonging to B_i , we give it a copy c_i of the colour c . If, furthermore, there are no arcs between the vertices of B_i , then $C:V \rightarrow \text{AVL_COL}$ such that $\forall v_j \in B_i, C(v_j) = c_i$ is a feasible colouring of the Unbounded_GRVCP on G associated with the solution S . Obviously, if any box exists that can be coloured by one of several colours, we can associate S with more than one feasible colouring of the Unbounded_GRVCP on G .

The objective, then, is to find a partition of V with the minimum number of boxes so that the vertices of the box are not adjacent and are colourable by the same colour.

Consider, for example, the graph G and the application of permitted colours P shown in figure 2.

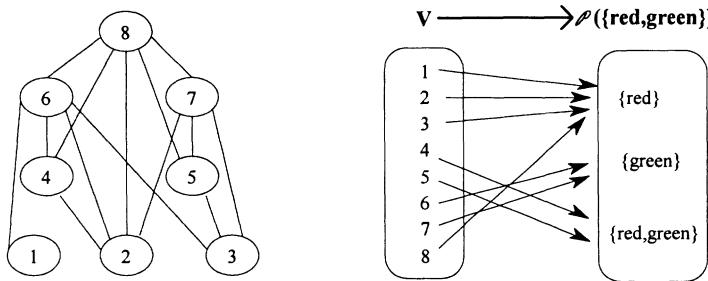


Figure 2: Example of a graph G and a permitted application of colours.

Three solutions are shown in figure 3. The first does not correspond to a feasible colouring of the Unbounded_GRVCP, as there are two adjacent vertices in the third box. The second solution is likewise unfeasible, as there is no colour capable of colouring all the vertices in the first box. However, the third solution is feasible: there are no adjacent vertices in the boxes and there is at least a colour available for colouring all the vertices in each box. Finally the second box can be coloured by a unit of red as well as by a unit of green. Thus there are two feasible colourings of the Unbounded_GRVCP associated with the third solution: {red, red, green, red} and {red, green, green, red} colouring the vertices in boxes 1, 2, 3 and 4 respectively, in both cases. Moreover, these two colourings are optimal colourings of the Unbounded_GRVCP.

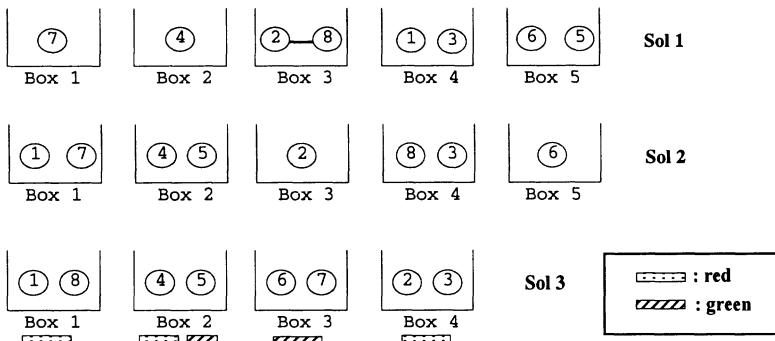


Figure 3: Three solutions for the example in figure 2.

Henceforth, we will say that a solution S is feasible if there exists an associated feasible colouring. The solution will be optimal if the associated colouring is optimal.

3.2. Evaluation of the Solutions

To measure the quality of a solution we use an evaluation function (EVA), which varies according to the best solution found. In this way, the same solution can be evaluated differently during the process of searching for the optimal colouring.

The function EVA is calculated on the **extended graph** \overline{G} , which is obtained by adding arcs to G between pairs of vertices which can not be assigned a single colour.

To help explain the workings of the function EVA, we will introduce a simplified evaluation function f , which does not depend on the previous solutions.

Let $S = \{B_1, B_2, \dots, B_n\}$ be a solution. Then we define:

$$f(S) = \text{NumBoxes}(S) + 2 \sum_{i=1}^n (\text{ARC}(B_i) + \text{NoCol}(B_i))$$

where:

NumBoxes(S) is the number of boxes in S .

ARC(B_i) is the number of arcs in the subgraph of \overline{G} induced by the vertices of the box B_i .

NoCol(B_i) is the minimum number of vertices in B_i which cannot be coloured by a single colour: $\text{NoCol}(B_i) = \min_{c \in \text{COL}} \{v \in B_i / c \notin P(v)\}$. The colours that minimise the expression $\text{NoCol}(B_i)$ constitute the set **Promisings(B_i)**.

Obviously, S is feasible if and only if $f(S) = \text{NumBoxes}(S)$. The feasible colourings associated with S will be obtained by assigning to each box B_i a unit of colour of the set **Promisings(B_i)**.

In the function f two contributions to the sum appearing on the right hand side can be seen. The last provides a measure of the unfeasibility of the solution. As it is

desired that the algorithm searches for a feasible colouring, we have applied a correction factor (the 2 in front of the sum) to give the search for a feasible solution priority over having a smaller number of boxes.

Nevertheless, if the algorithm has already visited a feasible solution, it is convenient to direct the search towards solutions with fewer boxes -better local optima- otherwise the current best feasible solution will not be improved. To achieve this objective we have added one more component to the function f , which penalises solutions which give a higher number of boxes than the best number already achieved. We thus arrive at the definitive evaluation function $EVA(S)$:

$$EVA(S) = \begin{cases} f(S) & \text{if } \text{NumBoxes}(S) < \text{UB} \\ f(S) + (\text{NumBoxes}(S) - \text{UB} + 1) \times \text{Penalty} & \text{if } \text{NumBoxes}(S) \geq \text{UB} \end{cases}$$

where **UB** is the number of boxes in the best feasible solution found so far and **Penalty** is a parameter to be fixed. The use of Penalty reinforces the anticycling mechanism avoiding a return to previous solutions.

We will show how to calculate the evaluation function using the example in figure 2. The extended graph \overline{G} would be obtained by adding all the possible arcs between the vertices of the sets $\{1,2,3,8\}$ and $\{6,7\}$. We assume that we have already obtained a feasible solution with 5 boxes, so the evaluation of the solution of figure 2 would be:

$$EVA(\text{Sol1}) = 5 + 2 \times ((0+0)+(0+0)+(1+0)+(0+0)+(0+0)) + (5-5+1) \times \text{Penalty} = 7 + \text{Penalty}$$

$$EVA(\text{Sol2}) = 5 + 2 \times ((1+1)+(0+0)+(0+0)+(0+0)+(0+0)) + (5-5+1) \times \text{Penalty} = 9 + \text{Penalty}$$

$$EVA(\text{Sol3}) = 4 + 2 \times ((0+0)+(0+0)+(0+0)+(0+0)) + (4-5+1) \times \text{Penalty} = 4.$$

3.3. The Initial Solution

The initial solution will be a single box containing all the vertices.

3.4. The Neighbourhood Structure in the Improving Phase

Let S be the current solution. The **moves** associated with S , **MOVE_SET(S)**, are defined in relation to the vertices of the graph, and consist of moving a vertex from the current box to another box which already exists, or to a new one which is created as a result of the move. Obviously, we do not consider the relocation of a single vertex from one box to a new one to be a move, because it would not change the solution. Likewise, we do not consider introducing a vertex v in a box B which has more than two vertices and which satisfies the condition $P(v) \cap P(w) = \emptyset, \forall w \in B$ to be a move either, because such move would hardly improve the current solution. These restrictions in the definition of the moves do not prevent us from visiting all the solutions.

3.5. The Improving Phase

In each iteration of the improving phase, we do not explore all the moves. Let $S = \{B_1, B_2, \dots, B_n\}$ be the current solution. The set of moves **MOVE_SET(S)** is partitioned in subsets **MOVE_SUBSET(B_i, S)**, $i=1, \dots, n$. The procedure for scanning these subsets is a variant of the Block Order Scan strategy (Glover (1992)).

These subsets are considered as a circular list divided into successive blocks, **Block(j, S)**, with $r(S)$ consecutive boxes. The number $r(S)$ is not a constant, but rather

it varies during the search according to the total number of boxes in the current solution. Figure 4 illustrates the block construction.

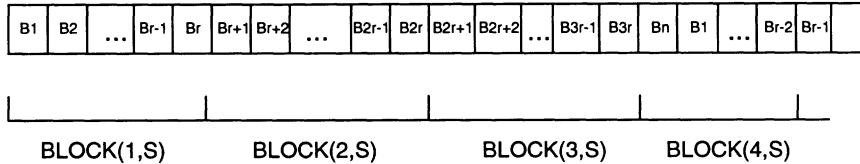


Figure 4: Block Order Scan strategy.

We examine the blocks in order. As a given block $\text{Block}(j,S)$ is encountered we randomly select m vertices from it. (Note that this method of selecting m vertices differs from that defined by Glover). The moves associated with these m vertices define the subset **RANDOM_MOVE_SUBSET(j,S)**.

Then, we evaluate all the moves in $\text{RANDOM_MOVE_SUBSET}(j,S)$ and we select those which return the minimum values of the evaluation function, randomly choosing one of them called **Candidate_Move**. The selected move is executed if it improves the evaluation of the current solution.

The previous process is repeated until a feasible solution is found, or until the list of boxes has been fully visited times_Imp times without improving upon the best solution found in the current improving phase.

The numbers $r(S)$, m and times_Imp are input parameters to the procedure.

If the improving phase finds a feasible solution, we activate a mechanism called **Cluster**. With this mechanism we direct the search to find a new feasible solution with the same colours but using fewer colour units. I.e., we try to organise the same solution in a different way.

The Cluster mechanism builds a new solution in two steps. First it assigns to each box B_j a random colour chosen from $\text{Promisings}(B_j)$. Then, it joins in the same box the vertices from the boxes assigned to the same colour. From this new solution the improving phase begins again.

The improving phase scheme is thus summarised as follows:

IMPROVING PHASE ALGORITHM

```

1.- Initialisation.
     $S = \text{current solution}.$ 
     $\text{times\_list}=0.$ 
    Compute  $r(S)$ .
2.- While( $\text{times\_list}<\text{times\_Imp}$ )
    Let  $BLOCK(j,S)$  be the block formed by the next  $r(S)$  boxes.
        (In the first iteration consider the first  $r(S)$  boxes)
    If  $BLOCK(j,S)$  contains the first box,  $\text{times\_list}=\text{times\_list}+1$ .
    Select randomly  $m$  vertices of  $BLOCK(j,S)$ .
    Evaluate all the moves in  $\text{RANDOM\_MOVE\_SUBSET}(j,S)$ .
    Obtain Candidate_Move.
    If ( $\text{EVA}(\text{Candidate\_Move})<\text{EVA}(S)$ )
        Carry out the move, obtaining a new solution  $S$ .
         $\text{times\_list}=0.$ 
    If  $S$  is feasible:
        Update the best feasible solution, if it is necessary.
        Activate Cluster mechanism.
         $S = \text{solution obtained by Cluster}.$ 
        If  $S$  is feasible, update the best feasible solution, if it is necessary.
    Compute  $r(S)$  again.

```

3.6. The Mixed Phase

The goal of the mixed phase is to drive the algorithm out of the region that it has been exploring during the improving phase, with the hope of locating a local optimum in a different region. We have developed two diversification strategies (**Partial_Div** and **Total_Div**) according to the state of the search.

In the improving phase a move is executed only if it is an improvement. Therefore, the sequence of solutions obtained in this phase define a descending trajectory in the search space. The trajectory terminates at a solution which is the best in its immediate neighbourhood. However, it could be that a small hop to a nearby region would allow the trajectory to continue downwards.

With this in mind, **Partial_Div** randomly selects 25% of the vertices and randomly redistributes them among the boxes. This diversification is only carried out provided that the improving phase has executed at least one movement.

If the improving phase has not made any movements, the descent trajectory is considered to be exhausted: hence it is convenient to provoke a major disruption to ensure significant diversification.

For this purpose, **Total_Div** creates p new boxes, where p is a randomly selected integer between 1 and half the number of boxes in the current solution. Then, a new solution is constructed by randomly distributing all the vertices over the p new boxes.

The diversification strategies proposed in this paper differ from the usual mixed phase in three main respects.

In the first place, the usual mixed phase selects a candidate move by means of the evaluation function, and automatically accepting it. However the diversification strategies proposed in this paper are random diversifications that do not require the evaluation of movements.

Secondly, the algorithm, rather than using a single diversification strategy, activates one of the two defined strategies (Partial_Div or Total_Div) depending on the state of the search.

Finally, in some sense, the number of vertices selected in Partial_Div and the number of boxes in Total_Div play the role of the tabu timing parameter t (Glover (1992)) which determines the number of iterations of the usual mixed phase. However, the calculation of p is influenced by the current solution.

4. Algorithm for the Bounded_GRVCP

The Bounded version of the algorithm presents two variations with regard to the Unbounded version: we have redefined the function f and have introduced a new mechanism of diversification. In other respects the algorithm is identical.

4.1. The Evaluation Function

To establish that a solution is feasible in the unbounded case, as we have no limit on the available colour units, it is sufficient to find a colour unit capable of colouring each box.

However, in the unbounded case, this condition is not sufficient since it could happen that the set of colour units found is not a subset of AVL_COL. The example introduced in the previous section (figures 2 and 3) serves to illustrate this idea. If only two red units are available, then Sol 3 would have only one associated feasible colouring, {red, green, green, red} for the vertices in boxes 1, 2, 3 and 4 respectively.

To measure this new requirement we have added one more term, NumBoxesNoAss(S), to the function f . NumBoxesNoAss(S) takes care of "counting" the number of boxes that, as a minimum, remain uncoloured if each box B_j is assigned a colour from Promisings(B_j).

To calculate NumBoxesNoAss(S), we solve a maximum cardinality matching problem on the bipartite graph $G(S)=(L(S) \cup R(S), A(S))$, where $L(S)$ contains a vertex for each box within the solution S , $R(S)$ contains a vertex for each colour unit of AVL_COL, and the set of arcs $A(S)$ contain an arc (B_j, c_j) if and only if $c_j \in \text{Promisings}(B_j)$. Note that $R(S)$ can be considered finite since it is sufficient to restrict consideration to as many units of each colour as there are boxes in the solution S .

If $MCM(S)$ is the cardinality of the maximum matching associated with S , then,

$$\text{NumBoxesNoAss}(S) = \text{NumBoxes}(S) - MCM(S)$$

and, the new function f is defined as:

$$f(S) = \text{NumBoxes}(S) + 2\text{NumBoxesNoAss}(S) + 3 \sum_{i=1}^n (\text{ARC}(B_i) + \text{NoCol}(B_i))$$

With the new definition of f , we can also state the following result:
S is feasible if and only if $f(S) = \text{NumBoxes}(S)$.

The feasible colourings associated with a feasible solution S are defined by the different maximum cardinality matchings of $G(S)$.

The evaluation function EVA for the Bounded_GRVCP is the same as for the Unbounded_GRVCP, but using the new definition of f .

We will show how to calculate the evaluation function using the example in figure 2. We assume that in the example we have only two red units and three green units and that a feasible solution has already been obtained for 5 boxes. We will calculate the value of the evaluation function for the solution Sol 2 of figure 3.

The set of Promisings associated with the boxes of Sol 2 is:
 $\text{Promisings}(B_1) = \{\text{red, green}\}$, $\text{Promisings}(B_2) = \{\text{red, green}\}$, $\text{Promisings}(B_3) = \{\text{red}\}$, $\text{Promisings}(B_4) = \{\text{red}\}$, $\text{Promisings}(B_5) = \{\text{green}\}$.

Figure 5 shows the graph $G(\text{Sol 2})$ and the maximum cardinality matching with black arcs. Then, $\text{MCM}(S) = 5$ and $\text{NumBoxesNoAss}(S) = 0$. Hence,
 $EVA(\text{Sol2}) = 5 + 2 \times 0 + 3 \times ((1+1)+(0+0)+(0+0)+(0+0)+(0+0)) + (5-5+1) \times \text{Penalty} = 11 + \text{Penalty}$.

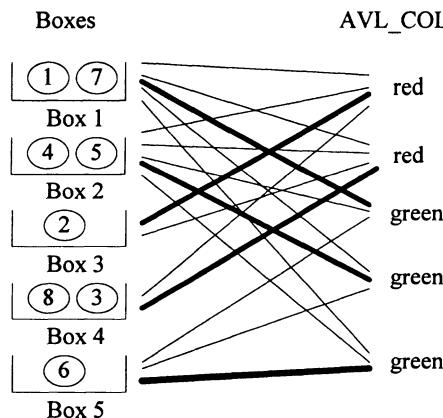


Figure 5: Maximum Cardinality Matching for Sol 2

4.2. The Mixed Phase

The Mixed Phase follows the same scheme as in the previous section, but with a new diversification mechanism (**Join_Div**) which is applied in the following situation.

The improving phase can terminate with a solution S which is feasible for the unbounded problem but unfeasible for the bounded one. This implies that there is at

least one box in the solution that has not been assigned any colour unit in the maximum cardinality matching.

We can consider that the boxes which are assigned colour units by the maximum cardinality matching form the feasible part of the solution and that the others form the unfeasible part. The aim of Join_Div is to restart the active search taking advantage of the feasible part of S.

To accomplish this Join_Div steps through the unmatched boxes B_i and, for each one, randomly selects a colour c from Promisings(B_i). Then it randomly selects a box B_j from the boxes matched to colour c and changes all the vertices in B_i to B_j .

5. Computational Results

In Quintanilla (1994) a Branch and Bound (**B&B**) algorithm is presented to solve the NWMP with a heterogeneous workforce. To the best of our knowledge, no exact algorithm exists to solve either the GRVCP or the RVCP, and since we wish to compare the solutions obtained by the TT algorithm with the optimal solutions, as test problems we have used examples of the NWMP in a heterogeneous workforce formulated as GRVCP problems.

The unbounded problems were generated by using all the possible combinations of the following parameter values. **M** (number of machines) = 5,10,15; **Max_Col** (maximum number of different types of workers) = 5,15; **Max_Act_Mac** (maximum number of jobs to be processed on each machine) = 5,10,15,20 and **Max_Col_Mac_Gnt** (controls the maximum number of types of workers capable of using each machine) = 3,6,10. For each combination 40 duplicates were tested yielding a total of 2880 randomly generated problems. The same problems were used in the bounded case, but with the number of workers of each type chosen randomly between 1 and 5.

Preliminary tests were carried out on a set of randomly generated instances of different sizes, in order to determine the values of the parameters of the TT algorithms. After experimentation the following values produced the best results in terms of solution quality and computational time:

$$\begin{aligned} IM=30, & \quad times_Imp=30, & \quad Penalty=9, & \quad m=5, \\ r(S)=\left\{\begin{array}{ll} \text{NumBoxes}(S) & \text{if } \text{NumBoxes}(S) \leq 3 \\ \max\left(3,\left\lfloor \frac{\text{NumBoxes}(S)}{5} \right\rfloor + 1\right) & \text{if } \text{NumBoxes}(S) > 3 \end{array}\right. \end{aligned}$$

We coded the algorithms in C and ran them on an IBM-compatible PC with a 66Mhz 486DX2 processor under DOS. In some problems the B&B algorithm was not able to guarantee optimality or even obtain a feasible solution because of the limit imposed on the total number of nodes, or because it ran out of memory.

The results are shown in tables 1 and 2, grouped according to the number of machines. The symbols are as follows:

N= Average number of vertices.

NoOpt= Number of problems in which the solution given by the B&B algorithm might not be optimal.

Sol_TT= Average of the colour units used in the solutions given by the TT algorithm.

Gap= Average distance of the solution given by the TT algorithm to the solution given by the B&B algorithm (%).

NF_B&B= Number of problems in which the B&B algorithm did not find a feasible solution.

UF= Number of problems recognised as unfeasible by the B&B algorithm.

NF_TT= Number of problems in which the TT algorithm did not find a feasible solution.

TB&B= Average CPU time used by the B&B algorithm, in seconds (excluding input and output).

TB&B_m, TB&B_M= Minimum (m) and maximum (M) CPU time used by the B&B algorithm, in seconds.

TTT= Average CPU time used by the TT algorithm, in seconds (excluding input and output).

TT_m, TT_M= Minimum and maximum CPU time used by the TT algorithm, in seconds.

Iter= Number of times that "times" is set to 0 in step 3.3 of the global algorithm.

I_m, I_M= Minimum and maximum values of Iter.

As we can see from table 1, our TT algorithm for the unbounded case was able to find a feasible solution in all 2880 of the test problems. The average distance between both the B&B and the TT solutions is very small (0.63%), but there is a big difference between the execution times of the two algorithms. Although for small problems the exact algorithm is better than the heuristic one, as the size of the problem grows the times observed for the TT algorithm are considerably shorter. Moreover, the execution times of the TT algorithm are far more stable (min = 0.94, max = 1763.8) compared to the B&B algorithm (min = 0.01, max = 23647).

Table 1: Computational Results for the Unbounded_GRVCP.

M	N	NoOpt	Sol_TT	Gap	NF_B&B UF	NF_TT	TB&B TB _m -TB _M	TTT TT _m -TT _M	Iter I _m -I _M
5	33.4	0	4.06	0.12	0 0	0	0.6 0.01-4.86	6.33 0.94-31.19	2.01 1-26
10	64.6	20	7.01	0.91	0 0	0	144.11 0.2-1545	37.84 2.42-240.6	3.53 1-32
15	102.1	133	9.96	0.87	0 0	0	1970.71 1.4-23647	153.98 6.7-1763.8	6.2 1-55

Table 2 shows that the bounded version of the TT algorithm was not able to find a feasible solution in 778 test problems, of which 751 were definitely unfeasible, while the B&B was not able to find a feasible solution in 756. The average difference between both the B&B and the TT solutions is also very small in the bounded case (0.64%). The main difference observed was in the execution times, in which the TT algorithm was the most stable, varying between (min = 0.82, max = 2226) while the exact algorithm varied between (min = 0.02, max = 13484).

Table 2: Computational Results for the Bounded_GRVCP.

M	N	NoOpt	Sol_TT	Gap	NF_B&B UF	NF_TT	TB&B TB _m -TB _M	TTT TT _m -TT _M	Iter I _m -I _M
5	33.4	0	4.05	0.04	105 105	109	0.66 0.02-5.13	9.33 0.82-160.5	1.77 1-35
10	64.6	27	6.95	0.55	248 247	256	521.11 0.2-3993	71.34 2.8-645.6	3.82 1-37
15	102.1	106	9.80	1.04	403 399	413	1607.99 1.1-13484	337.6 8.19-2226	6.11 1-51

6. Concluding Remarks

In this paper we have shown a modified Tabu Thresholding algorithm for the generalised restricted vertex colouring problem for the Bounded and Unbounded cases. The basic elements of TT, probabilistic tabu searching and candidate list procedures are supplemented with an evaluation function that depends on the best solution obtained so far, together with a mechanism which reinforces the aggressive search in the improving phase, and new diversification strategies which depend on the state of the search.

The computational study carried out on a collection of test problems with known restricted chromatic number confirms that the behaviour of the algorithm presented is satisfactory. In the unbounded case it has always found a feasible solution, and also for the great majority of bounded problems that have feasible solutions. Moreover, the distance of the solution given by the TT algorithm to the solution given by the B & B algorithm is quite small.

The test problems presented correspond to the formulation of the NWMP as a restricted colouring of vertices because we wish to compare the solution obtained by the TT with the optimal solution; however, numerous practical problems can be formulated in the same way, as indicated in section 1.

Moreover, the algorithm developed for the GRVCP allows variations in the objective function to be considered - such as calculating the set of colour units of minimum cost, extending a partial colouring with the minimum number of colour units or with the minimum cost, etc.- by introducing changes in the evaluation function only.

Acknowledgement

The authors would like to thank the referees for their helpful comments.

References

- D. Brélaz, New Methods to Color the Vertices of a Graph, *Communications of the ACM*, 22 (1979).
- J. Brown, Chromatic Scheduling and the Chromatic Number Problem, *Management Science*, 19 (1972).

- M. Chams, A. Hertz and D. de Werra, Some Experiments with Simulated Annealing for Coloring Graphs, *European Journal of Operational Research*, 32 (1987).
- N. Christofides, *Graph Theory. An Algorithmic Approach*, (Academic Press, 1975).
- S. Chu, Efficient Bounds on a Branch and Bound Algorithm for Graph Colouration, *International Journal of Mathematical Education Science Technology*, 22 (1991).
- D. de Werra, An Introduction to Timetabling, *European Journal of Operational Research*, 19 (1985).
- D. de Werra, Heuristics for Graphs Colorings, *Computing*, 7 (1990) .
- R. Dutton and. R. Brigham, A New Graph Colouring Algorithm, *The Computer Journal*, 24 (1981).
- F. Glover, Future Paths for Integer Programming and Links to Artificial Intelligence, *Computers and Operations Research*, 13 (1986).
- F. Glover, Simple Tabu Thresholding in Optimization, Working Paper, School of Business, University of Colorado, Boulder (1992).
- P.L. Hammer and S. Rudeanu, *Boolean Methods in Operations Research*, (Springer-Verlag, 1968).
- P. Hansen and M. Delattre, Complete-Link Cluster Analysis by Graph Coloring, *Journal of the America Statistical Association*, 73 (1978).
- A. Hertz and D. de Werra, Using Tabu Search Techniques for Graphs Colorings, *Computing*, 39 (1987).
- D. Johnson et al., Optimization by Simulated Annealing: An Experimental Evaluation; Part II, Graph Coloring and number partitioning, *Operations Research*, 39 (1991).
- S. Korman, The Graph Colouring Problem, in: *Combinatorial Optimization*, ed. N. Christofides et al. (John Wiley & Sons, 1979).
- M. Kubale, Some Results Concerning the Complexity of Restricted Colorings of Graphs, *Discrete Applied Mathematics*, 36 (1992).
- M. Kubale and Jackowski, A Generalized Implicit Enumeration Algorithm for Graph Coloring, *Communications of the ACM*, 28 (1985).
- F. Leighton, A Graph Coloring Algorithm for Large Scheduling Problems, *Journal of Research of the Bureau of Standards*, 84 (1979).
- D.W. Matula, G. Marble and J.D. Isaacson, Graph Coloring Algorithms, in: *Graph Theory and Computing*, ed. R. C. Read (Academic Press, New York, 1972).
- N. Metha, The Applications of a Graph Coloring Method to an Examination Scheduling Problem, *Interfaces*, 11 (1981).
- Th. Mohr, Parallel Tabu Search Algorithms for the Graph Coloring Problem, Working Paper, ORWP 88/11, Ecole Polytechnique Fédérale de Lausanne (1988).
- G. Neufeld and J. Tartar, Graph Coloring Conditions for the Existence of Solutions to the Timetable Problem, *Communications of the ACM*, 17 (1974).
- M.S. Quintanilla, *Técnicas Exactas y Heurísticas para la Asignación de una Plantilla de Trabajadores a una Planificación Establecida*, P.h.D.Thesis, Departamento de Estadística e Investigación Operativa, Universidad de Valencia, Spain (1994).
- J. Turner, Almost All k-Coloreable Graphs Are Easy to Color, *Journal of Algorithms*, 9 (1988).
- C.C. Wang, An Algorithm for the Chromatic Number of a Graph, *Journal of A.C.M.*, 21 (1974).

- D.J.A. Welsh and M.B. Powel, An Upper Bound on the Chromatic Number of a Graph and its Applications to Timetabling Problem, *Computer Journal*, 10 (1967).
- D.C. Wood, A Technique for Coloring of a Graph Applicable to Large Scale Timetabling Problems, *Computer Journal*, 12 (1969).
- A. Zykov, On Some Properties of Linear Complexes, *Amer. Math. Soc., Translation*, 79 (1952).

Chunking Applied to Reactive Tabu Search

David L. Woodruff

Graduate School of Management

University of California, Davis

Davis, CA 95616, USA

E-Mail: DLWoodruff@UCDavis.edu

Abstract:

In an earlier paper (Woodruff 1994) we describe the use of, and search for, groupings of solution vector elements that we refer to as *chunks*. The use of chunks has the potential to enhance heuristic search methods such as tabu search and genetic algorithms. In this paper we have two goals: 1) we describe the details of a fully generalized application of chunking to reactive tabu search (Battiti and Tecchiolli 1994) and give some computational results on a particular problem and 2) we use the chunks to explore the nature of the relationship between the search and *cups* (i.e., *domains of attraction* or *chaotic attractors*). We find that chunking allows us to characterize the shape and location of cups in a useful way. We are able to compute the distance from a group of solutions believed to be from one cup to a given solution and test that solution for membership in the cup.

Key Words: Chunking, heuristic search, tabu search.

1. Introduction

In an earlier paper (Woodruff 1994) we describe the use of, and search for, groupings of solution vector elements that we refer to as *chunks*. The use of chunks has the potential to enhance heuristic search methods such as tabu search and genetic algorithms. Human problem solvers often proceed by linking and integrating features they perceive as related and germane to the solution process. When the problem cannot be decomposed, then a common strategy is to form groupings of solution attributes so that the search space can be reduced and higher level relationships can be discovered or exploited. These groupings are what we refer to as chunks.

In this paper we have two goals: 1) we describe the details of a fully generalized application of chunking to reactive tabu search (Battiti and Tecchiolli 1994) and give some computational results on a particular problem and 2) we use the chunks to explore the nature of the relationship between the search and *cups* (Hajek 1988, Ryan 1992) which are also referred to as *domains of attraction* and/or *chaotic attractors*. We find that chunking allows us to characterize the shape and location of cups in a useful way. We are able to compute the distance from a group of solutions believed to be from one cup to a given solution and test that solution for membership in the cup.

Our ultimate goal is to find good solutions to problems generalized by

$$\text{Minimize } f(\mathbf{x}) : \mathbf{x} \in X \subset \Re^n \quad (1)$$

where the function f can be any mapping from X to \Re and the requirement $\mathbf{x} \in X$ summarizes constraints. We assume throughout the paper that n is finite and that the x_i are bounded integers, but our work can be extended to the mixed case where some or all variables can take on real values.

Section 2 gives formal definitions and motivations for chunking and for the general purpose strategy reactive tabu search (RTS). In §3 we review theorems that give some formal motivation for the use of chunking to obtain a metric between a solution and a group of solutions that depends on the attributes of the solution rather than the objective function value. Modifications to RTS to make use of chunking are given in §4 along with the results and insights from computational experiments. The final section offers some conclusions.

2. Definitions

2.1 Reactive Tabu Search

We assume that the reader is at least somewhat familiar with tabu search (Glover 1989). In this section we give a brief introduction to reactive tabu search (Battiti and Tecchiolli 1994).

As with tabu search in general, reactive tabu search is defined relative to one or more neighborhood structures and evaluation functions. For a set $B \subset \Re^n$, $B \supseteq X$ and every solution $\mathbf{x} \in B$, let $\mathcal{N}(\mathbf{x}) \in B$ denote those solutions that can be reached from \mathbf{x} with a single move. A move evaluation function $\hat{f}(\bullet)$, often resembles $f(\bullet)$, but may differ in order to take advantage of special knowledge of the problem. In cases where infeasible solutions are allowed during the search, $B \not\supseteq X$ and \hat{f} is constructed to encourage discovery of feasible solutions.

The search proceeds from trial solution \mathbf{x}^k to a trial solution

$$\mathbf{x}^{k+1} \in \mathcal{A} \subset \mathcal{N}(\mathbf{x}^k)$$

where \mathcal{A} is the set of admissible moves. It is formed by removing from $\mathcal{N}(\mathbf{x}^k)$ any solutions that would reverse the attributes of the most recent κ moves and that do not satisfy an aspiration threshold. The moves that are in $\mathcal{N}(\mathbf{x})$ but not in \mathcal{A} are said to be tabu. The value of κ is changed dynamically during reactive tabu search. It is increased if a repeat solution is detected and decreased slowly over time and/or if all moves are tabu.

The distinguishing feature of RTS is its reliance on long term memory in RTS that is concerned primarily with repeated solutions. If too many solutions (given by the parameter REPT) are repeated too often (given by the parameter CHAOS) then the search is assumed to be trapped by a particular cup. We conjure the metaphor of a chaotic attractor. In addition to recording repeated solutions, the number of moves between repeated solutions is also stored (recorded in the variable Running_Average). This value is assumed to be proportional to the size of the domain of attraction of the chaotic attractor. When REPT solutions have been repeated CHAOS times, the search executes a large number of random moves (with the number proportional to Running_Average) in order to try to escape the domain of attraction of the chaotic attractor.

As a basis of comparison, note that other forms of tabu search likewise make use of frequency-based memory as a foundation for diversification strategies (to drive the search into new regions), but

the frequencies are generally recorded for attributes of solutions, such as values of variables and functions, rather than for complete solutions. In addition, the trigger for diversification in other forms TS typically consists of either an iteration count or a limit on the number of iterations without finding a new best solution.

From the specific focus of RTS, there is a strong relationship between the notion of *cups* as introduced by Hajek (1988) and notion of domains of chaotic attraction. To understand this relationship, consider the directed graph induced by the neighborhood structure N . We construct a directed graph formed by considering solution vectors to be nodes. Arcs are defined by the neighborhood structure. We consider the *search graph* to occupy a two dimensional space and use the evaluation function value associated with each node to create a “surface” in a third dimension “above” the graph. We can then generate the following definitions concerning cups:

1. Let L be the set of local mins less global mins
2. Depth (Hajek 1988): for $\mathbf{x} \in L$ the depth is the minimum for which there is some \mathbf{y} with $f(\mathbf{y}) < f(\mathbf{x})$ that can be reached from x without passing through any z with $f(z) - f(x)$ greater than the depth. (The depth is the min uphill distance to ‘escape’ the local min).
3. Cup (Hajek 1988): set of states each reachable from each other without using a path through a state with an objective value exceeding some constant. Nodes in a cup with min f are at the “bottom” and their depth defines the depth of the cup.
4. Width (Ryan 1994): for $\mathbf{x} \in L$ the width, $W_{\mathbf{x}}$ is the minimum over all paths between \mathbf{x} and all \mathbf{y} such that $f(\mathbf{x}) < f(\mathbf{y})$ of the number of “uphill” moves in the path.

Roughly put, the point of Hayek’s paper was to produce the fastest possible cooling schedule that would still enable a simulated annealing algorithm to escape all cups infinitely often. The use of Hajek’s cooling schedule ensures convergence in probability to only global minima. Fox (1991) points out that a much slower cooling schedule is needed in order to have a finite expected time to see a global minimum. Bearing in mind that random search will, with probability one, see the optimal solution in finite time one must wonder if Charnes and Wolfe (1989) were strong enough in their condemnation of simulated annealing. But we digress.

When designing tabu search algorithms as well as simulated annealing algorithms, one must be concerned with the ability of the search to escape cups that do not contain the global minimum. Simulated annealing mimics the patience of nature by using randomness to eventually take the search out of fully explored cups. What Battiti and Tecchiolli did in conceptualizing RTS was to realize that tabu search, when trapped for too long in a cup, resembles a chaotic process in the presence of an attractor.

There is no known method of characterizing cups in general, although Ryan (1992) has proposed bounds on their depth and width for some specific problems. Hence, any method of escaping cups (including finite annealing) must be ad hoc. Reactive tabu search attempts takes advantage of knowledge about the time between occurrences of repeat solutions to estimate cup size. We suggest carrying that line of thinking further to estimate the location and shape of a cup so that we can estimate the distance of a solution from the cup. This will be discussed further in §4.

2.2 Chunks and Related Definitions

In this section the formal definitions related to chunking are reviewed, for more discussion see Woodruff (1994). Although a human problem solver can make use of very flexible chunking schemes, we restrict ourselves to a more specific definition. Our ideas are related to those of vocabulary building as described by Glover and Laguna (1993). To begin, we define a solution attribute function, $\mathbf{a} : \Re^n \rightarrow \Re^m$ that gives the attributes of a solution. Let M be the index set $\{1, \dots, m\}$.

We will refer to a non-empty subset of attribute vector indexes as a *chunk*. An instance of a partial vector corresponding to a chunk will be referred to as a *chunk instantiating value*. That is, a chunk instantiating value for a given chunk $C \subseteq M$ depends on the specific solution \mathbf{x} whose values are under consideration. The chunk value contained in $\mathbf{a}(\mathbf{x})$ for chunk C will be denoted by $\mathbf{a}(C; \mathbf{x})$. For example, if $C = \{i, j, k\}$, and $i < j < k$, then $\mathbf{a}(C; \mathbf{x}) = (a_i(\mathbf{x}), a_j(\mathbf{x}), a_k(\mathbf{x}))$. In implication of the definition is that $\mathbf{a}(\mathbf{x}) \equiv \mathbf{a}(M; \mathbf{x})$.

The mappings from chunk values to reals will be referred to as *chunk valuation* functions, v . Let $\mathbf{v}(\mathbf{a})$ be the vector of chunk valuations for a given attribute vector, \mathbf{a} . Of course, we will often be interested in the valuation vector for a particular chunk, C , of an attribute vector, \mathbf{a} of a solution \mathbf{x} which we write as $\mathbf{v}(\mathbf{a}(C, \mathbf{x}))$.

For the remainder of this paper, we are primarily concerned with collections of chunk sets. Let \mathcal{P} constitute the set of p chunks. We omit the subscript a from \mathcal{P} and omit parameter subscripts for the remainder of the section. An interesting special case is a partition of the indexes of the vector valued attribute function \mathbf{a} .

One of the most important ways in which we can exploit chunking is to switch from the usual search-graph-based orientation to a spatial orientation. We take a set of solution vectors and find their location and shape. This characterization then allows us to quantify the distance between a solution and the set whose location and shape have been determined. We shall describe applications to tabu search; however, another obvious application is to genetic algorithms and attempts to verify that a new solution will diversify an existing population.

In the case where solution vectors have elements that are either zero or one, the Hamming distance (the number of vector elements that differ) can be used to compute similarities between pairs of solutions. However, we are interesting in computing the distance of a solution from a set of solutions and we would like to be able to do it for permutation vectors, general integer vectors and $\{0, 1\}$ vectors.

To get the appropriate metric, we make use of a location vector (mean) $\boldsymbol{\mu}$ of length p and a $p \times p$ shape matrix (covariance matrix) \mathbf{S} . A p -vector \mathbf{v} can be said to have a *Mahalanobis distance* from $\boldsymbol{\mu}$ using \mathbf{S} that is given by

$$d(\boldsymbol{\mu}, \mathbf{S}; \mathbf{v}) = (\mathbf{v} - \boldsymbol{\mu})^T \mathbf{S}^{-1} (\mathbf{v} - \boldsymbol{\mu}) \quad (2)$$

Short distances are consistent with $\boldsymbol{\mu}$ and \mathbf{S} and longer distances are not as consistent.

3. A Partial Review of the Theory

The motivation for this metric just given comes from multi-variate probability theory. If \mathbf{S} and $\boldsymbol{\mu}$ define a multi-variate normal distribution, then for points governed by the distribution, the Mahalanobis distances are distributed χ^2 (see e.g., Anderson 1984 pages 72-75). Using an estimate of the shape and location of a population, we will infer that points with lower distances are more likely to have come from the population than those with larger distances. The following theorem gives one set of sufficient conditions for this to hold exactly. Others exist as well.

Theorem 1 (Woodruff 1994) Suppose we have two populations, Ω_1 governed by a multivariate normal distribution with mean μ_1 and covariance matrix \mathbf{S} and Ω_2 from a multi-variate normal (μ_2, \mathbf{S}_2) with smallest eigenvalue of \mathbf{S}_2 greater than the largest eigenvalue of \mathbf{S}_1 , then there is a finite distance, D , such that all \mathbf{x} satisfying $d(\mu_1, \mathbf{S}_1; \mathbf{x}) > D$ are classified as being from Ω_2 rather than Ω_1 according to maximum likelihood.

In many heuristic search applications, we are interested in distinguishing between a “population of interest” and other solutions. Consider these populations to be Ω_1 and Ω_2 respectively. In that light, the assumption concerning the eigenvalues is not as extreme as it might first appear. If the vector elements happen to be independent, then the assumption is equivalent to stating the all the variances in Ω_2 are greater than the variances in Ω_1 .

The use of chunks also reduces the dimension. For humans this has the benefit of making things a bit more tractable. For our work, reduced dimension greatly increases the chances that we will be able to estimate \mathbf{S} and μ and to identify outliers (Woodruff and Rocke 1994). The identification of multi-variate outliers is closely related to the process of verifying that a solution is different from a group of solutions (e.g., the solution is diversifying).

Both chunking and the use of a summation based valuation function are needed to make this a plausible scheme. Define a *summation based* valuation function as being of the form

$$v(\mathbf{a}(C; \mathbf{x})) = K \sum_{i \in C} a_i(\mathbf{x})$$

where K is constant for a given chunk size. Other classes of functionals might be used, but summation (such as the average attribute value, total, or sum of squares) often make sense to a human user and offer two very important, additional benefits.

First, summation based valuation functions offer a form of continuity. Two solution vectors that are very similar will have similar valuation vectors. Hence, if we want to require that we get two very different solution vectors, it is enough to require that we get two very different valuation vectors. This is useful in part because the valuation vectors can be much shorter than the solution vectors, but a more interesting reason is the second. We can obtain a distance metric. It is common practice to make use of summations to make the normal distribution plausible for individual vector elements, as we suggest in the next theorem.

Theorem 2 (Woodruff 1994) *If we make use of partitions \mathcal{P} each with a fixed number of cells, p that divide the m indexes of an attribute function, \mathbf{a} so that the variables within each cell, \mathcal{P}_i , $1 \leq i \leq p$, have some randomness and are weakly dependent (e.g., stationary and α -mixing with $\alpha_n = O(n^{-5})$) then summation based valuation functions $v(\mathcal{P}_i; \mathbf{a})$, will have distribution that becomes normal as n is increased without bound.*

Theorem 1 requires more than normally distributed vector elements with arbitrary covariance; it requires multi-variate normal vectors (i.e., and elliptical distribution). In order to obtain multi-normal vectors, we could use groupings as valuation vectors in a fashion reminiscent of techniques used in quality control . For simplicity of notation and because it seems to work well in practice, we won't group them in this paper. In other words, we will hope that the summation based valuation functions result in convergence that occurs *very* quickly.

4. Chunking Applications in RTS

4.1 Modifications to RTS

One of the keys to the success of RTS is escape from “chaotic attractors.” In our implementation, this is accomplished by making a random number of random moves where the random number is uniform on the interval (1, Running Average). This was found to work well on the instances we tested. Of course one could tune this further, but by using chunking one obviates the need. We have created a general purpose modification to RTS that uses summation based (i.e., average) valuation functions to verify that the escape is likely to have been successful.

Algorithm H is described by Woodruff (1994). It hashes the valuation vector during search and records the hash values; random escape is not terminated until a valuation vector is encountered that has not yet been seen. Algorithm H' samples the valuation functions with frequency $n/4$ to create a population estimate for the location, μ , and the shape (covariance), S , of the current chaotic attractor. The random escapes are repeated until the result is a vector whose valuation vector has a distance that is at least $2p$ (which is about $(\chi_p^2)^{-1}(0.95)$ for the values of p used in our experiments).

4.2 Computational Experiments

Our experiments make use of the medium sized newspaper distribution and production problem (MNDP) introduced by Hurter and

Van Buer (1993). There are natural user specified chunks and chunk valuation functions.

We consider a newspaper with multiple products (e.g., city edition, far western suburbs, etc.) with a printing press setup required between products. Trucks carry the newspapers to drop off points. We assume that there is no meaningful limit on the number of trucks that can be available at loading docks and that the time to load a truck is negligible. There is a constraint on the starting time for production to begin and the final allowable time for delivery to drop off points. The objective is primarily to minimize the number of trucks and secondarily to minimize the distance travelled.

We are given the drop off coordinates as (x,y) pairs with the press facility at $(0,0)$. For simplicity, and WLOG, we assume that some norm can be used for both time and distance together. For each drop off point, i , we are given the demand for newspapers and the product type. We are given a constraint on the overall last delivery time. The setup time to switch from product i to product j is also given. The objective function contains the cost for each truck used and for each distance unit travelled.

Although a standard mixed integer formulation can be given Hurter and Van Buer (1993), we prefer a single vector solution for heuristic search optimization. Solutions can be represented by a single vector, σ , which gives the order of the drop off points. We can compute the objective function value for a solution vector by assigning drop off points in sequence to a truck until either the truck is full (the next drop off point will result in exceeding the capacity) or the next drop off point would violate the time constraint. Computation of the time of delivery for drop off point $\sigma(i)$ is simply the sum of the production times for all drop off points up to an including i (including setup time) plus the total travel time for the truck to get to $\sigma(i)$. We may wish to maintain additional data structures to speed up the computation of the objective function value, but the solution is fully represented by the single vector σ . For problems with permutation vector solutions (such as the TSP and the MNDP) it proves useful to employ a one-to-one map for each solution vector σ to a chunk value vector a where each element in a corresponds to a drop off point and gives the position in σ for it.

There are a number of ways one might search for good chunks to use. In these experiments, we settled for naive chunks. We partitioned the attribute vectors into p chunks using a crude sorting of

the dropoff locations (primary sort by x axis coordinate, secondary by y followed by division into chunks of equal size). The average attribute evaluation function was used as an instance of a summation based function:

$$v(\mathbf{a}(C; \mathbf{x})) = \frac{1}{|C|} \sum_{i \in C} a_i(\mathbf{x})$$

where $|C|$ is the number of indexes in the chunk. To illustrate, suppose we have a solution vector $\sigma = (4, 3, 5, 2, 1)$ so $\mathbf{a}(\sigma) = (5, 4, 2, 1, 3)$. If there is a chunk $C = (1, 3)$ then $v(\mathbf{a}(C; \sigma)) = \frac{1}{2}(5 + 2)$. Dropoff points one and three occur fairly late in the sequence, so this value is greater than it would be if they occurred earlier.

Our computational experiments consisted of ten searches each beginning from a randomly generated starting solution. Since the chunks were fairly arbitrary, we tried the experiments with the number of chunks (p) varying between three and ten and with a search of 10,000 iterations. With three chunks, algorithm H' had some trouble due to singular covariance matrices, but otherwise the performance was not sensitive (in a statistically significant way) to the number of chunks or the choice between algorithm H and H'. The maximum improvement over RTS without chunking was 20% and the average was 5%. Of the ten replicates, there was no significant difference in three and the use of chunking produced improvement in seven.

These experiments confirm the value of chunking demonstrated by Woodruff (1994) and illustrate the use of distance based measures that were not demonstrated in that paper. Perhaps more interesting is the use of distances to characterize the search in and near cups (or chaotic attractors to use the metaphor of Battiti and Tecchiolli). That is the subject of the next section.

4.3 The Location and Shape of a Chaotic Attractor

In this section we are interested in simultaneously examining two things: 1) the notion of chaotic attractors (or cups) and the idea that RTS facilitates escape from them and 2) the notion of sampling the valuation vectors in a cup as a means of characterizing it by its shape and location (i.e., moving to a spatial orientation). To understand how we will do this, we must consider the details of the operation of RTS during our experiments.

Depending on the replicate, RTS found it necessary to escape either three or four times during the course of the 10,000 moves

it was allowed to make. Hence, we would hope that when we are using chunking to check the success of the escape, we would find that the search usually escapes to a new cup. To put it another way, we would hope that as we sample the chunk valuation vectors they seem to come from a single multi-variate normal population until we escape. After the escape, we would hope that they seem to come from a different population. Based on the assumption that the search topography is “rough” (i.e., there are many cups), we would even hope that the third “cup” is clearly different than, for example, the first most of the time.

We might prefer to test the assumptions of the theorems in support of chunking directly. Unfortunately, these tests are not likely to be productive. It is well known that the use of summation based valuation functions is almost certain to produce valuation vector elements that are roughly normal so this test will not tell us much that we do not already know. It would be nice to test for multi-variate normality, but good tests are essentially non-existent (Anderson 1984).

This leaves us in a potentially uncomfortable position. If we fail to see the behavior that we expect, we will not be sure if it is due to a shortcoming in our implementation of RTS or chunking (or perhaps an idiosyncratic property of the problem). Happily, we did get the behavior that we expected. To ease exposition, we define $D(\alpha, \beta)$ to be the set of distances from the location of the valuation functions for points sampled in cup α to the points sampled from cup β using the sample covariance matrix from the α sample in Equation 2. In other words, it is the set of distances

$$d(\boldsymbol{\mu}_\alpha, \mathbf{S}_\alpha; \mathbf{v}_i)$$

where $\boldsymbol{\mu}_\alpha$ and \mathbf{S}_α are the sample mean and covariance from the α cup valuation function samples and the \mathbf{v}_i are the β cup valuation function samples.

First we examine the replicate where chunking provided the greatest benefit. Figure 1 plots order statistics for $D(1, 1)$. The dashed line shows an approximation for the order statistics under the assumption of multi-variate normality which is $(\chi^2)^{-1}(i/(\eta+1))$ where η is the number of samples (see David 1981 for more information or better approximations). Although it is not a perfect fit, one certainly gets the impression that these distances approximate those from a sample of a cohesive multi-variate normal data. This is especially true when one examines the next figure.

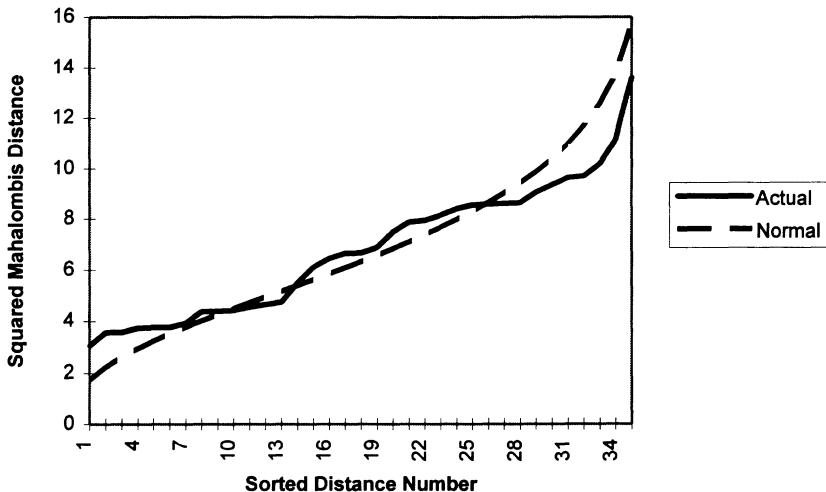


Figure 1: Order Statistics for $D(1,1)$

Figure 2 shows $D(1, 2)$, $D(2, 3)$ and $D(1, 4)$. This figure is typical of the sort of behavior that we encountered during all ten replicates. The sets $D(x, y)$ where x and y differ by one (in other words, distances between cups “next to each other” or “encountered in sequence”) usually contains some moderately low distances along with some large distances. The sets $D(x, y)$ with x and y differing by two or more typically contain only large distances. In other words, multiple escapes almost always achieve a new cup, but one escape is more interesting.

For both $D(1, 2)$ and $D(2, 3)$, the distances indicate that the search, although forced away from the previous “cup”, seems to return to solutions that have valuation vectors that are close to the valuation vectors for previous cup. However, the random escape moves have created enough changes that — coupled with tabu navigation induced by the tabu list — cause the search eventually to leave the “cup” (chaotic attractor) in convincing fashion.

Notice that these plots give us information that is radically different from traces of the objective function because they are based on the location and covariance structure of the solutions visited. In other words, there is no reason to believe that two solutions with similar objective functions are near or far in the search graph, while large distances based on chunking methods tend to reduce the

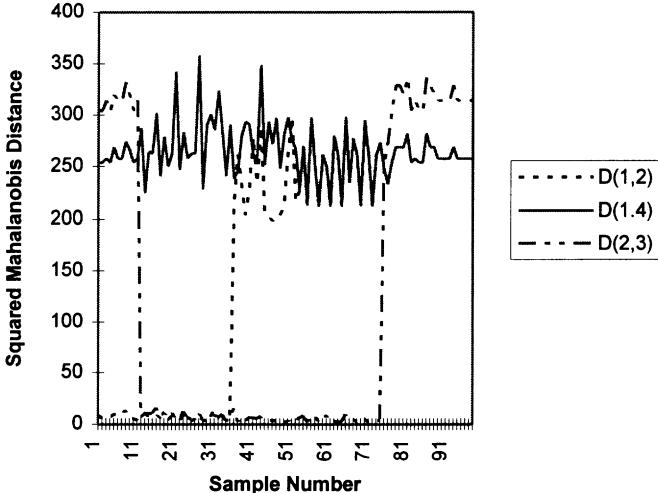


Figure 2: Search Trajectories Expressed as Distances

likelihood that the solution comes from the same region that was sampled to produce the metric parameters.

5. Conclusions

We have reviewed the theoretical foundations for chunking and the metaphor with human thought processes that motivate its use. We have also extended the computational results given by Woodruff (1994). It is not our goal in these experiments to convince the reader that chunking must be an integral part of every successful heuristic search algorithm. Rather, we have the less dramatic, but more plausible goal of demonstrating that it can be effective.

In this paper we also demonstrated that the use of chunking can yield interesting detailed information about the performance of a meta-strategy. In this case, it was reactive tabu search. Better understanding of the search enables improvements in the strategy either by re-design or by dynamically using information gleaned via the use of chunking during the search. This understanding was the result of calculating distances. In the original space of solution vectors there may be no known, useful way to determine the distance between a vector and a group of vectors. We can exploit chunking to obtain a metric.

The idea of a metric for diversification has an important place in

TS research. For example, it is proposed by Glover (1990) that distance be defined relative to notions of influence, and that thresholds be used to define *distance classes* as a basis for determining conditions under which moves in different classes should be regarded as attractive. Our chunking ideas give a precise way to characterize such classes and a theoretically supported method of determining the thresholds. As also noted Glover (1990), the learning approach of target analysis can be applied to generate information to exploit trade-offs based on distance for specific problem classes, and we observe that such a learning approach can also be used to identify rules for finding good chunks.

Chunking gives us the ability to form groupings of solution attributes so that the dimension can be reduced and higher level relationships can be discovered or exploited. The methods are easy to implement, have a good theoretical foundation and their application is limited only by the creativity of heuristic search researchers. This limit is probably not an active constraint.

Acknowledgments

The research reported in this paper was supported by grants from the National Science Foundation (DMS-93.01344, DMS-94.06193). This work has benefited greatly from the comments of Fred Glover.

6. References

- T.W. Anderson, *An Introduction to Multivariate Statistical Analysis*, John Wiley and Sons, New York, 1984.
- R. Battiti and G. Tecchiolli, "The Reactive Tabu Search," *ORSA Journal on Computing*, **6** (1994), 126-140.
- A. Charnes and M. Wolfe, "Extended Pincus Theorems and Convergence of Simulated Annealing," *Int. J. Systems Sci.*, **20** (1989) (1521-1533).
- H.A. David, *Order Statistics*, John Wiley and Sons, New York, 1981.
- A.J. Duncan, *Quality Control and Industrial Statistics*, Irwin, Homewood, IL, 1986.
- B.L. Fox, "Eliminating a Pathology of Simulated Annealing," working paper, University of Colorado, Denver (1991).

- F. Glover, "Tabu Search - Part I", *ORSA Journal on Computing*, **1** (1989), 190-206.
- F. Glover, "Tabu Search: A Tutorial," *Interfaces*, **20** (1990) 74-94.
- F. Glover and M. Laguna, "Tabu Search," in *Modern Heuristics for Combinatorial Problems*, C. Reeves, ed., Blackwell Scientific Publishing, 1993.
- B. Hajek (1988) "Cooling Schedules for Optimal Annealing," *Mathematics of Operations Research*, **13**, 311-329.
- A.P. Hurter and M.G. Van Buer, "The Newspaper Production and Distribution Problem: Medium Sized Newspapers," Working Paper, IE/MS, Northwestern University, 1993.
- J. Ryan, "The Depth and Width of Local Optima," Technical Report, Department of Mathematics, University of Colorado, Denver, (1992).
- M.M. Solomon, "Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints," *OR* **35** (1987) 254-265.
- D.L. Woodruff and Rocke, D. M. "Computable robust estimation of multivariate location and shape in high dimension using compound estimators," *Journal of the American Statistical Association*, **89** (1994), 888-896.
- D.L. Woodruff "Proposals for Chunking and Tabu Search," Technical Report, GSM, UC Davis, Davis CA 95616 (1994).

Tabu Search on the Geometric Traveling Salesman Problem

Martin Zachariasen and Martin Dam

Department of Computer Science (DIKU)

University of Copenhagen

Universitetsparken 1

DK-2100 Copenhagen East, Denmark

E-mail: martinz@diku.dk

Abstract:

This paper presents a new tabu search approach for the geometric Traveling Salesman Problem. The use of complex TSP transitions in a tabu search context is investigated; among these transitions are the classical Lin-Kernighan transition and a new transition, called the Flower transition. The neighbourhood of the complex transitions is reduced strategically by using computational geometry forming a so-called variable candidate set of neighbouring solutions, the average quality of which controlled by a parameter. A new diversification method based on a notion of solution-distance is used. The experimental results are comparable to the best published results in the literature.

Key Words: Combinatorial optimization, heuristics, local search, tabu search, traveling salesman problem

1. Introduction

The Traveling Salesman Problem (TSP) is probably the best known *NP*-hard problem of its genre: combinatorial optimization. Several approximation algorithms and heuristics exist for obtaining near-optimal solutions to the problem and in the last few years experiments with sophisticated heuristics have been successful, especially

for medium- to large-scale problem instances (+1000 nodes) [Johnson 1990, Martin et al. 1992, Fiechter 1994]. Most of these heuristics are based on the concept of local search, i.e. continuously to *transition* from one solution to a neighbouring solution, using a neighbourhood $\mathcal{N}(T)$ for every solution T .

The geometric TSP is stated as a set of nodes V in a metric space (e.g. the plane) and a distance function (metric) $d(t', t'') = |(t', t'')|$, $t', t'' \in V$ (e.g. the Euclidean distance in the plane). Let $D_r(t) = \{t' \in V | d(t, t') \leq r\}$ be the set of neighbours to t at most distance r from t . This set can be found efficiently by using geometric data structures [Bentley 1992].

A TSP-transition works by exchanging a number of edges (x_i , $i = 1..k$, $k \geq 2$) in the current tour with edges (y_i , $i = 1..k$) not in the current tour, while maintaining the tour-property. The edges x_i are called the *deleted* edges, and the edges y_i are the *added* edges. We assume that a pair of edges x_i, y_i always has one node in common and use the following notation: $x_i = (t_i^-, t_i)$ and $y_i = (t_i, t_i^+)$.

The sequence of deleted and added edges makes up an alternating cycle for the simplest transitions 2-Change and 3-Change, i.e. for 2-Change we have $t_2^- = t_1^+$ and $t_1^- = t_2^+$, and for 3-Change $t_2^- = t_1^+$, $t_3^- = t_2^+$ and $t_1^- = t_3^+$. In this case we denote by $t_i^\#$ the node $t_i^- (= t_{i-1}^+)$ ¹, which can be considered as the predecessor *or* successor of t_i in the tour (Figure 1).

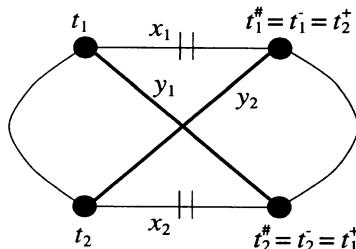


Figure 1: Node and edge labeling (2-Change).

The *gain* of a transition is the decrease in tour length. The gain obtained by each pairwise exchange of edges is $g_i = |x_i| - |y_i|$ and the cumulative gain is $G_0 = 0$, $G_i = G_{i-1} + g_i = \sum_{j=1}^i g_j$, $i \geq 1$. The total gain or decrease in tour length for a transition is then G_k .

¹With the exception that $t_1^\# = t_1^- = t_k^+$.

Experience has shown that using simple transitions (e.g. 2-Change or 3-Change) limits the obtainable performance - no matter how smart the local search procedure is. The problem is the small neighbourhood of each solution, since this determines the distribution of local minima, i.e. solutions that cannot be improved by making a single transition. The average quality of local minima for simple transitions is bad, and relatively, only very few local minima are close to the optimum solution.

This paper evaluates how the local search variant *tabu search* [Glover 1989, Glover 1993] can benefit from using complex TSP-transitions. Previous applications of tabu search to the TSP have only considered the 2-Change as the basic transition [Malek et al. 1989, Fiechter 1994]. The effectiveness of using stronger transitions, such as the 3-Change transition and the transition used in the famous Lin-Kernighan heuristic (L-K) [Lin and Kernighan 1973] is addressed in this paper.

The application of the L-K transition to tabu search is not trivial, since the neighbourhood generation procedure is highly biased towards generating improving transitions. As the performance of tabu search depends on the ability to generate non-improving transitions, a relaxation of the transition generation constraints is therefore needed (section 2).

In section 3 a new and strong TSP-transition, similar to the L-K transition, but with some distinct advantages - especially with relation to tabu search - is introduced. Section 4 summarizes our tabu search approach, which is independent of the transition type used, that is, all search parameters are the same for every type of transition. The computational results, including a comparison with other similar local search methods, are given in section 5. Because of its applicability and popularity, the 2-dimensional Euclidean TSP is used as a test case, but the algorithms can be generalized to any geometric TSP.

2. A variable candidate set for complex TSP-transitions

Incorporating complex transitions into a local search procedure obviously improves the solution quality obtained. The cost is a highly increased time-consumption evaluating the neighbourhoods. In order to make these transitions usable in practice a controlled reduction of the neighbourhood into a *candidate set* $\mathcal{V} \subseteq \mathcal{N}(T)$ is

required. The solutions in this candidate set should be biased towards the best solutions in the neighbourhood as to minimize the degradation of the performance.

Therefore we introduce a so-called *variable candidate set* $\mathcal{V}(\delta)$, where the average quality of the solutions in $\mathcal{V}(\delta)$ is controlled by a continuous parameter $\delta \geq 1$. When δ is increased the candidate set is enlarged and the average quality deteriorates².

Before we describe our variable candidate set more thoroughly, we first present some bare-bone algorithms for generating 3-Changes and L-K transitions, leaving out the less interesting technical details. All *improving* 3-Changes can be found using this algorithm [Bentley 1992]:

1. Select $t_1 \in V$, giving x_1 .
2. Let $t_2^\# \in D_{|x_1|}(t_1)$, giving y_1 and x_2 .
3. Let $t_3^\# \in D_{G_1+|x_2|}(t_2)$, giving y_2 , x_3 and y_3 .

This algorithm should be read as a sequence of nested loops, where every variable t_1 , $t_1^\#$, etc. runs through all possible values. We let the transitions generated at step 3 be the members of $\mathcal{V}(1)$; this set is guaranteed to include all improving transitions.

The algorithm makes heavy use of $D_r(t)$ node-neighbourhoods, which again rely on the geometric properties of the problem. Note the use of the cumulative gain G_1 showing the close relationship between the 3-Change algorithm and the iterative L-K transition generation algorithm. The selection procedure at level $i \geq 1$ for the L-K transition is (recall that $t_1 \in V$, giving x_1):

1. Given: x_1, x_2, \dots, x_i and y_1, y_2, \dots, y_{i-1} , $G_{i-1} > 0$.
2. Select $t_{i+1}^\# \in D_{G_{i-1}+|x_i|}(t_i)$, giving y_i, x_{i+1} (in order to satisfy gain criterion $G_i > 0$).
3. Set $G^* = \max(G^*, G_i + g_{i+1}^*)$ (best close-up value) and check stopping rule $G_i \leq G^*$. Note that $g_{i+1}^* = |x_{i+1}| - |y_{i+1}^*|$ is the close-up gain obtained by closing up with the edge y_{i+1}^* .

²It should be noted, that $\mathcal{N}(T)$ and $\mathcal{V}(\delta)$ are *solution* sets, but in the following we will often refer to them as *transition* sets. The meaning should be clear, and this view is also closer to the actual implementation.

This algorithm is considered as a transition *generation* procedure, i.e. the members of $\mathcal{V}(1)$ are all transitions that are evaluated (by computing their close-up values).

The $\mathcal{V}(1)$ set is ideal in an aggressive mode of search, but its application makes it hard to perform diversifying transitions, i.e. transitions, which break away from the current solution structure. We have therefore extended this candidate set by multiplying a parameter Δ (related to δ) to the radii of the node-neighbourhoods, as shown below.

The extension fulfills the condition $\mathcal{V}(1) \subseteq \mathcal{V}(\delta_1) \subseteq \mathcal{V}(\delta_2) \subseteq \mathcal{V}(\infty) = \mathcal{N}(T)$, $1 \leq \delta_1 \leq \delta_2$, and for small values of K and δ we have $|\mathcal{V}(K\delta)| \approx K|\mathcal{V}(\delta)|$. To fulfill this last condition, Δ has to be related to δ in a transition type dependent way. Because of the geometric properties a general (approximative) relation is $\Delta = \sqrt[Q]{\delta}$, $Q \geq 1$, where Q depends on the transition type; a suitable value is found experimentally. The square-root relationship originates in the functional relationship between the average number of nodes in a circle and its radius.

The original gain definition in section 1 is used to generate the basic candidate set $\mathcal{V}(1)$. We define a *relaxed gain* as $\hat{g}_i = \Delta|x_i| - |y_i|$, which reflects the multiplication of the radii. A relaxed cumulative gain \hat{G}_i is recursively defined as

$$\hat{G}_0 = 0, \quad \hat{G}_i = \Delta(\hat{G}_{i-1} + |x_i|) - |y_i| = \Delta\hat{G}_{i-1} + \hat{g}_i, \quad i \geq 1$$

Observe that the relaxed gain definition is identical to the original gain definition when $\Delta = 1$. Also constraints based on this gain become less restrictive (i.e. the gain is larger) when Δ is increased. One main feature of $\mathcal{V}(\delta)$ is that it covers the whole neighbourhood when δ (and thus Δ) becomes large, i.e. $\mathcal{V}(\infty) = \mathcal{N}(T)$.

By using this relaxed gain definition, the 3-Change algorithm for generating the $\mathcal{V}(\delta)$ set becomes:

1. Select $t_1 \in V$, giving x_1 .
2. Let $t_2^\# \in D_{\Delta|x_1|}(t_1)$, giving y_1 and x_2 .
3. Let $t_3^\# \in D_{\Delta(\hat{G}_1 + |x_2|)}(t_2)$, giving y_2 , x_3 and y_3 .

We have $\hat{G}_1 > 0$, so the radius of the second query is always positive; also note that $\hat{G}_2 > 0$. By increasing Δ sufficiently, all possible 3-Changes will be covered.

Similarly the L-K algorithm is:

1. Given: x_1, x_2, \dots, x_i and $y_1, y_2, \dots, y_{i-1}, \hat{G}_{i-1} > 0$.
2. Select $t_{i+1}^\# \in D_{\Delta(\hat{G}_{i-1} + |x_i|)}(t_i)$, giving y_i, x_{i+1} (in order to satisfy relaxed gain criterion $\hat{G}_i > 0$).
3. Set $G^* = \max(G^*, G_i + g_{i+1}^*)$ (best close-up value) and check relaxed stopping rule $\hat{G}_i \leq G^*$.

Note that G^* holds the real (or original) gain - otherwise the stopping rule would be much too permissible. The stopping rule above is already less restrictive than the original, since we always have $\hat{G}_i \geq G_i$.

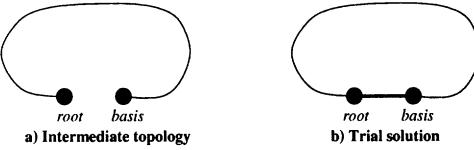
This ends our variable candidate set description for the 3-Change and L-K transition. In the next section a variable candidate set is similarly defined for a new complex transition and in section 4 its use in a tabu search context is discussed.

3. The Flower transition

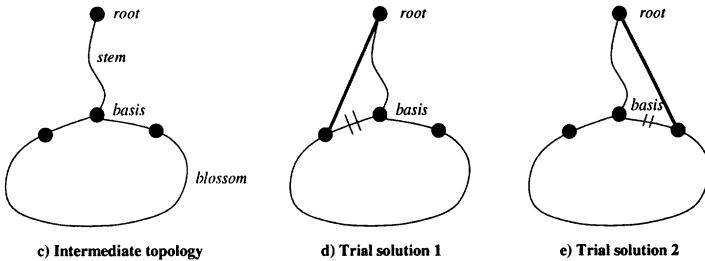
We now present a new transition, which is based on ideas from [Glover 1992]. This so-called *Flower* transition is similar to the L-K transition, but generally produces a larger neighbourhood by relaxing the restrictions upon which edges can be deleted or added.

The *initial step* in the L-K transition generation procedure is the deletion of a tour edge x_1 . This deletion transforms the tour into the *intermediate topology*, a tour path (Figure 2a), which is preserved during subsequent add/delete steps. The *root* is the vertex from which a new edge is to be added and the *basis* is the vertex from which the first add/delete step was made. The intermediate topology is used to obtain *trial solutions* (by adding the edge from root to basis) in order to decide, whether more add/delete steps should be made or the best transition generated should be applied (Figure 2b).

When an edge $y_i = (t_i, t_{i+1}^\#)$ is added from the root only one tour path edge is a candidate for deletion, namely the one on the



The Lin-Kernighan transition



The Flower transition

Figure 2: Intermediate topologies.

path from $t_{i+1}^\#$ to the root - the other edge does not preserve the intermediate topology.

Instead of a tour path, the *Flower* transition intermediate topology is a cycle (the blossom) with an attached path (the stem). This topology produces two trial solutions at every step (Figure 2c,d,e).

The edge selection strategy (or rules for leading from one intermediate topology to another) is similar to the L-K transition selection strategy. Unlike the L-K transition, no edge is initially deleted to transform the tour into the intermediate topology (the flower), because the tour itself is a special flower with zero length stem. But one of the edges $x'_1 = (t'_1, t_1)$ and $x''_1 = (t_1, t''_1)$, incident to the basis t_1 , will finally be deleted (Figure 2d,e); the problem is that initially we do not know which one. If we let x_1 be the longer of the two incident edges x'_1 and x''_1 , and define the initial gain as usual to $g_1 = |x_1| - |y_1|$, we are optimistic, because we expect the longer of the two edges to be deleted.

In the subsequent steps, though, we do know which edge is deleted, so the usual gain definition can be used at all levels. The candidates for $t_{i+1}^\#$ must be in $D_{G_{i-1} + |x_i|}(t_i)$ in order to satisfy the

gain criterion.

When a candidate for $t_{i+1}^{\#}$ is found, it must be determined whether it is on the blossom or on the stem (Figure 3). Depending on whether the blossom or stem is hit, we transform the current intermediate topology into a new one by applying one of these two rules (Figure 3):

- **Rule 1** Two candidates for t_{i+1} are considered and the one that maximizes $|(t_{i+1}^{\#}, t_{i+1})|$ is selected.
- **Rule 2** Only one candidate for t_{i+1} is applicable.

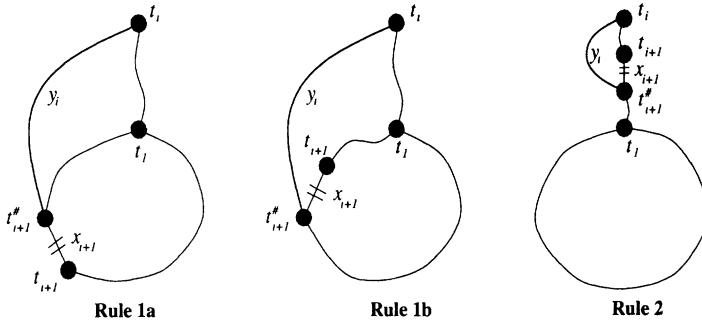


Figure 3: Flower transition add/delete step.

If both edges x'_1 and x''_1 are on the blossom at a given stage, there are two possibilities for closing up the tour; otherwise only one possibility exists as one of the edges x'_1 and x''_1 is required to be deleted. The close-up gains are (see Figure 2d,e):

1. $g'_{i+1} = |x_{i+1}| - |(t_{i+1}, t'_1)| + (|x'_1| - |x_1|)$.
2. $g''_{i+1} = |x_{i+1}| - |(t_{i+1}, t''_1)| + (|x''_1| - |x_1|)$.

The reason that the term with $|x_1|$ enters the close-up gain is that we must adjust for the lack of knowledge when setting up the initial gain g_1 , i.e. that we didn't know which one of the edges x'_1 and x''_1 that was going to be deleted.

The traditional L-K transition does not allow added edges to be deleted at subsequent steps; this in fact means, that all deleted edges must be a part of the original tour. Thus the L-K transition is *connected*, that is, the added and deleted edges make up one

(alternating) cycle. A single L-K transition cannot cover a simple *disconnected 4-Change* (Figure 4), which has shown to be quite successful as a diversifying transition (this transition is called a *kick* in [Martin et al. 1992] and a *super-move* in [Fiechter 1994]).

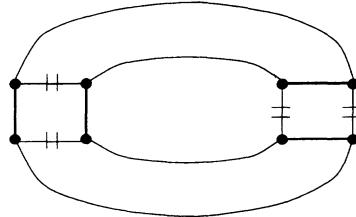


Figure 4: Disconnected 4-Change.

In the current Flower transition implementation all added edges are allowed to be deleted at subsequent steps, except for the first added edge y_1 as this would imply that the identity of the basis would change. This makes the Flower transition capable to generate disconnected transitions.

Briefly, the *variable* candidate set for the Flower transition is generated as follows at level i (recall that $t_1 \in V$, giving x_1 as the longest of the two incident edges to t_1):

1. Given: x_1, x_2, \dots, x_i and $y_1, y_2, \dots, y_{i-1}, \hat{G}_{i-1} > 0$.
2. Select $t_{i+1}^\# \in D_{\Delta(\hat{G}_{i-1} + |x_i|)}(t_i)$ (in order to satisfy relaxed gain criterion $\hat{G}_i > 0$).
3. Determine whether $t_{i+1}^\#$ is on the blossom or on the stem and apply the according rule giving t_{i+1} and x_{i+1} (when the blossom is hit, the longer of two edges incident to $t_{i+1}^\#$ is selected as x_{i+1}).
4. Determine close-up gains g'_{i+1} and g''_{i+1} giving two possible values for g_{i+1}^* .
5. Set $G^* = \max(G^*, G_i + g_{i+1}^*)$ (best close-up value) and check relaxed stopping rule $\hat{G}_i \leq G^*$.

The experimental performance of this transition, both in a simple iterated descent and in a tabu search context, is given in section 5.

4. Applying tabu search to the geometric TSP

This section presents our tabu search approach, concentrating on the application of tabu search to the geometric TSP. In addition, some new ideas, generally applicable to other problems as well, are described. We assume that the reader is familiar with basic concepts of tabu search.

4.1. Attribute based memory

Local search methods are fully described by their neighbour selection strategy, i.e. how the next solution is selected from the neighbourhood $\mathcal{N}(T)$ of the current solution T . In tabu search the neighbour selection depends more or less on the whole search process. In general, the neighbour $T' \in \mathcal{N}(T)$ with minimum objective value $c(T')$ is selected, but a number of neighbours may be *tabu*, since they have been visited recently or have a strong similarity with recently visited solutions.

We have chosen to use the attribute based version of tabu search, in which a number of (solution) attributes define the tabu status of a solution. The undirected edges of a geometric TSP-tour are the natural attributes to be used. The approach is similar to the one used in the intensification algorithm of [Fiechter 1994], but instead of fixed-length tabu lists of edges, we focus on the tabu status of individual edges, since the number of exchanges edges k may vary from iteration to iteration.

The tenures of deleted and added edges are different ($t_{deleted}$ and t_{added} respectively). This is based on the observation that the number of edges that can be added is much larger than the number of edges that can be deleted, meaning that the tenure $t_{deleted}$ of deleted edges (which are candidates to be added at subsequent iterations) should be given a larger value than t_{added} .

When a transition is evaluated a number of the involved edges may be tabu-active. A simple rule would be to require all involved edges to be tabu-active, in order to make the transition tabu, but then a transition with a small k would have a greater probability of being tabu than a transition with a large k . The chance of becoming tabu should be independent on k , and we have therefore chosen the following rule: A transition is tabu, if the percentage of tabu-active deleted edges is higher than $p_{deleted}$ or the number of tabu-active added edges is higher than p_{added} .

The aspiration criterion also differs from the one typically used - a special case of the so-called *region best* aspiration criterion [Glover 1993] which we have named the *recent best* criterion is applied. A solution fulfills the recent best criterion, if it has a better objective value than the t_{RB} most recent solutions. This aspiration criterion is an indication of an interesting search direction.

4.2. Short-term control

Our tabu search algorithm alternates between the two short-term (or low-level) modes of search *descent* and *ascent*. Starting with an initial solution constructed by the *Nearest Neighbour* heuristic, the search first enters a descent. When the search is in this mode, only improving solutions are allowed and the smallest candidate set, $\mathcal{V}(1)$ (section 2), is used since we know that it holds all improving neighbours (if any). A descent ends in a local minimum by using the following selection strategy:

The first C candidates from $\mathcal{V}(1)$ are generated: If no improving solution (tabu or not) is found, the generation continues until an improving solution is found or all solutions in $\mathcal{V}(1)$ have been evaluated. If any improving solution is found, the most improving non-tabu solution is selected - otherwise the most improving tabu solution is selected. If no improving solutions are found, that is, at a local minimum, the least non-improving non-tabu solution is selected and the search enters an ascent mode of search.

In ascent mode δC neighbours from $\mathcal{V}(\delta)$ are generated and the best non-tabu solution selected; the actual value of δ is set by the long-term control of the search (section 4.3). When the recent best aspiration criterion is fulfilled, the search enters a descent again. Thus, we use the aspiration criterion to control the switching between the short-term search modes.

4.3. Long-term control

The long-term or high-level control of search has two counteracting objectives: 1) Direct the search towards good encountered solutions (*intensification*) 2) Drive the search into new unexplored and promising areas (*diversification*). We try to reach these objectives by implementing a long-term memory containing information on the *frequency* of edges involved in transitions and local minima, and on the *identity* (including tour length and number of times visited) of

all local minima visited.

Several bias functions³ based on the relative frequency of these counts have been evaluated. Also a new bias function, based upon a definition of a *solution-distance* function, is introduced. The closeness of two solutions is measured as the number of edges common to both solutions. This bias function has very small memory requirements compared to the above alternatives. By using the current best solution as a reference point, the search can be encouraged to be “close to” or “far from” this reference point, depending on the state of the search.

The sign of the bias function is set in agreement with the intensification or diversification objectives. The alternation between the long-term search strategies and the updating of δ is (eventually) made when a local minimum is reached (initially the intensification search strategy is selected):

```

if new best local minimum then
     $\delta = \delta_{min}$ , LongTermSearchStrategy = Intensification
end if

if no improvement made during the last  $i_{LM}$  local minima then
    if LongTermSearchStrategy = Intensification then
        LongTermSearchStrategy = Diversification
    else
         $\delta = \delta + \delta_{inc}$  /* increase  $\delta$  */
        if  $\delta > \delta_{max}$  then  $\delta = \delta_{min}$  /* reset  $\delta$  */
    end if
end if
```

The search control parameters were tuned by using a number of selected problems. Some parameters are problem size dependent, but nevertheless we chose to use constant values, since the performance was not affected critically by parameter changes. Table 1 gives a summary of the parameter values used in all computational experiments reported in the next section.

5. Computational results

The computational experiments were made on two sets of test problems: 1) Randomly generated instances with points uniformly drawn

³Often referred to as *penalty* functions.

Table 1: Search control parameters.

<i>Parameter(s)</i>	<i>Description</i>	<i>Default value</i>
C	No. of neighbours to check	200
$t_{deleted}$ t_{added}	Tenure deleted/added edges	30 15
$p_{deleted}$ p_{added}	Tabu threshold for edges	80 40
t_{RS}	Aspiration region size	10
δ_{min} δ_{max} δ_{inc}	Control parameters for δ	2 4 0.1
i_{LM}	Non-improvement period	3

in the unit square 2) Selected instances from the TSPLIB problem library [Reinelt 1991]. All tests were run on a HP9000/735 computer system.

The sizes of the randomly generated problems were 500, 1000, 5000 and 10000, and five instances were generated for each size. The results reported are averages for each size, and are given as normalized tour length $c(T)/\sqrt{n}$ in order to make a comparison to [Fiechter 1994].

Table 2 shows the iterated descent results for each of the four transitions 2-Change, 3-Change, L-K and Flower. This table gives the immediate strength of each transition and in addition, it allows us to compare the simple iterated descent method to our tabu search approach. The performance of the new Flower transition is similar to the L-K transition; both transitions are, as expected, much stronger than 2- and 3-Change.

The performance of tabu search using the same transitions is given in Table 3. The overall picture is the same, but the Flower is stronger on smaller problems, while the L-K performs somewhat better on larger problems. All transitions perform better compared to their iterated descent versions, particularly on larger problems.

The disconnected 4-Change is known to be effective as a diversifying transition and this is demonstrated in Table 4, where this transition is used in ascent mode and the same four transitions used in descent mode. Especially the weaker transition benefit greatly from this improvement. The results are also compared to those from [Fiechter 1994] (approximately the same CPU-time on a SILICON GRAPHICS station), and our approach generally obtains a better solution quality. Still, for large problems the heuristic of [Fiechter

Table 2: Random instances: Iterated descent.

Problem size	Time (sec.)	2-Change	3-Change	L-K	Flower
500	100	0.7605	0.7481	0.7408	0.7407
	200	0.7601	0.7460	0.7395	0.7407
	300	0.7601	0.7459	0.7391	0.7405
	400	0.7601	0.7458	0.7390	0.7402
1000	200	0.7573	0.7424	0.7329	0.7325
	400	0.7573	0.7409	0.7324	0.7321
	600	0.7564	0.7409	0.7324	0.7317
	800	0.7562	0.7407	0.7323	0.7316
5000	1000	0.7601	0.7422	0.7321	0.7311
	2000	0.7595	0.7421	0.7310	0.7304
	3000	0.7595	0.7419	0.7310	0.7303
	4000	0.7595	0.7419	0.7307	0.7303
10000	2000	0.7582	0.7406	0.7293	0.7293
	4000	0.7581	0.7402	0.7289	0.7293
	6000	0.7578	0.7402	0.7286	0.7289
	8000	0.7576	0.7401	0.7284	0.7287

All results are given as average normalized tour length.

1994] beats our 2-Change tabu search version.

A number of tests were also made on some well studied real-world instances from TSPLIB for which optimum solutions are known. Our approach was compared with the best heuristic known for the geometric TSP: *Large-Step Markov chains* [Martin et al. 1992], which is based on the disconnected 4-Change and L-K transition (Table 5). Approximately the same amount of CPU-time was assigned to all methods - from 1 to 8 hours depending on the problem size.

Again tabu search is more effective than iterated descent, and the Flower transition generally does a better job than L-K on these real-world instances. The Markov chains heuristic is superior to our tabu search approach, and there seems to be two explanations: 1) A much larger number of unique local minima are visited by the Markov chains heuristic 2) A random diversification strategy is better than a memory based strategy.

The last statement is based on our experiments evaluating the

Table 3: Random instances: Tabu search.

Problem size	Time (sec.)	2-Change	3-Change	L-K	Flower
500	100	0.7646	0.7472	0.7426	0.7391
	200	0.7628	0.7467	0.7412	0.7381
	300	0.7628	0.7467	0.7405	0.7372
	400	0.7628	0.7467	0.7392	0.7370
1000	200	0.7580	0.7403	0.7329	0.7332
	400	0.7566	0.7395	0.7322	0.7310
	600	0.7561	0.7385	0.7316	0.7305
	800	0.7557	0.7381	0.7316	0.7304
5000	1000	0.7581	0.7405	0.7300	0.7320
	2000	0.7576	0.7395	0.7287	0.7294
	3000	0.7570	0.7389	0.7283	0.7285
	4000	0.7567	0.7384	0.7280	0.7282
10000	2000	0.7578	0.7392	0.7281	0.7297
	4000	0.7570	0.7384	0.7266	0.7297
	6000	0.7566	0.7381	0.7263	0.7282
	8000	0.7564	0.7374	0.7260	0.7273

All results are given as average normalized tour length.

different long-term control strategies, i.e. bias functions, described in section 4.3. We were not able to measure any significant difference between various approaches based on attribute counts and our solution-distance approach. Also, none of them seemed to make the search completely independent on the initial solution - even if the bias function changed the original objective function considerably.

6. Conclusion

This paper described a transition type independent tabu search approach for the geometric TSP. It was demonstrated that it is possible and advantageous to use strong transitions in tabu search, even within the same running time frame. The cost is an increased programming effort.

The tabu search approach clearly outperforms iterated descent and the Flower transition is a bit stronger than L-K on real-world instances, but the difference is not very significant. The Large-Step Markov chains method is stronger than our tabu search approach, and its superiority primarily stems from a visitation of more unique

Table 4: Random instances: Tabu search with disconnected 4-Change in ascent mode.

Problem size	Time (sec.)	2-Change	3-Change	L-K	Flower
500	100	0.7488	0.7449	0.7405	0.7390
	200	0.7456	0.7445	0.7394	0.7382
	300	0.7439	0.7430	0.7392	0.7379
	400	0.7429	0.7426	0.7386	0.7378
	*)	0.7488			
1000	200	0.7416	0.7366	0.7322	0.7322
	400	0.7391	0.7360	0.7312	0.7313
	600	0.7380	0.7356	0.7304	0.7305
	800	0.7370	0.7352	0.7304	0.7298
	*)	0.7429			
5000	1000	0.7413	0.7377	0.7303	0.7320
	2000	0.7397	0.7369	0.7295	0.7310
	3000	0.7390	0.7360	0.7287	0.7305
	4000	0.7381	0.7351	0.7282	0.7302
	*)	0.7363			
10000	2000	0.7434	0.7376	0.7285	0.7297
	4000	0.7413	0.7359	0.7278	0.7297
	6000	0.7403	0.7344	0.7274	0.7292
	8000	0.7395	0.7343	0.7270	0.7291
	*)	0.7338			

All results are given as average normalized tour length.

*) Best results from [Fiechter 1994].

Table 5: Library instances: Comparison of local search heuristics.

Problem	ID	ID	TS L-K	TS Flower	Markov L-K
	L-K	Flower			
lin318	0.33	0.16	0.35	0.29	0.22
att532	0.31	0.38	0.22	0.16	0.01
rat783	0.60	0.60	0.28	0.18	0.06
pr2392	1.15	0.90	0.68	0.51	0.32

Average excess (%) from optimum of five independent runs.

ID: Iterated descent.

TS: Tabu search (disc. 4-Change in ascent mode).

Markov: Large-Step Markov chains [Martin et al. 1992].

local minima.

An investigation of the local minima topology for the transitions described might be able to clarify, how memory based diversification strategies can be made more effective.

Acknowledgement

The authors would like to thank Paweł Winter, Jens Clausen and the referees for valuable comments and suggestions.

7. References

- J. L. Bentley, Fast Algorithms for Geometric Traveling Salesman Problems. *ORSA Journal on Computing* **4**(4), 387–411, 1992.
- C.-N. Fiechter, A parallel tabu search algorithm for large traveling salesman problems. *Discrete Applied Mathematics* **51**, 243–267, 1994.
- F. Glover, Tabu Search – Part I. *ORSA Journal on Computing* **1**(3), 190–206, 1989.
- F. Glover, Ejection Chains, Reference Structures and Alternating Path Methods for Traveling Salesman Problems. *Technical report*, Graduate School of Business, University of Colorado, Boulder, 1992.
- F. Glover and M. Laguna, Tabu Search in: *Modern Heuristic Techniques for Combinatorial Problems*, ed. C.R. Reeves (Blackwell, Oxford, 1993).
- D. S. Johnson, Local Optimization and the Traveling Salesman Problem. *Proc. Seventeenth Colloquium on Automata, Languages and Programming*, Springer-Verlag, 446–461, 1990.
- S. Lin and B. W. Kernighan, An Effective Heuristic Algorithm for the Traveling-Salesman Problem. *Operations Research* **21**, 498–516, 1973.
- M. Malek, M. Guruswamy, M. Pandya and H. Owens, Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem. *Annals of Operations Research* **21**, 59–84, 1989.
- O. Martin, S. W. Otto and E. W. Felten, Large-step Markov chains for TSP incorporating local search heuristics. *Operations Research Letters* **11**(4), 219–224, 1992.
- G. Reinelt, TSPLIB - A Traveling Salesman Problem Library. *ORSA Journal on Computing* **3**(4), 376–384, 1991.

Mixing Different Components of Metaheuristics

Irène Charon and Olivier Hudry
École nationale supérieure des télécommunications
46 rue Barrault 75634 Paris Cedex 13 France
E-mails: *charon@inf.enst.fr* and *hudry@inf.enst.fr*

Abstract:

This paper studies the effects of mixing different components of metaheuristics applied to the TSP, namely simulated annealing and a new combinatorial optimization one, called the noising method. The results deal with the comparison of these methods with a given number of evaluations of the function to optimize. Many combinations have been tested: they show that the classical patterns are not always the best.

Keywords: Combinatorial optimization, metaheuristics, simulated annealing, noising method, hybridation.

1. Introduction

The iterative improvement methods, simulated annealing and Tabu search provide heuristics general enough to be adapted to the majority of combinatorial optimization problems. They are sometimes called "metaheuristics" because they are not designed for a specific problem but can be applied to different types of problems. We recently proposed a new one called "the noising method" (see Charon and Hudry (1993)).

This paper studies different methods that are, in some ways, intermediate between simulated annealing and the noising method. To do this, we isolated several components (the way of exploring the neighborhood, the probability of accepting an unfavourable transformation, and other ingredients more linked to the noising method) and we combined them in several manners. All the possibilities are not studied because of their too great number, but we chose to compare ten of them. From our experiments, it seems that the noising method works

better than the simulated annealing method on the TSP; our aim is to try to find which components involve such a conclusion, in order to find interesting hybridizations between these methods later.

The basic method is an iterative improvement one, also called "descent" for a minimization problem. From it, we may derive many others that differ from the descent mainly by the fact that an unfavourable transformation is accepted from time to time, randomly or not. One of them is the simulated annealing method (SA), another one is the Tabu search (TS), a third is the noising method (N). SA and N belong to the ten methods of which the study is described below (for a general presentation of the now "classic" methods, as SA or TS, and for a good selection of references, see for example Pirlot (1992); for more specific surveys, see for instance van Laarhoven and Aarts (1987) or Vidal (1993) for SA and Glover *et al.* (1993) for TS).

2. The Problem and the Ten Studied Methods

The problem considered here is the NP-hard Traveling Salesman Problem (TSP) in the case of an undirected graph: given a weighted undirected graph, find a minimum-weighted Hamiltonian cycle. Two sub-problems are considered and solved in slightly different ways. In the first one, the weights of the complete graph are chosen randomly with a uniform probability distribution on a non-negative integer interval. In the second one, the vertices are located in the plane and the weights of the edges are given by the Euclidean distances between the vertices; in this case, we studied several types of graphs: a graph for which the vertices are distributed randomly with a uniform probability distribution inside the unit square; Euclidean graphs coming from Reinelt (1991); a graph of which the vertices are located on all the crossings of a regular grid (the advantage in these two last cases is that we know the optimal lengths). In the following, n will always denote the number of vertices.

The ten methods that we studied can be described as follows:

- choose randomly an initial solution (a Hamiltonian cycle) which becomes the current solution;
- while a stop criterion is not fulfilled, substitute to the current solution another solution chosen in the neighborhood of the current one.

In all the methods, we store the best solution found since the beginning: it is this solution which is returned by the algorithm.

Three points must be specified: what is the neighborhood of a solution ? how to choose the new solution from the current one ? what is the stop criterion ?

The neighborhood $N(C)$ of a Hamiltonian cycle C is the same for all the methods. In the general case, $N(C)$ is the set of all the cycles that we obtain by applying a 2-opt transformation to C (see Lin and Kernighan (1973)). For the Euclidean graphs with many vertices, we partitioned the unit square into cells in order to get around three vertices per cell, as suggested in Bonomi and Lutton (1984). For this, we divide the side of the unit square into p equal parts, with $p = 2\lfloor 0.5\sqrt{n/3} \rfloor$ (so, the number of cells is around $n/3$ and the average number of vertices in a cell is 3; in addition, p is even and then it is simple to find a rather good initial Hamiltonian cycle; see for instance Bonomi and Lutton (1984) for more details). The neighborhood $N(C)$ of a Hamiltonian cycle C is then the set of the cycles that we get by applying the 2-opt transformation dealing with two edges u and v such that an extremity of u and an extremity of v are in a same cell or in two adjacent cells.

The second parameter to detail is the way of choosing the next solution from the current one. It is one point on which the ten studied methods differ the one to the other. Three ways are possible to explore a neighborhood. The first one consists in choosing randomly a neighbor; if it is not accepted as the new current solution, we choose randomly another one, and so on; we call it a "random exploration" (it is used for instance in SA). In the second possibility, that we call a "systematic exploration", we look, in a deterministic manner, for a neighbor which will be accepted: we generate one after the other the successive neighbors, without generating twice the same, until we find an accepted one (for instance, for the randomly-weighted TSP, if $x_1 \dots x_i x_{i+1} \dots x_n$ is the current Hamiltonian cycle, we try to apply a 2-opt with the edges $\{x_1, x_2\}$ and $\{x_3, x_4\}$, then with $\{x_1, x_2\}$ and $\{x_4, x_5\}$..., until $\{x_1, x_2\}$ and $\{x_{n-1}, x_n\}$, then with $\{x_2, x_3\}$ and $\{x_4, x_5\}$, and so on). The third way is this one used for instance in TS, and which we call an "exhaustive exploration"; it consists in exploring the whole neighborhood in order to choose the best (or the least bad) neighbor.

The stop criterion is given by the number of Hamiltonian cycles that we generate (when we keep them as well as when we reject them). More

precisely, at the beginning we choose a number E of evaluations (an evaluation means the computation of the length of a Hamiltonian cycle; the examination of one Hamiltonian cycle requires one evaluation); when the number of Hamiltonian cycles that we have evaluated (that is, generated) is equal to E , the method is over. The unit that we use to count the number of evaluated Hamiltonian cycles is the neighborhood size NS of a Hamiltonian cycle. For the randomly-weighted TSP with n vertices, NS is equal to $n(n - 3)/2$; for the partitioned Euclidean TSP, NS is around $15n$. We define a parameter e by $E = e \times NS$.

We can describe now the ten methods.

D: this method is based on the descent. To compute the next solution, we apply a systematic exploration of the neighborhood of the current solution until we reach a local minimum. Usually a descent is very much faster than the other methods and needs much less evaluations. Increasing the allocated number of evaluations (stop criterion) cannot improve the result of a descent, as we stop in fact when a local minimum is reached. So, in order to compare this method with the others, when a local minimum is reached, a new descent is restarted with a new random initial Hamiltonian cycle. Then the final result is of course the best solution found until the allocated number of evaluations is exhausted.

SA: this method is in fact a classic SA (in particular, the pattern followed in the case of the partitioned Euclidean TSP is similar to the one applied in Bonomi and Lutton (1984)). The temperature decreases geometrically with a constant ratio always equal to 0.925 in our experiments. The number of temperature changes is a constant given by the user; from it and the total number of allowed evaluations, the number of trials at fixed temperature can be deduced. The neighbors of the current solution are generated according to a random exploration.

SA-R: the only difference with SA is that, periodically, we replace the current solution by the best one found since the beginning; we call "restart" this operation. The aim is to come back near a promising solution whereas we could have gone far from it without having explored its neighborhood. Our experiments led us to choose the following frequency for the restart operation: if the total number of evaluations is equal to $e \times NS$, then the number of evaluations allowed between two restarts is equal to $\sqrt{e} \times NS$ (so it means that there are

around \sqrt{e} restarts).

SA-D: here also, the method is almost the same as SA. But periodically, we insert a descent into the SA process: between two descents, we apply the usual pattern of SA with $4NS$ evaluations. This number could have been considered as a parameter, but it would have been more difficult to tune so many parameters.

SA-RD: we modify SA by adding the restart operation of SA-R and the periodical descents as in SA-D.

SA-S: it is the same as SA but with a systematic exploration instead of a random one: the neighborhood is systematically checked until a transformation is accepted; then the current solution is changed and we go on.

SA-A: it is an "adaptable" SA. First, we define a function u which gives the number $u(p)$ of unfavourable transformations that we wish to accept during NS evaluations, when $p \times NS$ evaluations have already been performed; if the total number of allowed evaluations is $e \times NS$,

we choose $u(p) = (a - b) \frac{(e - p)^2}{e^2} + b$, a and b being parameters of the method with $a \geq b \geq 0$ (so $u(0) = a$, $u(e) = b$, $u'(e) = 0$). After $p \times NS$ evaluations, if the number of unfavourable transformations performed during the last NS evaluations is less than $u(p)$, we multiply the temperature by 1.05 and otherwise by 0.95. Moreover, we use a systematic exploration and, in the general case (but not for the partitioned Euclidean TSP's), we apply the restart operation described in SA-R and the periodic insertion of a descent as in SA-D.

N: this method is the noising method (see Charon and Hudry (1993)). As in SA-S, the exploration of the neighborhood is systematic. Moreover, the temperature parameter of a classic SA pattern is replaced by a new parameter *rate* that is considered as a "rate of noise". It is initialized by *max_rate* and decreases arithmetically (instead of geometrically as in a classic SA) down to a value *min_rate*; *rate* decreases every NS evaluations. When a transformation is proposed, we compute the variation Δ of the length of the current solution; we randomly choose a real *noise* in $[-\text{rate}, \text{rate}]$ with a uniform distribution and we add *noise* to Δ ; the transformation is accepted if $(\text{noise} + \Delta)$ is negative.

N-RD: is the same as N, but with the restart operation described in SA-R and the periodic insertion of a descent as in SA-D.

N-SA: this method is between SA and N. With respect to N, the systematic exploration is replaced by a random one (as in SA); with respect to SA, the acceptance probability of a new solution is given by the law used in N.

3. Comparison Criteria and Implementation

It is difficult to compare the efficiency of metaheuristics (for instance, the conclusions of Hertz and de Werra (1987) and of Johnson *et al.* (1991) are not the same though they deal with the same problem). The conclusions depend on the studied problem, which is here the TSP, and they may not stand for other problems.

For a given instance of a problem, a method can be considered to be better than another if its result is nearer the optimum with the same amount of resources (CPU time, memory space...). When we deal with random methods, as here, we must study them statistically. For the resources, we did not take the memory space into account, because the ten methods are equivalent with respect to this criterion. About CPU time, we considered it as a criterion, of course, but not as the only one, because it may induce a bias since it depends (among others):

- 1) on the computer: it is not the most important thing if all the programs run always on the same computer (as we did), though the interpretation remains computer-dependent; but even on a given machine, we may sometimes observe fluctuations in the CPU time measures for exactly the same operations;
- 2) on the programming itself: the slightest programming clumsiness may involve important consequences, for example by multiplying the CPU time by a non-negligible constant (this explanation is suggested in Pirlot (1992) for the differences quoted above between Hertz and de Werra (1987) and Johnson *et al.* (1991));
- 3) on the adequation of the computer and the programming language to the program: mathematical functions as the exponentiation or the random function may consume an important part of CPU time; then it can be worth adding a specialized co-processor or optimizing these parts of the program in order to speed it up: these elements do not change anything to the methods, but they may inverse their ranks in the experiments.

It is for these reasons that we did not choose CPU time as the main criterion to compare the ten methods. It would be possible to compare them according to the number of accepted transformations; if the updatings involved by accepting a transformation are long, this criterion may be the most relevant: for the TSP, if a Hamiltonian cycle is coded as the succession of the vertices according to the order in which we meet them (with an arbitrary starting point), as it is the case here, then accepting a 2-opt may involve $O(n)$ elementary operations. Another possibility, that we chose, is to compare the methods according to the number of evaluations (that is, the number of Hamiltonian cycles of which we compute the lengths) because it is less sensitive to the drawbacks stated above for CPU time.

The tables of the next sections show the results that we got from a program written in C and running on a SUN station. Each row corresponds to one of the ten methods, for 50 trials with the same tuning of the parameters. At each trial, a new initial solution is randomly computed. The columns give respectively: the average CPU times over the 50 trials, in seconds; the average numbers of accepted solutions during the 50 trials (column "changes"); the average, the best and the worst lengths of the 50 solutions computed by each method; the standard-deviations of these 50 lengths (column " σ "). All the tables correspond to 1 000 NS evaluations.

Figures 1 and 8 show the average lengths of solutions according to the number of evaluations (expressed in NS).

We also draw figures to make the comparison of the methods easier. We have one figure for each graph. On each figure, we have one vertical line for each method (except for D which is too bad). The middle of a line corresponds to the average length for N trials ($N = 50$ or 100). Moreover, it is well-known that, when the number N of the observed lengths is great, the average length follows a Gaussian law of parameters m_0 and σ_0 / \sqrt{N} , where m_0 and σ_0 are respectively the average and standard-deviation of the random variable X associated with one experiment (it does not depend on the law followed by X). The theoretic average that we would like to know is m_0 . m_0 can be located in the interval $[m - 1.96\sigma / \sqrt{N}, m + 1.96\sigma / \sqrt{N}]$ with a probability equal to 0.95, where m and σ are the computed mean and standard-

deviation; this gives the 95 % confidence interval (this does not mean that 95 % of the computed results are in this interval, but that the average value is in this interval with a probability equal to 0.95). The vertical lines of our figures give these intervals.

We have two series of tests: the first corresponds to the general case, with no hypothesis on the weights. The second deals only with Euclidean graphs, and in this case we use the partition as explained before.

4. Results for the general case

We studied many graphs; we report the results only for six TSP's; they are qualitatively the same for the others. The first TSP has $n = 100$ vertices and its weights are integers randomly generated between 1 and 100 with a uniform distribution. The minimum length is not known, but after a great number of trials, including some with much longer times than presented here, we never got less than 286. This graph is named here RwA100. The second is similar but with weights between 1 and 1000; its name is RwB100; the best length that we found is 2258.

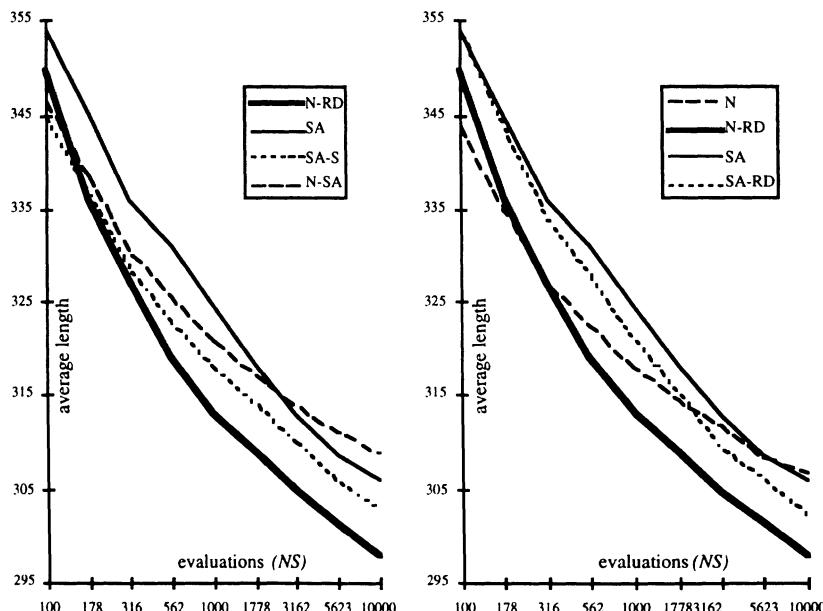


Figure 1: average length of solutions for RwA100

method	time	changes	average	best	worst	σ
D	8 s	43 557	394.91	379	409	7.38
SA	81 s	19 659	324.50	310	336	6.92
SA-R	76 s	19 109	323.03	310	336	7.57
SA-D	51 s	9 093	322.53	308	340	8.53
SA-RD	47 s	8 735	319.40	302	337	7.62
SA-S	51 s	19 659	319.43	302	333	7.91
SA-A	30 s	10 637	312.77	297	335	6.65
N	25 s	8 040	317.96	304	327	5.94
N-RD	21 s	4 568	314.00	296	337	6.73
N-SA	64 s	7 978	320.83	303	332	6.59

Table 1: results for RWA100 with 1 000 NS evaluations

The tuning is the same for all the trials (except the number of evaluations). For each method, we kept of course the best parameters we found from many trials not reported here. Their values are:

- SA, SA-R, SA-D, SA-RD, SA-S: initial temperature = 6, temperature decreasing rate = 0.925, number of temperature decreases = 30.
- SA-A : $a = 10$, $b = 1$.
- N, N-RD, N-SA: maximum noise-rate = 10, minimum noise-rate = 5.

In figure 1, there is nothing about SA-R, SA-D and SA-A; the results of SA-R and SA-D are very close to these of SA (slightly better), the results of SA-A are very close to these of N-RD, and it is not easy to draw too near curves. Table 1 gives detailed results for RWA100 with 1 000 NS evaluations. The results are qualitatively similar to them for RWB100.

The four other graphs are Euclidean TSP's which are described in Reinelt (1991): KroA100, Pr76, Lin105, Pr152; the numbers appearing in the names give their numbers of vertices. For each, we tried the methods with 1 000 NS evaluations. The results are summarized in the figures 4 to 7; we also give the analogous results for RWA100 in figure 2 and for RWB100 in figure 3. For the four Euclidean TSP's, the optima are known: they correspond with the lowest values given on the vertical axis. We indicate, on the horizontal axis, the number of times that this optimal value has been obtained over the 100 trials.

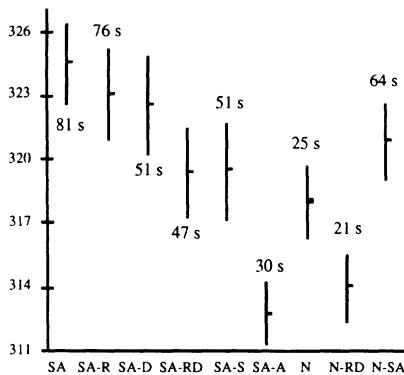


Figure 2: RwA100

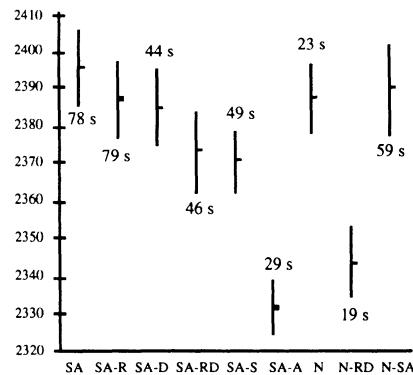


Figure 3: RwB100

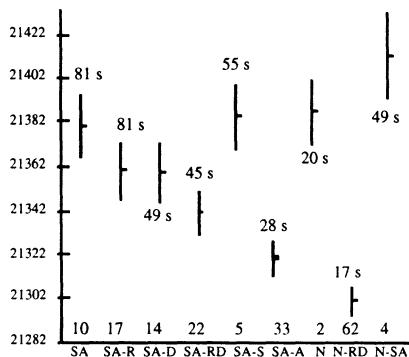


Figure 4: KroA100

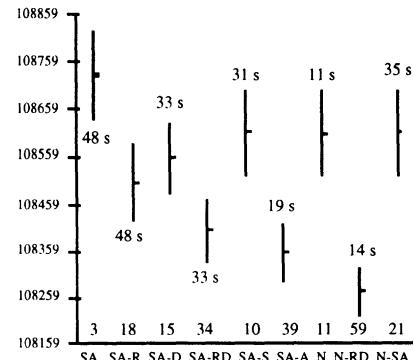


Figure 5: Pr76

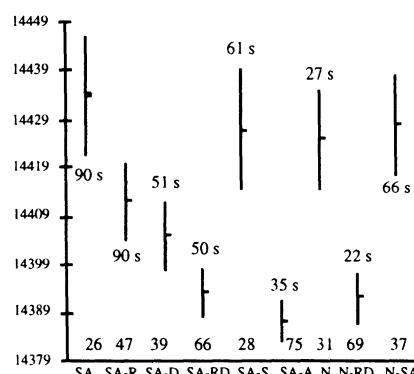


Figure 6: Lin105

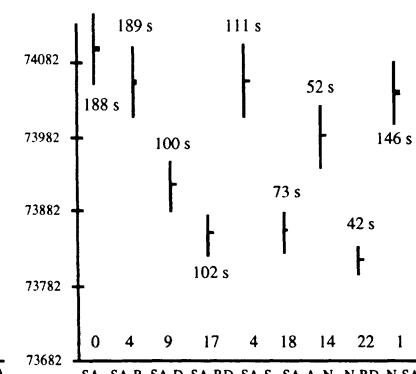


Figure 7: Pr152

The following conclusions may be drawn from these experiments:

- acceptance rule: the comparison between SA and N-SA on the one

hand, and SA-S and N on the other one shows that the use of the noise seems to be better (with respect to the length of the computed solution) than the classic simulated annealing rule when the number of evaluations is rather small; but it is worst for greater values. Nevertheless, we may notice that N and N-SA are around twice faster than SA and SA-S respectively. We may also notice that N and N-SA perform two times less changes than SA and SA-S.

- restart and periodic descents: the comparison between SA and SA-RD and between N and N-RD shows that these two components improve the methods quite a lot, especially when applied together. They involve faster computations and cut down the number of changes by two.
- systematic exploration of the neighborhood: the comparison between SA and SA-S and between N-SA and N shows that a systematic exploration brings a slight improvement while reducing the CPU time by two.
- SA-A is approximatively as efficient as N-RD; we tried also SA-S using the restart operation and periodical descents: it is better than SA-S but much worse than SA-A. So, it seems that to adapt the temperature as the method goes along improves the result.

5. Results for partitioned Euclidean TSP's

We consider now an Euclidean TSP with 1000 vertices belonging to the unit square; their locations are chosen randomly with a uniform distribution on the unit square. This one is divided in $18^2 = 324$ cells. The minimum length is not known; the best one we found is around 23.161. This graph is named here Euc1000.

The tuning is the same for all the trials (except the number of evaluations). The best values that we found for the parameters are:

- SA, SA-R, SA-D, SA-RD, SA-S: initial temperature = 0.02, temperature decreasing rate = 0.925, number of temperature decreases = 30.
- SA-A: $a = 500$, $b = 20$.
- N, N-RD, N-SA: maximum noise-rate = 0.04, mimimum noise-rate = 0.

In figure 8, there is nothing about SA-R, SA-D and SA-A; the results of SA-R and SA-A are slightly better than these of SA; the results of SA-D are very close to these of N-RD (slightly better). As above, Table 2 gives details for the results obtained with 1 000 NS evaluations.

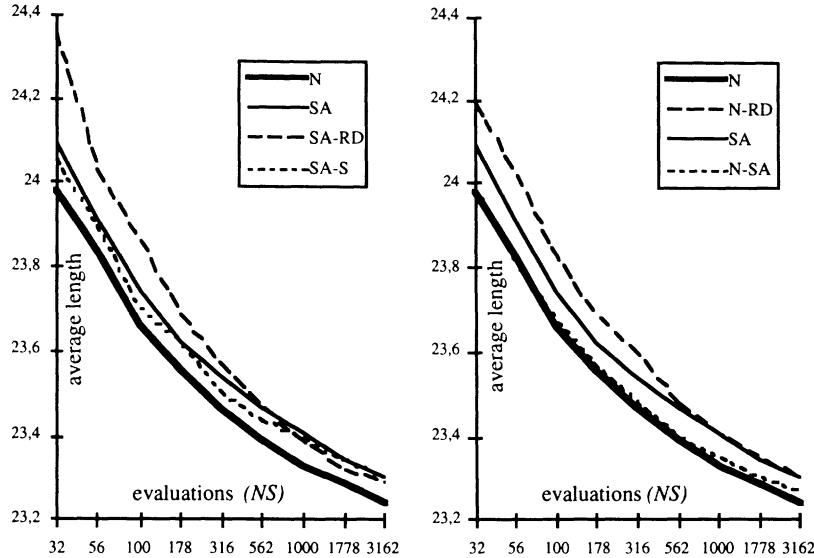


Figure 8: average length of solutions for Euc1000

method	time	changes	average	best	worst	σ
D	127 s	169 237	25.327	25.012	25.608	0.129
SA	615 s	800 413	23.412	23.289	23.582	0.075
SA-R	611 s	798 800	23.386	23.265	23.508	0.068
SA-D	427 s	364 960	23.410	23.235	23.581	0.081
SA-RD	424 s	362 376	23.393	23.274	23.593	0.076
SA-S	314 s	1 057 401	23.400	23.277	23.597	0.071
SA-A	253 s	481 063	23.389	23.234	23.546	0.068
N	206 s	392 252	23.331	23.233	23.571	0.075
N-RD	160 s	189 252	23.409	23.237	23.541	0.073
N-SA	492 s	346 716	23.350	23.242	23.462	0.077

Table 2: results for Euc1000 with 1 000 NS evaluations

The last example that we report is an Euclidean TSP with 2500 vertices located on the crossings of a regular grid in the unit square: their coordinates are $(0.01 + 0.02i, 0.01 + 0.02j)$ with $0 \leq i, j \leq 49$. The unit square is divided in $28^2 = 784$ cells. This graph will be called Grid2500. It is easy to see that, for n even, the minimum length of such a graph is \sqrt{n} , that is here 50: the optimal solution goes only through "horizontal" or "vertical" edges (that is, from a vertex of coordinates (i, j) to a vertex of coordinates $(i \pm 1, j)$ or $(i, j \pm 1)$). A measure of the quality of a

solution is to count the number of "diagonal edges", that is edges from a vertex (i, j) to a vertex $(i \pm 1, j \pm 1)$. For the columns "best" and "worst", we give the number of such edges between brackets.

The best parameters that we found for this instance are:

- SA, SA-R, SA-D, SA-RD, SA-S: initial temperature = 0.015, temperature decreasing rate = 0.925; number of temperature decreases = 30.
- SA-A: $a = 50$, $b = 5$
- N, N-RD, N-SA: maximum noise-rate = 0.03; minimum noise-rate = 0.

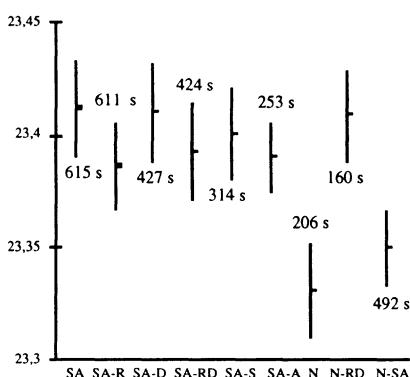


Figure 9: Euc1000

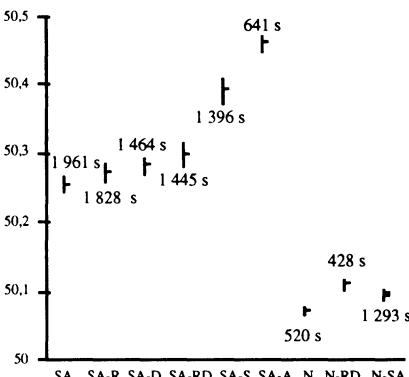


Figure 10: Grid2500

Some of the results are represented in Figures 9 and 10, where the 95 % confidence intervals for the average length of the solutions are represented. Notice also that, with a greater CPU time (5000 evaluations), N was able to find an optimal solution.

method	time	changes	average	best	worst	σ
D	343 s	326 934	53.215	53.081 (372)	53.297 (398)	0.457
SA	1961 s	2 524 208	50.255	50.182 (22)	50.298 (36)	0.032
SA-R	1 828 s	2 519 604	50.272	50.182 (22)	50.365 (44)	0.045
SA-D	1 464 s	1 282 866	50.282	50.199 (24)	50.365 (44)	0.042
SA-RD	1 445 s	1 275 013	50.298	50.182 (22)	50.431 (52)	0.055
SA-S	1 396 s	2 662 674	50.391	50.265 (32)	50.514 (62)	0.064
SA-A	641 s	372 475	50.460	50.364 (44)	50.563 (68)	0.048
N	520 s	832 578	50.071	50.033 (4)	50.119 (14)	0.023
N-RD	428 s	428 550	50.111	50.050 (6)	50.166 (20)	0.026
N-SA	1 293 s	807 492	50.095	50.050 (6)	50.133 (16)	0.023

Table 3: results for Grid2500 with 1 000 NS evaluations

The following conclusions may be drawn from these experiments:

- acceptance rule: the comparison between SA and N-SA, between SA-S and N and also, for Grid2500, between SA-RD and N-RD shows that the use of the noise brings a significant improvement to the methods, more particularly for Grid2500. In addition, the CPU time and the number of changes are also much less for N-SA, N-RD and N than for SA, SA-RD and SA-S respectively.
- restart and periodic descents: unlike the general case, these variants do not involve improvements and may even be harmful. More precisely, the restart operation improves SA slightly for Euc1000 but not for Grid2500. The periodic descents do not improve the lengths found by SA (the results are about the same for Euc1000, and SA performs a little better than SA-D for Grid2500), but SA-D requires less CPU time and much less changes than SA. These two components give about the same results when applied together to SA and worse results when applied together to N, but they speed up both SA and N and reduce the number of changes by a factor 2.
- systematic exploration: the comparison between SA and SA-S shows that the systematic exploration of the neighborhood does not bring significant improvement to SA for Euc1000 and is not a good component for Grid2500, even if it reduces the CPU time. On the contrary, the comparison between N-SA and N shows that the systematic exploration is always a good component in the noising methods: it gives better results in smaller CPU times without increasing the number of changes.
- SA-A, slightly better than SA for Euc1000, is the worst method in the case of Grid2500.

6. Conclusions

Other graphs with other characteristics were considered to test the ten methods; the results are of the same kind and led us to the same qualitative conclusions. Moreover, in a previous study, we compared SA to N and N-RD with another software written in ADA; the results that we got from this first study are utterly consistent with these of the experiments reported above.

The conclusions would not be quite different if the criterion to compare the methods was the CPU time, but it would sharpen the gap between the ten methods: the least performing (at fixed number of

evaluations) methods are also the most CPU time consuming. The reason of this is not only because random drawings and mathematical fonctions need time, but also because SA performs more changes of the current solution.

This study of mixing the components of different metaheuristics should be carried out. For the noising methods (N and N-RD), other variants will be tried: to replace the uniform noise by a Gaussian one; not to center the noise around 0 but on a negative value in order to reject improving transformations less often; to decrease the rate of noise not arithmetically (but, for example, geometrically). An "adaptable" noising method will also be tried, and other functions will be used to guide the temperature of SA or the rate of noise of N. Moreover, some of the methods studied here will be applied to other problems.

7. References

- E. Bonomi and J.-L. Lutton, The N-city traveling salesman problem: statistical mechanics and the Metropolis algorithm, *SIAM Review* 26 (1984) 552.
- I. Charon and O. Hudry, The noising method: a new method for combinatorial optimization, *Operations Research Letters* 14 (1993) 133.
- F. Glover, E. Taillard and D. de Werra, A user's guide to Tabu search, *Annals of Operations Research* 41 (1993) 3.
- A. Hertz and D. de Werra, Using Tabu search techniques for graph coloring, *Computing* 29 (1987) 345.
- D.S. Johnson, C.R. Aragon, L.A. McGeoch and C. Schevon, Optimization by simulated annealing: an experimental evaluation, Part II (Graph coloring and number partitioning), *Operations Research* 39 (1991) 378.
- P.J.M. van Laarhoven and E.H.L. Aarts, *Simulated Annealing: Theory and Practice*, (Kluwer Academic Publishers, Dordrecht 1987).
- S. Lin et B.W. Kernighan, An effective heuristic algorithm for the traveling salesman problem, *Operations Research* 21 (1973) 498.
- M. Pirlot, General local search heuristics in Combinatorial Optimization: a tutorial, *JORBEL* 32 (1992) 7.
- G. Reinelt, TSPLIB - A Traveling Salesman Problem Library, *ORSA Journal on Computing* 3 (1991) 376.
- R.V.V. Vidal (ed.): *Applied Simulated Annealing*, (Springer, Berlin 1993).

A Probabilistic Analysis of Local Search

H.M.M. ten Eikelder¹

M.G.A. Verhoeven¹

T.W.M. Vossen¹

E.H.L. Aarts^{2,1}

¹Eindhoven University of Technology
P.O. Box 513, 5600 MB Eindhoven, Netherlands

²Philips Research Laboratories
P.O. Box 80000, 5600 JA Eindhoven, Netherlands

Abstract

We present a theoretical average-case analysis of a 2-opt algorithm for the traveling salesman problem. First, we give a model which allows us to compute the required number of steps and the distribution of final solutions found by a best improvement algorithm. This model is empirically validated for a restricted version of the 2-opt neighborhood. Secondly, we present a semi-empirical analysis of the average-case performance of an iterated 2-opt and Lin-Kernighan algorithm based on empirically obtained parameters.

Key words: Probabilistic analysis, iterative improvement, iterated local search, traveling salesman problem.

1 Introduction

Local search algorithms constitute a class of approximation algorithms for hard combinatorial optimization problems. The performance of a local search algorithm can be quantified by the relative error of the obtained solutions and its running time. Empirical results show that local search can find good-quality solutions within low-order polynomial running times. It is, however, conjectured that worst-case running times can not be bounded polynomially [Johnson, Papadimitriou & Yannakakis, 1988]. Furthermore, it is not possible to give theoretical upper bounds

on the relative error of local minima. So, there is a considerable difference between the worst-case and empirical average-case behavior of local search algorithms. Theoretical average-case analysis is therefore useful and provides a better understanding of local search algorithms. However, only a few results on this issue are presented in the literature.

In this paper we first present a theoretical average-case analysis of the performance of an iterative improvement algorithm for the traveling salesman problem. We concentrate on the well-known 2-opt algorithm introduced by Croes [1958]. We formulate a probabilistic model, which we use to compute the costs of the solutions found by local search and the required number of steps. The theoretical results fit well with the empirically observed behavior. Secondly, we analyze the average-case performance of iterated local search methods, which can serve as the starting point for the analysis of other more advanced local search heuristics.

2 Local search

An instance of a combinatorial optimization problem is given by a compact representation of a pair (\mathcal{S}, f) , where \mathcal{S} is the set of solutions, and $f : \mathcal{S} \rightarrow \mathbb{Z}$ is a function that gives the cost of a solution. The objective is to find a solution with minimal cost. Local search algorithms are based on repeatedly replacing a solution by a neighboring solution. An essential element in local search algorithms is the concept of a neighborhood structure. A *neighborhood structure* $\mathcal{N} : \mathcal{S} \rightarrow \mathcal{P}(\mathcal{S})$ assigns to each solution a set of solutions, called a *neighborhood*. A solution $s \in \mathcal{S}$ is a *local minimum* w.r.t. \mathcal{N} , if $f(s') \geq f(s)$ for each $s' \in \mathcal{N}(s)$.

Iterative improvement is the basic local search algorithm. An iterative improvement algorithm starts off with a randomly chosen solution or a solution constructed by some heuristic. Next, the algorithm repeatedly tries to improve the current solution by replacing it with a neighbor with lower cost. If a solution has been reached that has no neighbors with lower costs, a local minimum has been found. A pivoting rule determines which neighbor becomes the new current solution. Well-known pivoting rules are *first improvement*, which replaces the current solution with the first lower cost neighbor that is found, and *best improvement* which uses the solution that has lowest cost among all neighbors.

Local search algorithms have the advantage of being generally applicable and flexible, since they only require a specification of the solution space, a cost function and a neighborhood structure. A disadvantage is that local search algorithms may get trapped in local minima of poor quality. To overcome this disadvantage, many variants of the basic local

search algorithm have been proposed in literature. Well-known examples are simulated annealing, tabu search, and genetic local search. For an overview of local search we refer to [Aarts & Lenstra, 1995].

Here we concentrate on the analysis of best improvement algorithms and iterated local search [Johnson, 1990; Martin, Otto & Felten, 1991; Boese, Kahng & Muddu, 1994]. Iterated local search algorithms repeatedly execute an iterative improvement algorithm. Each time the iterative improvement algorithm terminates, the obtained local minimum is modified and the iterative improvement algorithm is restarted with the modified local minimum.

2.1 Local search for the traveling salesman problem

Probably the best-known combinatorial optimization problem is the Traveling Salesman Problem (TSP). In the TSP a salesman wishes to visit a number of cities once and return to the starting point, in such a way that the total distance covered is as short as possible. The TSP belongs to the class of NP-hard problems and therefore considerable effort has been spent in designing efficient approximation algorithms. Formally, the TSP can be defined as follows.

Definition 2.1. Let $V = \{0, \dots, n - 1\}$ be a set of n cities and $d_{ij} \in \mathbb{R}$ a distance for each $i, j \in V$. A tour t is a set of n edges $\{e_0, \dots, e_{n-1}\}$ that constitutes a Hamiltonian cycle in the complete graph $(V, V \times V)$. A tour t is represented by a bijection π , where π_i gives the city at the i^{th} position in t , such that $e_i = (\pi_i, \pi_{(i+1)\text{mod } n})$ and $\pi_0 = 0$. The solution space \mathcal{S} of a TSP instance is the set of all tours. The cost function f is given by

$$f(t) = \sum_{(i,j) \in t} d_{ij}.$$

The problem is to find a tour $t \in \mathcal{S}$ for which $f(t)$ is minimal. \square

We consider only *symmetric* TSP instances, in which distances satisfy $d_{ij} = d_{ji}$ for each $i, j \in V$.

The average cost of the solutions found by local search strongly depends on the choice of the neighborhood structure. Therefore various neighborhood structures have been introduced for the TSP, most of which are based on edge exchanges. In the 2-opt neighborhood \mathcal{N}_2 of Croes [1958] a tour t' is a neighbor of tour t , if t' can be obtained from t by removing two edges and inserting two edges such that t' is obtained. Formally, it is defined as follows.

Definition 2.2. Let $t \in \mathcal{S}$ be represented by π , and define the function $2\text{-exchange}(t, r, s) : \mathcal{S} \times \mathbb{N} \times \mathbb{N} \rightarrow \mathcal{S}$ which gives for each tour the tour that is obtained by removing the two edges at the r^{th} and s^{th} position in the tour and inserting two different edges. More specifically

$$2\text{-exchange}(t, r, s) = t \setminus \{(\pi_r, \pi_{r+1}), (\pi_s, \pi_{s+1})\} \cup \{(\pi_r, \pi_s), (\pi_{r+1}, \pi_{s+1})\}.$$

Then, \mathcal{N}_2 is defined by $\mathcal{N}_2(t) = \{2\text{-exchange}(t, r, s) \mid 0 \leq r, s < n \wedge 1 < r - s < n - 1\}$. \square

For an overview of the worst-case complexity and empirical behavior of local search for the TSP, we refer to [Johnson, 1990].

Other research has addressed the theoretical average-case behavior of local search. Stadler & Schnabl [1992] investigate the structure of the 2-opt neighborhood using simplifications for the dependencies between neighbors which lead to an flawed model. Kern [1989] showed with a probabilistic analysis that the 2-opt algorithm for Euclidian instances of the TSP has an average-case running time that is polynomially bounded. Other probabilistic models for local search have been studied by Tovey [1985]. In this study artificial problems are considered with special neighborhood graphs with regular structures, e.g. the hypercube. The cost function for these problems is chosen to induce an orientation on this graph. Different distributions on the edges are considered and for some cases low-order polynomial average-case running times are proved. More recently, Chandra, Karloff & Tovey [1994] obtained similar results for 2-opt and 3-opt algorithms for the TSP, and derived upper bounds on the average cost of the local minima obtained by these algorithms. Most effort in these papers is focused on proving upper bounds on the average case behavior of local search, whereas we compute the actual distribution of the cost of local minima and the required number of steps, instead of deriving upper bounds on the average-case behavior.

3 A probabilistic analysis of the 2-opt neighborhood

In this section we discuss the distribution of local minima found by best improvement algorithms that use the 2-opt neighborhood structure, and the distribution of the number of steps required to find local minima. The distribution of local minima and the distribution of final solutions found by iterative improvement are not equivalent, because not all local minima have the same probability of being found by an iterative improvement algorithm. It may be the case that local minima with low cost have a larger attraction region than local minima with high cost, and are therefore more often found by local search.

3.1 Preliminaries

An instance of the TSP is completely specified by its distance matrix. As the first step in our approach we assume that the distances d_{ij} are independently drawn from some distribution. This is a common approach in probabilistic analysis; see for instance [Kirkpatrick & Toulouse, 1985; Weinberger, 1991] However, it should be noted that the assumption of independence is in fact a restriction that excludes Euclidean instances, because then the distances are dependent. In view of the above assumption, we can easily define a class of instances by letting the edge lengths \underline{d}_{ij} be independent, identically distributed random variables with mean μ_l and variance σ_l^2 . Consequently, we can also view the cost $\underline{f}(t)$ of a tour t as a random variable, i.e., the sum of n independent, identically distributed edge lengths. Hence, due to the central limit theorem, $\underline{f}(t)$ has approximately a normal distribution with mean $\mu = n\mu_l$ and variance $\sigma^2 = n\sigma_l^2$. The corresponding density is called ω_{tour} .

3.2 The distribution of final solutions

Consider a tour t_0 with neighboring tours t_1, \dots, t_b with $b = |\mathcal{N}_2(t)| = n \cdot (n - 3)/2$. The analysis of best improvement is based on the computation of the *step probability*

$$g(c, c') = \mathbb{P}\{\forall_{i \in \{1, \dots, b\}} \underline{f}(t_i) > c' \mid \underline{f}(t_0) = c\}, \quad (1)$$

i.e., the conditional probability that all neighbors of a tour t_0 have costs at least c' if tour t_0 has cost c . The computation of (1) is discussed in the next section. Various notions can be expressed in terms of the step probability g defined in (1). First, note that $g(c, c)$ is the probability that a tour with cost c is a local minimum. Hence, the density of local minima is given by

$$A^{-1} g(c, c) \omega_{\text{tour}}(c), \quad (2)$$

where A is the probability that an arbitrary tour is a local minimum, which is equal to

$$\int_{-\infty}^{\infty} g(c, c) \omega_{\text{tour}}(c) dc. \quad (3)$$

Next consider a tour t_0 , with neighbors t_1, \dots, t_b , that is not a local minimum. Let \mathcal{S}' denote the set of local minima. Recall that in best improvement a step is made to a neighbor with lowest costs. To investigate such a step, we compute for $c' < c$ the probability

$$\mathbb{P}\{\min_{1 \leq i \leq b} \underline{f}(t_i) \leq c' \mid \underline{f}(t_0) = c \wedge t_0 \notin \mathcal{S}'\} =$$

$$\begin{aligned} \mathbf{P}\{\min_{1 \leq i \leq b} \underline{f}(t_i) \leq c' \wedge t_0 \notin \mathcal{S}' \mid \underline{f}(t_0) = c\} / \mathbf{P}\{t_0 \notin \mathcal{S}' \mid \underline{f}(t_0) = c\} = \\ \mathbf{P}\{\min_{1 \leq i \leq b} \underline{f}(t_i) \leq c' \mid \underline{f}(t_0) = c\} / \mathbf{P}\{t_0 \notin \mathcal{S}' \mid \underline{f}(t_0) = c\} = \\ (1 - g(c, c')) / (1 - g(c, c)), \end{aligned}$$

where we have used the observation that t_0 cannot be a local minimum if $\underline{f}(t_0) = c$ and $\min_{1 \leq i \leq b} \underline{f}(t_i) \leq c' < c$. The density corresponding with the above probability is

$$P(c, c') = \frac{\partial}{\partial c'} \mathbf{P}\{\min_{1 \leq i \leq b} \underline{f}(t_i) \leq c' \mid \underline{f}(t_0) = c \wedge t_0 \notin \mathcal{S}'\} = \frac{-\frac{\partial}{\partial c'} g(c, c')}{1 - g(c, c)}.$$

This expression is the density as function of c' of the cost after one best improvement step, given that the initial tour has cost c and is not a local minimum.

Next, we give recurrence relations that describe the density of the local minima found after k best improvement steps. Let ρ_k be the density of the local minima found after at most k steps and let η_k be the density of the remaining tours. If we start the sequence of best improvement steps with a randomly generated tour, then $\rho_0(c) = 0$ for all c and $\eta_0 = \omega_{\text{tour}}$. After the k^{th} step the density of the residual tours is equal to η_k . Consider such a tour with cost c . There is a probability $g(c, c)$ that it turns out to be a local minimum. This is found out in the $k + 1^{\text{th}}$ step, hence

$$\rho_{k+1}(c) = \rho_k(c) + g(c, c)\eta_k(c). \quad (4)$$

On the other hand there is a probability $1 - g(c, c)$ that the tour with cost c is not a local minimum. Then with probability density $P(c, c')$, it is transformed into a tour with cost c' in the $k + 1^{\text{th}}$ step. Hence,

$$\eta_{k+1}(c') = \int_{c'}^{\infty} \eta_k(c)(1 - g(c, c))P(c, c')dc. \quad (5)$$

This set of recurrence relations allows us to compute the densities of the detected local minima and the residual tours after an arbitrary number of steps. Then, $\lim_{k \rightarrow \infty} \eta_k = 0$ and $\lim_{k \rightarrow \infty} \rho_k = \rho_{\text{fin}}$, the density of the final solutions. Moreover, note that $\rho_k - \rho_{k-1}$ is the density of the local minima found in the k^{th} step. So if $\underline{\text{steps}}$ is the random variable describing the number of steps until a local minimum is found, then

$$\mathbf{P}\{\underline{\text{steps}} = k\} = \int_{-\infty}^{\infty} g(c, c)\eta_{k-1}(c)dc. \quad (6)$$

Finally, we remark that the density of the final solutions can also be obtained from a simple integral equation. Note that we can write $\rho_{\text{fin}}(c) = g(c, c)\phi(c)$ with $\phi(c) = \sum_{k=0}^{\infty} \eta_k(c)$. Then from (5) we infer that ϕ satisfies the following integral equation.

$$\phi(c) = \omega_{\text{tour}}(c) + \int_c^{\infty} \phi(c')(1 - g(c', c'))P(c', c)dc'.$$

3.3 Evaluation of the step probability

The problem now reduces to evaluating the step probability (1). We need the joint distribution of $\underline{f}(t_0), \underline{f}(t_1), \dots, \underline{f}(t_b)$ to compute this probability. As these tours have a large number of edges in common, the corresponding tour lengths are not independent. A simple computation shows that two tours, say s and t , that have k edges in common, have a covariance given by

$$\mathbf{E}\{(\underline{f}(t) - \mu) \cdot (\underline{f}(s) - \mu)\} = k\sigma_t^2,$$

where $\mu = n\mu_l$ is the mean tour length, μ_l the mean edge length, and σ_l^2 the variance of edge lengths. It is easily seen that the tour t_0 and each of its neighbors t_i , with $1 \leq i \leq b$, have $n - 2$ edges in common. However, two neighbors of t_0 , say t_i and t_j , can have $n - 3$ or $n - 4$ edges in common. The case of $n - 3$ common edges occurs if, while going from t_0 to t_i respectively t_j , a common edge is removed or introduced. Consequently the $(b+1) \times (b+1)$ covariance matrix of the random variables $\underline{f}(t_0), \underline{f}(t_1), \dots, \underline{f}(t_b)$ is rather unstructured. Hence it is very difficult to compute or numerically approximate the right hand side of (1) for the 2-opt neighborhood, as we argue in [Vossen, Verhoeven, Ten Eikelder & Aarts, 1995]. Therefore, in the remaining part of this section we concentrate on the neighborhood \mathcal{N}'_2 , which is a restricted version of the 2-opt neighborhood, defined by

$$\mathcal{N}'_2(t) = \{2\text{-exchange}(t, r, h(t)) \mid 0 \leq r < h(t) - 1 \wedge h(t) + 1 < r < n\}.$$

So, whereas in the classical 2-opt neighborhood two arbitrary edges can be removed, here one of the edges to be removed from a tour t is fixed and given by $h(t)$ for a function $h : S \rightarrow 0, \dots, n - 1$. In this neighborhood structure a tour t has only $b = |\mathcal{N}'_2(t)| = n - 3$ neighbors and all these neighbors have $n - 3$ edges in common.

The transition properties and several related notions for the best improvement algorithm can be computed ab initio for the \mathcal{N}'_2 neighborhood structure. If we assume that the edge lengths are normally distributed

with mean μ_l and variance σ_l^2 , the costs $f(t_0), f(t_1), \dots, f(t_b)$ have a joint normal distribution [Papoulis, 1965] with mean μ and covariance matrix R given by

$$\begin{aligned} R_{ii} &= n\sigma_l^2 && \text{for } 0 \leq i \leq b \\ R_{i0} = R_{0i} &= (n-2)\sigma_l^2 && \text{for } 1 \leq i \leq b \\ R_{ij} = R_{ji} &= (n-3)\sigma_l^2 && \text{for } 1 \leq i < j \leq b, \end{aligned}$$

where R_{ii} gives the variance, R_{i0} the covariance between t_0 and its neighbors, and R_{ij} the covariance between two neighbors of t_0 . As (1) is a conditional probability, we first have to compute the conditional density $\omega_{\text{cond}}(c_1, \dots, c_b \mid c)$, i.e. the density of $f(t_1), \dots, f(t_b)$, given that $f(t_0) = c$. It is a known result (see for instance [Papoulis, 1965; Tong, 1990]) that ω_{cond} is again a joint normal density, with all means equal to $\mu' = \mu + R_{i0}/R_{00}(c - \mu) = (1 - 2/n)c + (2/n)\mu$. The variances are given by $r_{ii} = R_{ii} - R_{0i}^2/R_{00} = (4 - 4/n)\sigma_l^2$, and the covariances by $r_{ij} = R_{ij} - R_{i0}R_{j0}/R_{00} = (1 - 4/n)\sigma_l^2$. Now (1) can be rewritten as

$$g(c, c') = \int_{c'}^{\infty} dc_1 \cdots \int_{c'}^{\infty} dc_b \omega_{\text{cond}}(c_1, \dots, c_b \mid c).$$

The essential observation is that now all covariances are equal and positive. In this case the b -fold integral can be simplified to a simple integral [Stuart & Ord, 1987]. The result is that $g(c, c')$ is equal to

$$\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} e^{-\frac{s^2}{2}} \left(\frac{1}{2} \operatorname{erfc} \left(\frac{c' - (1 - 2/n)c - 2\mu_l}{\sqrt{6}\sigma_l} - s\sqrt{\frac{1}{6} - \frac{2}{3n}} \right) \right)^b ds.$$

This integral can easily be computed numerically.

3.4 Empirical results

We compare the theoretically obtained density of final solutions ρ_{fin} and density (6) of the required number of steps with empirically obtained densities. We consider instances with 200 and 400 cities in which edge lengths are distributed according to a standard normal distribution, which implies that tour lengths can be negative. Other distributions of edge lengths in which cost of neighbors can be approximated by a joint normal distribution, can also be used. The results are averages over five instances, although there is little difference between the means of local minima in individual instances since these quantities are self-averaging, i.e., these quantities are equal for instances in which edge lengths are samples from the same distribution, if the number of cities is sufficiently

large [Vossen, Verhoeven, Ten Eikelder & Aarts, 1995]. For each instance we have sampled 20,000 local minima found by a best improvement algorithm. The fixed edge $h(t)$ in the neighborhood $\mathcal{N}'_2(t)$ is randomly chosen for a tour t .

Figure 1 and 2 give the results for instances with 200 and 400 cities, respectively. In these figures, the densities ρ_{fin} of the costs of final solutions found by a best improvement algorithm are given. Furthermore, the density (6) of the number of steps required by a best improvement algorithm to find a local minimum is given. We observe that a good fit between theoretical predictions and empirical results is obtained.

The probability that an arbitrary tour is a local minimum is given by expression (3). For $n = 100, 200, 400$ this probability is $1, 23 \cdot 10^{-4}$, $1, 85 \cdot 10^{-5}$, and $2, 69 \cdot 10^{-6}$, respectively. For five instances with 100 cities we determined empirically the density of local minima by randomly generating 37 million tours of which 4410 turned out to be local minima. The lefthand side of Figure 3 gives the corresponding empirical density as well as the theoretical density of local minima given in (2). Again, we obtain a good fit between theoretical and empirical results.

Finally, the righthand side of Figure 3 presents the density of local minima as given in (2) and the density of final solutions as obtained by iterative best improvement for instances with 100 cities. Typically, the average cost of final solutions is much lower than the average cost of local minima. This indicates that local minima with low cost have a much higher probability of being found by local search than local minima with high cost, which is of course an advantageous property for the average-case performance of local search.

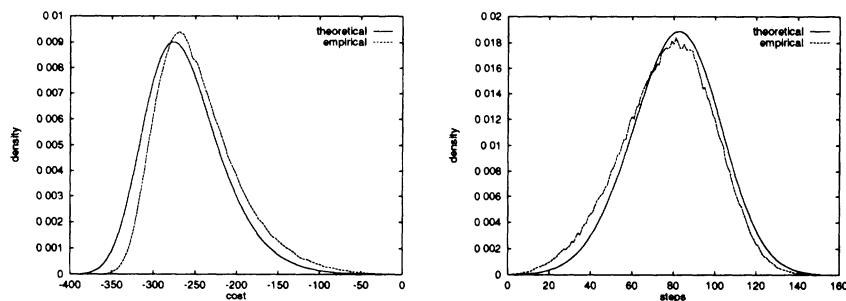


Figure 1: Results for instances with 200 cities.

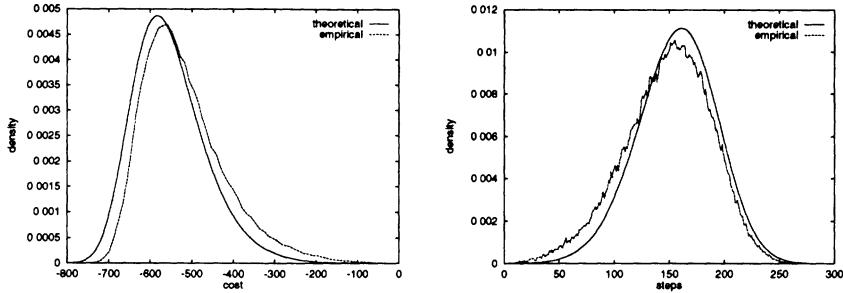


Figure 2: Results for instances with 400 cities.

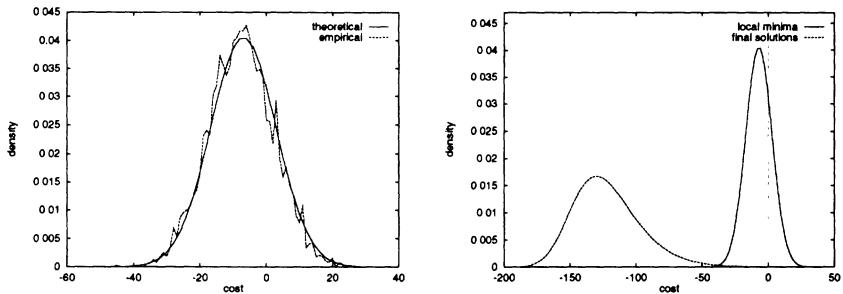


Figure 3: Densities of local minima and final solutions.

4 A semi-empirical analysis of iterated local search

The best known heuristic to handle the TSP is the iterated Lin–Kernighan algorithm of Johnson [1990]. In Johnson’s iterated Lin–Kernighan algorithm the best local minimum found so far is modified by a 4-exchange, replacing four edges by four new edges, and used as starting solution for the next run of the Lin–Kernighan heuristic.

In this section we approximate the average running time an iterated local search algorithm needs to find a solution within a given range from the optimal solution. Time is measured by the number of evaluations, i.e., the number of tours examined by the algorithm. Our approximation is of a semi-empirical nature, which means that the average number of evaluations is expressed as a function of empirically obtained parameters, viz., the distribution of the final solutions and the average running time of the employed iterative improvement algorithm.

To approximate the average running time we need the average number of iterations and the number of evaluations performed per iteration. Furthermore, we assume that an iterated local search algorithm samples local minima, which means that we neglect the intermediate solutions

generated by the iterative improvement algorithm. Let $\underline{itnr}_\epsilon$ be the random variable which gives the number of iterations until a solution with a given relative error ϵ has been found, then

$$\mathbf{P}\{\underline{itnr}_\epsilon \leq i\} = 1 - \prod_{k=1}^i (1 - \mathbf{P}\{f(s_k) \leq (1 + \epsilon)f_{\text{opt}}\}), \quad (7)$$

where s_k is the final tour obtained at iteration k .

In order to determine the average number of iterations needed, we assume that the modification mechanism is primarily a diversification mechanism. Hence, we consider subsequent local minima to be independent, so that probability $\mathbf{P}\{f(s_k) \leq c\}$ is independent of k . It should be noted that this is a rather strong assumption, since the modification mechanism only changes four edges. Given the density $\rho_{\text{fin}}(c)$ of the costs of final solutions, we have

$$\mathbf{P}\{f(s_k) \leq (1 + \epsilon)f_{\text{opt}}\} = p(\epsilon) = \int_{-\infty}^{(1+\epsilon)f_{\text{opt}}} \rho_{\text{fin}}(c) dc.$$

This allows us to express probability (7) as

$$\mathbf{P}\{\underline{itnr}_\epsilon \leq i\} = 1 - (1 - p(\epsilon))^i. \quad (8)$$

Expression (8) implies that the number of iterations is distributed according to a geometrical distribution. So, the average number of iterations to find a solution within a given relative error ϵ is equal to $1/p(\epsilon)$.

In order to approximate the average number of evaluations needed per iteration, we need the following observations. After the first iteration the iterative improvement algorithm starts with a tour that differs only four edges from a local minimum. Consequently, the average number of evaluations required per iteration is substantially lower than required by the first iteration. This is also found empirically. Hence, we differentiate between the following two values.

- k_1 , the average number of evaluations needed to reach a local minimum from a randomly chosen start solution.
- k_2 , the average number of evaluations needed to reach a local minimum from a solution obtained by modifying a local minimum.

The values k_1 and k_2 are obtained empirically. We have noticed that the relative difference between k_1 and k_2 increases as the size of the problem instance grows.

Let $\underline{evals}_\epsilon$ denote the random variable which gives the number of evaluations until a solution with a given relative error ϵ is found. Then,

$\underline{\text{evals}}_\epsilon = k_1 + (\underline{\text{itnr}}_\epsilon - 1)k_2$, and

$$\mathbf{P}\{\underline{\text{evals}}_\epsilon \leq j\} = \mathbf{P}\{\underline{\text{itnr}}_\epsilon \leq \frac{j - k_1}{k_2} + 1\} =$$

$$\begin{cases} 0 & \text{if } j < k_1 \\ 1 - (1 - p(\epsilon))^{\frac{j - k_1}{k_2} + 1} & \text{if } j \geq k_1 \end{cases} \quad (9)$$

The expected number of evaluations needed to find a solution within a given relative error ϵ is equal to $k_1 + (\frac{1}{p(\epsilon)} - 1)k_2$.

4.1 Empirical results

We have analyzed iterated 2-opt and Lin–Kernighan algorithms with a biased 4-exchange of the last found local minimum as modification mechanism. Both algorithms have been tested on instances from Reinelt’s library of TSP instances. In order to acquire the average number of evaluations empirically, 100 executions of the iterated local search algorithm have been performed for each instance.

Figures 4 and 5 show the probability presented in (9) of finding a solution with a relative error ϵ in a given number of evaluations by the iterated 2-opt algorithm for the instance kroB200 and the iterated Lin–Kernighan algorithm for the instance u574, respectively. The parameter $p(\epsilon)$, the probability that a local minimum is within the given relative error ϵ and the parameter k_1 are obtained empirically by performing 1,000 runs with an iterative improvement algorithm. The solid curves represent the theoretically predicted distributions, the dashed curves the empirically obtained distributions. Iterated local search for other instances, not shown in Figure 4 and 5, displays a similar behavior.

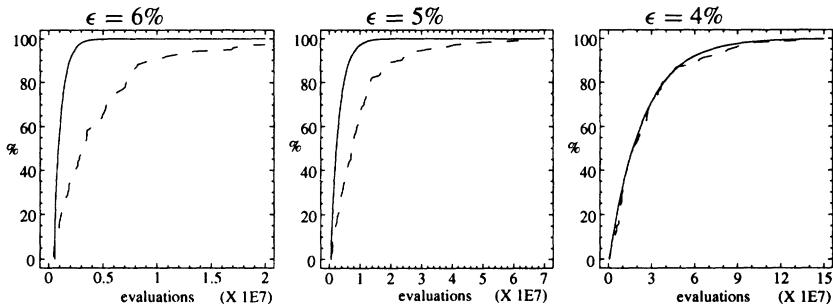


Figure 4: Probability of finding a sub-optimal tour with iterated 2-opt for kroB200.

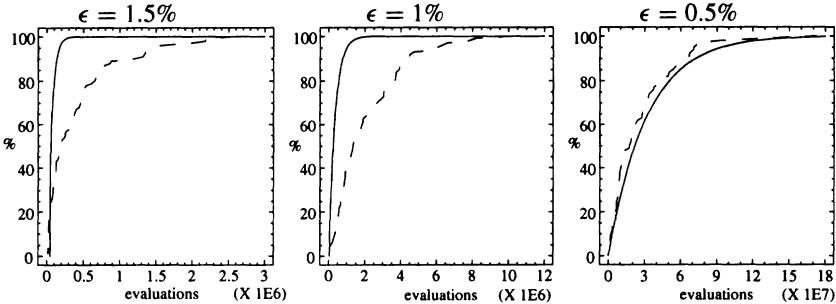


Figure 5: Probability of finding a sub-optimal tour with iterated Lin-Kernighan for u574.

We observe that a good agreement between our theoretical prediction and the empirical results is only obtained for low values of ϵ . This is explained by the observation that even a small under- or overestimation of the probability $p(\epsilon)$ can have a large effect on the theoretical curves. Furthermore, we have found that the empirical distributions fit well with geometrical distributions.

5 Conclusions

In this paper we have discussed the average-case performance, both with respect to quality of final solutions and running times, of a 2-opt algorithm for the traveling salesman problem. We have outlined and empirically validated a probabilistic approach to obtain the distribution of final solutions and required number of steps for a best improvement algorithm that uses a restricted 2-opt neighborhood. An interesting observation is that local minima with low cost have a higher probability of being found by local search than local minima with high cost.

Furthermore, we have given a semi-empirical analysis of the behavior of iterated local search. The main result here is that the time to find a solution within a given relative error is distributed according to a geometrical distribution somewhat shifted to the right to compensate for the preliminary behavior. The parameters of this distribution are determined by the neighborhood and iteration mechanism at hand. The analysis of iterated local search can also be useful as starting point for the analysis of tabu search which also samples local minima.

Future research will concentrate on the analysis of a 2-opt algorithm for Euclidean instances.

References

- AARTS, E.H.L., AND J.K. LENSTRA (eds.) [1995], *Local Search in Combinatorial Optimization*, Wiley.
- BOESE, K.D., A.B. KAHNG, AND S. MUDDU [1994], A new adaptive multi-start technique for combinatorial global optimization, *Operations Research Letters* **16**, 101–113.
- CHANDRA, B., H. KARLOFF, AND C. TOVEY [1994], New results on the old k-opt algorithm for the TSP, *Proc. of the 5th annual ACM SIAM Symposium on Discrete Algorithms*, 150–159.
- CROES, G.A. [1958], A method for solving traveling salesman problems, *Operations Research* **6**, 791–812.
- JOHNSON, D.S. [1990], Local optimization and the traveling salesman problem, *Proc. 17th Colloquium on Automata, Languages, and Programming*, LNCS 447, 446–461.
- JOHNSON, D.S., C.H. PAPADIMITRIOU, AND M. YANNAKAKIS [1988], How easy is local search?, *Journal of Computer and System Sciences* **37**, 79–100.
- KERN, W. [1989], A probabilistic analysis of the switching algorithm for the Euclidian TSP, *Mathematical Programming* **44**, 213–219.
- KIRKPATRICK, S., AND G. TOULOUSE [1985], Configuration space analysis of Travelling Salesman Problems, *Journal Physique* **46**, 1277–1292.
- MARTIN, O., S.W. OTTO, AND E.W. FELTEN [1991], Large-steps markov chains for the travelling salesman problem, *Complex Systems* **5**, 299–326.
- PAPOULIS, A. [1965], *Probability, Random Variables, and Stochastic Processes*, McGraw-Hill.
- STADLER, P.F., AND W. SCHNABL [1992], The landscape of the traveling salesman problem, *Physics Letters A* **161**, 337–344.
- STUART, A., AND J.K. ORD [1987], *Kendall's Advanced Theory of Statistics*, vol 1: distribution theory, Griffin & Company.
- TONG, Y.L. [1990], *The Multivariate Normal Distribution*, Springer-Verlag.
- TOVEY, C.A. [1985], Hill climbing with multiple local optima, *SIAM Journal of Discrete Mathematics* **6**, 383–393.
- VOSSEN, T.W.M., M.G.A. VERHOEVEN, H.M.M. TEN EIKELDER, AND E.H.L. AARTS [1995], *A quantitative analysis of iterated local search*, Computing Science Report 95-06, Eindhoven University of Technology, Netherlands.
- WEINBERGER, E.D. [1991], Local properties of Kauffman's N-k model: A tunably rugged energy landscape, *Physical Review A* **44**, 6399–6413.

The Clustered Traveling Salesman Problem: A Genetic Approach

Jean-Yves Potvin
François Guertin

*Centre de recherche sur les transports
Université de Montréal
C.P. 6128, Succ. Centre-Ville
Montréal H3C 3J7, Canada
E-mail: potvin@iro.umontreal.ca*

Abstract:

The Clustered Traveling Salesman Problem is an extension of the classical Traveling Salesman Problem, where the set of vertices is partitioned into clusters. The goal is to find the shortest tour in such a way that all vertices of each cluster are visited contiguously. In this paper, a genetic algorithm is proposed to solve this problem. Computational results are reported on a set of Euclidean problems, and comparisons are provided with another recent heuristic.

KeyWords: Traveling salesman problem, clusters, genetic algorithms.

1. Introduction

This paper describes a genetic algorithm for solving the Clustered Traveling Salesman Problem (CTSP). This problem is an extension of the Traveling Salesman Problem (TSP), where the vertex set is partitioned into subsets or "clusters" of vertices that must be visited contiguously. Let $G=(V,E)$ be a graph where V is the vertex set and E is the edge set. The vertex set V is partitioned into clusters V_0, V_1, \dots, V_m , where V_0 contains a single vertex called the depot. A non negative cost or distance is associated with each edge. The CTSP consists of determining a least cost Hamiltonian cycle on G such that the vertices of each cluster are visited contiguously. Furthermore, the clusters can be visited in any order.

Chisman (1975) showed that the CTSP can be transformed into a TSP by adding an arbitrarily large constant to the cost of each edge linking two different clusters. After this transformation, any solution procedure for the TSP can be used. Due to the addition of the large constant, the solution procedure introduces the minimum number of inter-cluster edges (i.e., $m+1$). Hence, the vertices within each cluster are visited contiguously, as stated in the formulation of the CTSP. Applications of this problem arise in automated warehouse routing, Chisman (1975), Lokin (1978), and production planning, Lokin (1978). Different heuristics for solving the CTSP with a predefined order on the clusters may also be found in Gendreau et al. (1994).

The paper is organized as follows. First, genetic algorithms are briefly introduced in Section 2. Then, a specialized genetic algorithm is presented in Section 3. Finally, computational results are reported on a set of Euclidean problems, and comparisons are provided with the recent heuristic of Gendreau et al. (1992).

2. A Simple Genetic Algorithm

In order to apply a genetic algorithm to a given problem, solutions must first be encoded as chromosomes (typically, bit strings). Then, a fitness function for evaluating the quality of each chromosome is defined. Using these two components, a simple genetic algorithm can be described as follows, Goldberg (1989):

- Step 1. *Initialization.* Create an initial random population of chromosomes and evaluate each chromosome. Set the current population to this initial population.
- Step 2. *Reproduction.* Select two parent chromosomes from the current population. The selection process is stochastic, and a chromosome with high fitness is more likely to be selected.
- Step 3. *Recombination.* Generate two offspring from the two parent chromosomes by exchanging bit strings (crossover).
- Step 4. *Mutation.* Apply a random mutation to each offspring (with a small probability).
- Step 5. Repeat Steps 2, 3 and 4, until the number of offspring in the new population is the same as the number of chromosomes in the old population.
- Step 6. Evaluate each offspring. Set the current population to the new population of offspring and go back to Step 2.

This procedure is repeated for a fixed number of generations, or until no more improvement is observed. The best chromosome produced during the search is the final result of the genetic algorithm. Through this process, it is expected that an initial population of randomly generated chromosomes with low fitness values will improve as parents are replaced by better offspring. In the following, the main components of the genetic search are briefly described.

First, the encoding transforms a solution to a problem into a bit string. For example, if the problem is to maximize a numerical function f whose domain is a set of integers, then each chromosome can represent an integer in base-2 notation. In this case, the fitness of each chromosome is $f(x)$, where x is the integer encoded in the chromosome.

The reproduction phase favors the best chromosomes through a bias in the selection process. Namely, chromosomes with high fitness values are more likely to be selected as parents (typically, the probability of selection of a given chromosome is proportional to its fitness). Once the parents are selected, the recombination phase creates two new offspring through the exchange of bit strings taken from two parent chromosomes. Here, the underlying assumption is that better chromosomes can be produced by combining bit patterns taken from two good chromosomes. An example of one-point crossover is shown in Figure 1 for two parent bit strings of length 5. First, a cross point is (randomly) chosen between the second and third bit on the parent chromosomes. Then, the parents exchange their end parts to create two new offspring.

1	1		1	1	1	(parent 1)
0	0		0	0	0	(parent 2)
<hr/>						
1	1	0	0	0		(offspring 1)
0	0	1	1	1		(offspring 2)

Figure 1: One-point crossover on two bit strings.

Finally, the mutation operator maintains a minimum level of diversity in the population through random modifications to the bit values. Namely, the mutation operator flips bit values from 0 to 1, or from 1 to 0 on each offspring, with a small probability at each position. A good

level of diversity must be maintained in the population during the genetic search. If the population is homogeneous, the search will not make any further progress because the offspring will be similar to their parents.

In the next section, this simple genetic algorithm will be extended to solve the CTSP.

3. A Genetic Algorithm for the CTSP

Here, the genetic algorithm is applied to an ordering problem. Accordingly, our implementation significantly departs from the classical scheme. In particular, chromosomes are now sequences of integers rather than sequences of bits (where each integer is a particular vertex), and specialized crossover and mutation operators are used to produce new orderings from old ones. Given these characteristics, the main components of the genetic algorithm can now be introduced more formally.

3.1 Representation

As mentioned before, a solution to the CTSP (i.e., a tour) is encoded as a sequence of integers, where each integer is a particular vertex. Implicitly, the last vertex and the first vertex in the sequence are linked together (to form a tour).

3.2 Reproduction

The raw fitness of each chromosome is the length of the CTSP tour encoded on the chromosome. However, these fitness values are transformed using linear ranking to avoid convergence problems associated with the raw values, Baker (1985), Whitley (1989). Then, Stochastic Universal Selection or SUS is applied to the new fitness values in order to select the parent chromosomes, Baker (1985). SUS reduces the variance associated with pure proportional selection (where each chromosome can be selected between 0 and n times over n different trials). Namely, SUS guarantees that the number of selections for any given chromosome is at least the floor, and at most the ceiling, of its expected number of selections.

3.3 Recombination

In the case of the CTSP, hierarchical recombination is applied to the parent chromosomes. That is, recombination is first applied at the cluster level to find a good ordering of the clusters. Then, recombination

is applied at the vertex level to find a good ordering of the vertices within each cluster. In both cases, edge recombination ER was chosen because it has already been successfully applied to permutation problems, like the TSP, Whitley et al. (1989).

The edge recombination operator ER works as follows. Starting from some initial cluster (vertex), the offspring solution is progressively extended through the selection of inter(intra)-cluster edges taken from both parents. During this procedure, the creation of a cycle with previously selected edges must be avoided. Accordingly, an edge is selected at random when the parent solutions only contain edges that lead from the current cluster (vertex) to clusters (vertices) already included in the offspring.

In order to reduce the number of random edges, which are not inherited from the parents, the selection process within ER exploits a data structure known as the "edge map". For each cluster (vertex), this data structure collects the incident edges found on both parents. These parental edges are said to be "active" during the recombination process when they lead to clusters (vertices) not yet included in the offspring. Hence, they can be used to extend the offspring. The heuristic rule for constructing the offspring is to select the parental edge that leads from the current cluster (vertex) to the cluster (vertex) with the minimum number of active edges, among those that still possess active edges and have not yet been included in the offspring. In case of equality between two or more clusters (vertices), the parental edge is selected at random among the available candidates. Using this strategy, ER is less likely to get trapped in a dead end, namely, a cluster (vertex) with no active parental edges (which would imply the selection of a random edge).

Figure 2 shows two parent solutions at the cluster level. In this example, the CTSP is defined over five different clusters: V_0, V_1, V_2, V_3, V_4 and V_5 . At this stage, a new ordering of the clusters will be produced from the orderings found on both parents. Accordingly, the ordering of the vertices within each cluster is not specified.

parent 1 :	V_0	V_1	V_2	V_3	V_4	V_5
parent 2 :	V_0	V_4	V_5	V_2	V_1	V_3

Figure 2: Two parent solutions for the CTSP.

The initial edge map for the two parents is illustrated in Figure 3. This edge map is updated after the selection of each cluster. Choosing V_0 as a starting point, all edges incident to this cluster must first be removed from the edge map (c.f., the column under V_0 in Figure 3). Then, from V_0 we can go to V_1 , V_3 , V_4 and V_5 . Clusters V_1 , V_4 and V_5 have two active edges, while V_3 has three active edges (given that the edges leading to V_0 have been removed). Hence, a cluster is randomly chosen among V_1 , V_4 and V_5 . Assuming that V_1 is selected, the column under V_1 is removed from the edge map. From V_1 , it is possible to go to V_2 and V_3 . Both clusters have two remaining active edges, so we assume that V_3 is chosen and the column under V_3 is removed from the edge map. From V_3 we can go to V_2 and V_4 which both have one remaining active edge, and we select V_2 . From V_2 , we must go to V_5 , and from V_5 we must go to V_4 . Hence, the final offspring is: $V_0 - V_1 - V_3 - V_2 - V_5 - V_4$. Here, the inter-cluster edges V_0-V_1 , V_3-V_2 , and V_5-V_4 are inherited from parent 1, while V_1-V_3 , V_2-V_5 and V_4-V_0 are inherited from parent 2.

	V_0	V_1	V_2	V_3	V_4	V_5
V_0 has edges to		•		•	•	•
V_1 has edges to	•		•	•		
V_2 has edges to		•		•		•
V_3 has edges to	•	•	•		•	
V_4 has edges to	•			•		•
V_5 has edges to	•		•		•	

Figure 3: The initial edge map.

The new ordering is then applied to the two parent solutions (without modifying the sequence of vertices within each cluster). Finally, new orderings of vertices within the clusters are produced by applying ER at the vertex level. Namely, a new ordering of the vertices within a given cluster is produced from the orderings found on the parent solutions (like recombination at the cluster level, but the basic elements are vertices, not clusters). The final offspring is obtained at the end of the application of ER at the vertex level.

3.4 Mutation

The mutation operator is the well-known 2-opt local search heuristic of Lin (1965). Hence, 2-opt exchanges are systematically applied to the current solution until a new improved solution is found (which becomes the new current solution). This procedure is repeated until the current solution cannot be further improved through the application of 2-opt exchanges.

The mutation operator is applied first at the cluster level on the inter-cluster edges and then within each cluster (with a predefined probability at each cluster). Although it would have been possible to iterate over the inter-cluster and intra-cluster levels until no more improvement is found, a single pass was performed due to the fairly high computational requirements.

3.5 Generation Replacement

The new population of offspring replaces the old population. However, elitism is used and the L best solutions at a given generation are maintained in the next generation, where L is a parameter of the algorithm.

4. Computational Results

In this section, the parameter settings for the genetic algorithm are first presented. Then, the test problems are introduced and computational results on these problems are reported. Comparisons are also provided with a recent heuristic.

4.1 Implementation details

The parameter settings of the genetic algorithm were determined empirically through a "trial and error" process. The best values are the following:

- (a) recombination: the crossover rate is set at 60% (i.e., 60% of the parents undergo crossover).
- (b) mutation: the mutation rate is set at 10% (i.e., 10% of the offspring undergo mutation). Then, a 2-opt local search heuristic is applied at the cluster level and within each cluster (with a probability of 50% at each cluster).
- (c) population size: 100
- (d) number of generations: 500

- (e) generation replacement: parameter L is set to 5.

4.2 The test problems

Two different types of Euclidean problems were generated to test the genetic algorithm. These two types are:

- Random (R): the depot and the vertices are randomly generated within a $[0,100]^2$ square, according to a uniform distribution. Then, the vertices are randomly assigned to the clusters, in the order in which they are generated.
- Geometric (G): the depot is located at (50,50) within a $[0,100]^2$ square. Then, the square is divided into equal rectangles, each corresponding to a cluster (see Figure 4). Finally, the vertices are randomly generated within each rectangle, according to a uniform distribution.

V_1	V_2	V_3	V_4	V_5
V_{10}	V_9	V_8	V_7	V_6

Figure 4: Geometric problems.

Problems were generated with $N=100, 300, 500$ vertices and $m=4, 10$ clusters, with each cluster containing N/m vertices. The code was written in C and the tests were performed on a Sun Sparc 10 workstation.

4.3 Computational results on the test problems

The computational results are summarized in Tables 1 and 2 for both the genetic algorithm GA and the GENIUS heuristic, which is a recently developed heuristic for the Traveling Salesman Problem, Gendreau et al. (1992). Here, GENIUS is applied to a transformed cost matrix obtained through the addition of an arbitrary large constant to each edge linking two different clusters. The results shown in the Tables were

produced with a neighborhood size set to 5 (see Gendreau et al. (1992) for details). GENIUS+ is the same as GENIUS, except that it was run as long as GA on each problem by performing multiple runs and by selecting the best solution at the end (given that GENIUS is much less computationally expensive than GA and contains stochastic components as well).

Each number is the average over ten different instances. For each heuristic, the first column is the ratio of the solution produced by the heuristic to a lower bound obtained through the procedure of Jongens and Volgenant (1985). The second column is the computation time in seconds. Five independent experiments were performed on each problem type and the Tables show the best average, the worst average and the overall average, in this order, for each method. Note that these methods can be much closer to the optimum, given that the ratios are taken over a lower bound and not over the true optimum.

As we can see, the gap between the worst and the best average for GA never exceeds 1%, and is often much smaller than 1%. Its averages are within 5.5% of the optimum in all cases, and some averages are even within 1% of the optimum (for the problems of type G with $m=10$). With respect to computation time, GA is faster on problems with a few vertices per cluster (i.e., $m=10$) when compared to problems with a larger number of customers per cluster (i.e., $m=4$). This is quite understandable, given that the optimization problems within each cluster are much smaller.

The same population size and number of generations were used for $N=100$, 300 and 500. Still, the quality of the solutions do not decrease much when the size of the problems increases, apart from the problems of type R with $m=4$ where a degradation of about 2% is observed. In fact, the random problems with $m=10$ and $N=100$ are difficult to solve because inter-cluster and intra-cluster optimization are equally important (no such difficulty is observed on the problems of type G because a good inter-cluster ordering is much easier to find on these well-structured problems).

Clearly, GA substantially closed the gap between the solutions produced by GENIUS and the optimum. However, the improvement provided by the genetic algorithm was obtained through a large increase

in computation time. When GA is compared to GENIUS+, the picture is quite different. No significant difference is observed on the problems of type G. On the problems of type R, substantial differences are observed for $m=10$ only. On these problems, the gap between the two methods decreases from about 2% to 0.5% as the size of the problems increases from $N=100$ to $N=500$. This is an indication that GA better addresses the interaction between intra- and inter-cluster optimization. This interaction is not such an important issue when intra-cluster optimization clearly dominates (i.e., when $m=4$ or when $m=10$ for a sufficiently large number of customers) thus allowing GENIUS+ to do as well as the genetic algorithm.

Table 1: Numerical results produced by GA and GENIUS on random problems.

Type R		GA		GENIUS		GENIUS+
m	N	Ratio	Comput. Time	Ratio	Comput. Time	Ratio
4	100	1.017	28.2	1.050	1.8	1.023
		1.021		1.057		1.028
		1.019		1.054		1.025
	300	1.025	132.0	1.043	10.8	1.031
		1.035		1.047		1.035
		1.029		1.045		1.033
	500	1.035	315.5	1.044	29.2	1.036
		1.037		1.049		1.039
		1.036		1.047		1.038
10	100	1.053	23.1	1.088	1.8	1.072
		1.055		1.105		1.078
		1.054		1.095		1.076
	300	1.042	66.2	1.063	10.1	1.054
		1.045		1.070		1.057
		1.043		1.067		1.055
	500	1.047	139.4	1.061	25.8	1.052
		1.051		1.069		1.055
		1.049		1.064		1.054

Table 2: Numerical results produced by GA and GENIUS on geometric problems.

Type G		GA		GENIUS		GENIUS+
m	N	Ratio	Comput. Time	Ratio	Comput. Time	Ratio
4	100	1.012	27.8	1.026	2.5	1.011
		1.019		1.040		1.013
		1.014		1.031		1.012
	300	1.026	170.4	1.039	10.3	1.025
		1.031		1.043		1.030
		1.027		1.041		1.027
	500	1.025	298.2	1.037	29.4	1.027
		1.027		1.042		1.029
		1.026		1.040		1.028
10	100	1.013	23.4	1.032	1.5	1.012
		1.015		1.045		1.016
		1.014		1.038		1.014
	300	1.008	68.9	1.024	10.1	1.013
		1.011		1.034		1.015
		1.009		1.029		1.014
	500	1.023	118.7	1.037	30.0	1.025
		1.025		1.044		1.028
		1.024		1.040		1.026

5. Conclusion

In this paper, a genetic algorithm for the CTSP was described. This algorithm solved problems with up to 500 vertices within 5.5% of the optimum. It also outperformed the GENIUS heuristic, which was applied to an equivalent TSP (through the addition of a large constant to the cost of the inter-cluster edges). The improvement provided by the genetic algorithm was substantial, but the computational requirements were also more important.

Note also that it would be possible to solve such problems with genetic algorithms previously developed for the TSP, by using the transformed cost matrix (as we did with the GENIUS heuristic).

However, the magnitude of the constant added to the cost of the inter-cluster edges would be an important issue in this case. In particular, the genetic algorithm could quickly converge to the first feasible solution found during the search if the constant is too large, or failed to explore the feasible domain if the constant is too small.

Acknowledgments. This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), and by the Quebec Fonds pour la Formation de Chercheurs et l'Aide à la Recherche (FCAR).

6. References

- J.E. Baker, Adaptive selection methods for genetic algorithms, in: *Proceedings of the First International Conference on Genetic Algorithms*, ed. J.J. Grefenstette (Lawrence Erlbaum Associates, Hillsdale, NJ, 1985) p. 101.
- J.E. Baker, Reducing bias and inefficiency in the selection algorithm, in: *Proceedings of the Second International Conference on Genetic Algorithms*, ed. J.J. Grefenstette (Lawrence Erlbaum Associates, Hillsdale, NJ, 1987) p. 14.
- J.A. Chisman, The clustered traveling salesman problem, *Computers & Operations Research* 2 (1975) 115.
- M. Gendreau, A. Hertz and G. Laporte, New insertion and postoptimization procedures for the traveling salesman problem, *Operations Research* 40 (1992) 1086.
- M. Gendreau, G. Laporte and J.Y. Potvin, Heuristics for the clustered traveling salesman problem, Technical report, CRT-94-54, Centre de recherche sur les transports, Université de Montréal, Montréal, Canada (1994).
- D.E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, (Addison Wesley, Reading, MA, 1989).
- K. Jongens and T. Volgenant, The symmetric clustered traveling salesman problem, *European Journal of Operational Research* 19 (1985) 68.
- F.C.J. Lokin, Procedures for travelling salesman problems with additional constraints, *European Journal of Operational Research* 3 (1978) 135.

- D. Whitley, The genitor algorithm and selection pressure: why rank-based allocation of reproductive trials is best, in: *Proceedings of the Third International Conference on Genetic Algorithms*, ed. J.D. Schaffer (Morgan Kaufmann Publishers, San Mateo, CA, 1989) p. 116.
- D. Whitley, T. Starkweather and D. Fuquay, Scheduling problems and traveling salesmen: the genetic edge recombination operator, in: *Proceedings of the Third International Conference on Genetic Algorithms*, ed. J.D. Schaffer (Morgan Kaufmann Publishers, San Mateo, CA, 1989) p. 133.

A Tabu Search based Heuristic for Arc Routing with a Capacity Constraint and Time Deadline

Richard W. Eglese

Department of Management Science

University of Lancaster

Lancaster LA1 4YX, U.K.

E-mail: R.Eglese@lancaster.ac.uk

Leon Y. O. Li

Department of Management

Hong Kong Polytechnic University

Hong Kong

E-mail: msleonli@hkpucc.polyu.edu.hk

Abstract:

The problem considered is to find the minimum cost set of routes which enable vehicles to service a set of required arcs in a network subject to the service capacity of the vehicles and a time constraint by which each arc must be serviced. The problem is NP-Hard so heuristic methods are required to give good solutions within an acceptable computing time. A new tabu search based heuristic called LOCSAR is introduced and compared with other approaches.

Key Words: arc routing, heuristics, local search, tabu search.

1. Introduction

Arc routing problems arise, for example, when roads require treatment or customers must be served who are located along roads. So practical examples include the routing of people or vehicles for postal deliveries, meter reading and winter gritting, where the roads need to be treated with salt or some other agent to prevent the surface becoming icy. An arc routing problem is represented by a graph where a subset of the arcs must be treated on one or more routes, possibly

subject to constraints. Good surveys of methods and applications of arc routing may be found in Assad and Golden (1995) and Eiselt et al (1994a,b).

The problem considered in this paper is one which arises from the practical application of winter gritting. In this application, routes are required for winter gritting vehicles which treat certain roads with salt when the temperature drops and there is a danger of ice forming. The gritters are able to spread salt on to both sides of the carriageway, so the gritter only needs to travel down a road once in order to treat it, unless there is a central reservation forming a dual carriageway. In these cases, each side of the dual carriageway would be represented as a separate arc in the network. The speed of the gritters while treating a road (the treatment speed) is normally less than the speed when travelling along a road but not treating it (the deadheading speed). The objective is to minimise the total cost of the routes. This is made up of two components: the fixed costs associated with each gritter, including the cost of the driver, and the variable costs which depend on the total distance travelled, mainly represented by fuel costs. In practice, the cost values imply that the minimum number of gritters should be used.

The routes must conform to two constraints. The first derives from the capacity of each vehicle to carry salt and can be written in the form of a constraint on the total length of arcs treated on a route. The second is a service time constraint which stipulates that each road requiring service must be treated by a specified deadline. This means that the time required for each route up to the end of the last road treated on that route must be less than or equal to the specified deadline.

In modelling this application, there is an issue over whether these two constraints should be regarded as hard constraints, which must never be exceeded, or whether they should be regarded as soft constraints which could be exceeded, but at some appropriate penalty cost. It appears to be more appropriate to model the constraints as soft constraints. In the case of the time deadline, this is a level of service determined by the management and in practice the time required to travel each route may vary with the driver and traffic conditions, so planning to be a few minutes over the deadline for a particular route is likely to be acceptable, particularly if this will lead to a reduction in

the number of vehicles required. The capacity of the gritters depends on the rate at which the salt is spread from the gritter. In practice it may be possible to spread salt on a route a little more thinly than originally planned so that the gritter does not run out of salt before completing treatment of the required roads on its route. So a solution where a route marginally exceeds the original capacity constraint may be acceptable in practice.

The type of problem which is therefore considered in this paper can be more formally specified as follows:

Given a connected graph $G(N,A)$ where N is the set of nodes or vertices (corresponding to road junctions) and A is a set of undirected arcs or edges (corresponding to roads) and a set of required arcs which need to be treated R , $R \subseteq A$, vehicles of capacity Q and a time deadline of T , with treatment speed s_t and deadheading speed s_d , then the objective is:

Minimise $z = C_N N + C_D D + C_{CAP} CAP^+ + C_{TIME} TIME^+$
where C_N is the fixed cost for a route,

N is the number of routes,

C_D is the cost per unit distance travelled,

D is the total distance travelled,

C_{CAP} is the penalty cost coefficient for exceeding vehicle capacity,

CAP^+ is the total of the distances by which the routes exceed the capacity limit Q ,

C_{TIME} is the penalty cost coefficient for exceeding the time deadline,

$TIME^+$ is the total of the times by which the routes exceed the time deadline T .

Each required arc must be included for treatment on exactly one route. Each route must start and finish at a specified depot node.

2. Previous Work

It has been shown in Pearn et al. (1987) that an arc routing problem can be converted to an equivalent node routing problem. However the size of the resulting formulation increases and so it appears to be worthwhile to formulate methods which approach the arc routing

problem directly. Up to now there have been few attempts to apply Tabu Search to arc routing problems. One such attempt is due to Greistorfer (1994). Most computational studies have been applied to the CARP and the time deadline constraint has not been considered.

A heuristic previously designed for this problem is described in Eglese (1994). It builds on the approach described by Male and Liebman (1978) for refuse collection routing. The method starts by solving the unconstrained postman problem for the network and finds the deadheading arcs (those which are travelled but not treated) in the optimal single tour. This large tour is then divided into a set of 'cycles' where each cycle consists of a set of treated or deadheading arcs which form a subtour, starting and ending at the same vertex. Vehicle routes are represented by trees of adjacent cycles and an initial solution is found using a savings criterion. The solution is then improved using a Simulated Annealing algorithm.

However although restricting the solution to those made up of the cycles created in the first stage reduced the size of the problem and provided a natural neighbourhood structure for a local search heuristic, better solutions could sometimes be found where the vehicle routes were not made up of cycles formed in the first stage.

The Time Constrained Two Phases Algorithm (TCTPA) was therefore devised (Li 1992) which builds up routes using individual arcs. The heuristic gives a first feasible solution very quickly and is capable of handling other features which are to be found in the practical application, such as the roads being divided into sets having different time deadlines, a mixed capacity fleet of vehicles and the possibility of refilling the gritters at certain locations during their route. In addition, the computer program allows the user to interact with the heuristic to try alternative possibilities and attempt to improve the solution with the help of a map displayed on the screen. Although this heuristic could outperform the 'cycles' based heuristic, the quality of the final solution inevitably depends on the skill of the user and the time spent in trying alternative routes.

There therefore appeared to be a need for a solution method which could attempt to improve an initial feasible solution and search for

better solutions without the need for user involvement and this was the motivation behind the approach described here.

3. A New Heuristic Algorithm: LOCSAR

The new heuristic algorithm presented here is named LOCSAR (LOCal Search for Arc Routing). LOCSAR is based on examining moves from a local neighbourhood of the current solution to search for an improved solution and the current version incorporates features of Tabu Search to guide the search process. The method makes use of ideas which have been used successfully for node routing problems as described by Osman and Salhi (1994) and Gendreau et al.(1993).

An outline of the Tabu Search procedure implemented in LOCSAR is presented in Figure 1.

Step 0: Initialisation

Generate an initial feasible solution, S.

Set the number of iterations, $n = 0$.

Set the best current solution $S_{best} = S$.

Set the number of iterations to the best solution $n_{best} = 0$.

Evaluate all the moves in the neighbourhood $N(S)$ and store a candidate list of moves.

Step 1:

Select the best move from the candidate list and update the current solution from S to S' .

Set $n = n + 1$.

If $z(S') < z(S_{best})$ then set $S_{best} = S'$ and $n_{best} = n$.

Update the tabu list corresponding to the accepted move.

Step 2:

Update the members of the candidate list affected by the move from S to S' .

Step 3:

If $n - n_{best} > n_{stop}$ then STOP else GO TO Step 1.

Figure 1: Outline of the Tabu Search procedure implemented in LOCSAR

3.1 Initialisation

The initial solution is provided by the Time Constraint Two Phases Algorithm described in Li (1992). Different initial solutions can be provided using the interactive facility which is incorporated in this method. The base runs described in this paper used the initial solution provided automatically by this heuristic which has a negligible running time compared to LOCSAR.

3.2 Definition of the neighbourhood

A key decision in designing an arc routing heuristic of this type is to decide what constitutes a neighbourhood move. Greistorfer (1994) , for example, defines a move by taking a single arc which is treated on a route and inserting it elsewhere. However routes are often made up of a series of adjacent arcs which are all treated on the same route. It was therefore felt worthwhile to explore the possibility of moves which are more general than that previously described.

In LOCSAR, a section is defined as a sequence of arcs which are treated consecutively on a route. The length of a section is the number of treated arcs in the section. The arcs do not have to be adjacent, so for a vehicle to travel from the beginning to the end of a section, it may be necessary to deadhead along the shortest path between successive arcs which are being treated, but it will always start and finish with a treated arc.

A move consists of a section being removed from a route and either forming a new route or being inserted into an existing route. The section may be travelled in the original direction or after being reversed. On the route from which the section is removed, the section is replaced by deadheading along the shortest path from node p to node q where node p is the end of the last treated arc before the removed section (or the depot node if the section includes the first treated arc on the route) and node q is the start of the next treated arc after the removed section (or the depot node if the section includes the last treated arc on the route).

The section may be inserted in a route before *any* treated arc on the route or after the last treated arc. Let node r represent the end of the last treated arc before the inserted section (or the depot node if the

section is inserted before the first treated arc) and let node s represent the start of the first treated arc after the inserted section (or the depot node if the section is inserted after the last treated arc). After insertion, the new route will include deadheading along the shortest path from node r to the start of the section, travelling to treat the arcs on the section, and deadheading from the end of the section to node s. If the section forms a new route then both nodes r and s represent the depot node. If the shortest distances between every node in the network are stored, then the change in the value of the objective function due to a section move can be calculated quickly.

Note that a section may be removed from a route and then inserted into the remaining part of the route in a different position and/or direction. This allows the same type of neighbourhood move to be used to improve individual routes. A consequence of defining the neighbourhood in this way is that there may be many moves which remove a section of a route and replace it in that route which have a zero effect on the objective.

For example, suppose there is a route which is currently feasible with respect to the capacity and time deadline constraints and which includes treatment of an arc which is a dead end. If this arc is not the last on the route to be treated, then simply removing this arc from the route, reversing the direction in which it is treated and replacing it in the same sequence of treated arcs on the route is a valid move which has no effect on any of the components making up the objective. In order to prevent the algorithm cycling through many moves of this type, it was decided that if a move involves only one route then it must reduce the current objective in order to be admissible.

In LOCSAR every move is evaluated using every section with length less than or equal to MSL (Maximum Section Length), a parameter set by the user. In addition the method also evaluates moves where a section consists of an entire route from the first treated arc to the last treated arc. This means that the effect of merging any two routes into one route is always evaluated.

For small problems, all neighbourhood moves can be considered and evaluated, but for large problems, it may be worthwhile restricting the

candidate set of moves to exclude those which are unlikely to lead to low cost solutions. This has been implemented by including a parameter NBD_DIST1. If after evaluating the change in deadheading distance resulting from a move, the extra deadheading is more than NBD_DIST1, the program ascribes an artificially high cost to this move and does not carry out the calculations to update the changes to CAP⁺ and TIME⁺. This appears to give a worthwhile speed-up without sacrificing solution quality providing NBD_DIST1 is not too small.

A greater speed-up can be obtained by not considering or evaluating at all any moves where the start of the section to be moved is more than a certain distance from where it is to be inserted. This is controlled by parameter NBD_DIST2. However if this is done, then there is a much greater potential for missing good moves. For example, a route could currently consist of a cluster of arcs close to one another, but far from the depot. It may be optimal also to treat an arc close to the depot as part of this route, particularly if the original route includes deadheading along it and its inclusion will bring the route to the capacity and/or time limits.

3.3 Determination of candidate list of moves

For all neighbourhood moves where a section is moved from route i to route j ($j=0$ denotes the creation of a new route), the value of the least cost move, subject to tabu and other considerations explained below, is stored in an array TABLE[i,j]. If N is the current number of routes, there are $N(N+1)$ candidate moves and another associated array stores the details of the moves. At each iteration, the move which corresponds to the minimum value of TABLE[i,j] is selected so a best-improve strategy is used. The advantage of storing the information this way, which is similar to the data structure suggested by Osman and Salhi (1994) is that when TABLE is updated in Step 2 of the method, if the previous move involved $i=I$ and $j=J$, then the cost of moving a section from route K to route L is unchanged if $K \neq I$ or J , and $L \neq I$ or J . So unnecessary computation can be avoided while retaining a candidate list consisting of best moves.

3.4 Tabu list

In order to prevent the algorithm cycling and returning to the same solutions it is necessary to introduce a condition to prevent this. As it is costly to store and check previous solutions, it is usual to carry this out by not allowing the reversal of moves for a certain number of iterations. These moves are said to be members of a tabu list. A section move from route i to route j is considered tabu if the same section is moved from route j to route i . The tabu restriction is implemented by updating an array $\text{TABU}[i,a]$ where i represents the route number and a represents a required arc. After a move involving a section starting with arc a and finishing with arc b from route i to route j , $\text{TABU}[i,a]$ and $\text{TABU}[i,b]$ are set equal to $n + \text{TLL}$, where n is the current iteration, and TLL is the Tabu List Length. Then a move of a section beginning with arc a and finishing with arc b from route i to route j is only allowed if

$$n > \text{TABU}[j,a] \text{ or } n > \text{TABU}[j,b].$$

The elements of TABU are initially all set to zero. Note that this form of tabu restriction may allow an individual arc to be moved from one route to another and then immediately to be returned to the original route, but not if the arcs moved comprise the same section.

The length of the tabu list is a parameter which needs to be decided. Allowing the length of the tabu list to vary can increase the probability of finding a global optimum. In this algorithm we allow TLL to be chosen at random from a uniform distribution with minimum 5 and maximum 10 as suggested by Glover and Laguna(1993) and used in TABROUTE (Gendreau et al. 1993).

3.5 Aspiration criterion

The tabu list restriction may be over restrictive in forbidding moves which lead to good solutions. After evaluating a move from S to S' , LOCSAR therefore allows its tabu status to be overridden if

$$z(S') < z(S_{\text{best}}).$$

3.6 Diversification

The algorithm as defined so far will efficiently find a local optimum and the tabu list will go some way towards diversifying the search for a better solution to other areas. In some applications of tabu search, this

is enough to obtain very good solutions. However when the algorithm described so far was tested, it was found that with this type of problem, a form of cycling could occur which prevented the search being extended to other areas.

The problem is best explained by an extreme case exhibited by one example. In this example, the depot node was positioned at the end of a single required arc which represented the short access road from the depot itself to the rest of the road network. All routes from this depot needed to travel over this road. A move which removed this arc from one route and inserted it as the first treated arc on another route does not change the number of routes or the total distance travelled. When a local optimum had been found consisting of a solution where CAP^+ and $TIME^+$ were zero and there was some spare time and capacity on each route, this move had a zero effect on the objective function. Although the tabu restriction prevented the arc from being moved from one route to another and back again, there was a tendency for the arc to be moved between all the routes and so produce cycling. It is desirable for the algorithm to explore the effect of moving the arc between routes, but we do not wish the algorithm to become stuck doing that.

The problem was overcome by introducing a factor to encourage diversification of the search. This can be thought of as an aspect of long term memory in contrast to the short term memory implemented by the tabu list. Arcs which have been moved frequently are discouraged from being accepted to move again by adding to the evaluation of the move a factor proportional to the frequency of movement.

Following an approach suggested by Taillard (1992) and Gendreau et al. (1993), if a move from S to S' reduces the value of the objective (i.e. $z(S') < z(S)$) then the move is not penalised, but if the cost of the move is non negative (i.e. $z(S') \geq z(S)$) then the cost of the move is increased by the product of the following four factors:

- i) DELTA_{\max} , the maximum absolute difference in the objective function between two successive iterations,
- ii) N , the number of routes, which is approximately proportional to the square root of the size of the neighbourhood,

iii) DIV_FAC, a scaling factor which is one of the parameters to set in LOCSAR,

iv) FREQ, which is defined as follows:

if n is the iteration number,

n_a is the number of times arc a has been moved,

a is the first arc in the section being evaluated for a move,

b is the final arc in the section being evaluated for a move,

then $FREQ = \text{Max}(n_a, n_b)/n$.

If DIV_FAC is set too low, then the algorithm will tend to cycle in the manner described earlier; if it is set too high, this may lessen the probability of obtaining a good solution. Experimental runs for the data tested suggested that the best results were obtained with a value of DIV_FAC of about 4 for the test problems.

3.7 Stopping Rule

A simple stopping rule is used where the search stops if the objective has not improved within a certain number of iterations, n_{stop} , which is specified at the start of the run. The search may also be terminated if a maximum number of iterations has been reached to avoid extremely long runs.

4. Description of Experiments

LOCSAR was tested on data representing real road networks in the north west of England. For the purposes of the tests all the roads in the networks are regarded as undirected, though LOCSAR can be extended in a natural way to deal with mixed problems involving directed and undirected arcs. The problems tested each involve a single depot, but LOCSAR has been written to deal with multiple depots too. These networks had already been used to test other approaches to the same problem and a lower bound had been calculated for the number of routes and the total distance travelled up to the end of the last road to be treated.

The following parameter values were used in the experiments:

C_N = 10000

C_D = 1 (where distances are measured in metres)

C_{CAP} = 0 (where capacity constraints are not relevant)

= 3 (otherwise)

C_{TIME} = 80000 (where time is measured in hours)

These values were chosen, after some experimentation, in order to give solutions which required a minimum number of vehicles and resulted in solutions likely to be feasible in practice.

There are three networks used for the problems tested. Area East has 77 nodes and 111 arcs. Area South has 140 nodes and 203 arcs. Area North has 254 nodes and 380 arcs. Problems A and B differ in the lengths used for the arcs or the position of the depot. Problems labelled ending with 1 have a single time deadline of 2 hours less 5 minutes loading time and those ending with 2 have an additional capacity constraint of 40 km of treated roads on each route. The speed while treating a road is set to 25 km/h and the speed while deadheading is 40 km/h.

For East and South, n_{stop} = 500 and a limit of 2000 iterations was allowed which was occasionally reached in the case of Area South. For North, n_{stop} = 1000 and a limit of 4000 iterations was allowed. The parameter NBD_DIST1 was set at 5000m. NBD_DIST2 was unlimited for areas East and South and set to 10000m for area North.

The initial solution for each run was provided by the output of the algorithm described in Li (1992) without any user intervention to try to improve the result.

Because the length of the tabu list is controlled by a random number generator, each run was repeated 5 times using a different random number stream for each run. The effect of the random number stream used was not great, but was large enough to make it unsafe to draw conclusions from a single run. The results in Table 1 show both the best result and the mean of the 5 runs .

Table 1: Value of objective for different values of MSL.

		Max. 3	Length of 2	Section 1
East A1	Average	443007	452977	458836
	Best	443007	442390	452672
East B1	Average	467553	482795	466098
	Best	463745	465025	463745
East A2	Average	455748	453013	466215
	Best	452672	452672	453062
East B2	Average	481050	488336	465777
	Best	466277	470291	464948
South A1	Average	852481	875741	857370
	Best	828828	857317	832928
South B1	Average	749627	756863	758027
	Best	735564	738902	739392
South A2	Average	873642	870545	884710
	Best	852914	852562	857335
South B2	Average	734452	741260	759047
	Best	727436	725541	752318

The best result in each row is shown in bold.

5. Conclusions

5.1 The effect of section length

The average and the best solution could be given by runs with this parameter set to 1, 2 or 3, indicating that it is sometimes helpful to allow moves involving more than the movement of a single arc, but setting MSL to a high value does not necessarily result in the best solution value. Generally speaking, the run time was longer the higher the value of the parameter, though even this was not always the case, since, for example the mean run time for problem East A1 was greater when MSL=2 than when MSL=3, due to a greater number of iterations being carried out. Table 2 gives average results for run times and the proportion of moves accepted.

Table 2: Run statistics

	MSL = 3						MSL = 2						MSL = 1					
	Run time	no. iter.	*	1	2	3	Run time	no. iter.	*	1	2	Run time	no. iter.	*	1			
East A1	935	622	1.9	84.6	10.0	3.5	1193	1192	1.9	84.7	13.5	505	943	1.6	98.4			
East B1	1330	1012	1.8	81.5	12.5	4.2	811	838	1.4	80.6	18.0	631	1172	2.2	97.8			
East A2	839	639	1.7	82.6	12.1	3.6	673	704	2.6	82.3	15.1	648	1187	1.1	98.8			
East B2	1453	1103	1.5	79.6	15.1	3.8	755	780	1.5	83.0	15.6	605	1120	2.3	97.7			
South A1	8199	1750	0.5	82.0	12.9	4.6	4929	1502	0.5	84.2	15.3	2547	1378	0.4	99.6			
South B1	5610	1310	1.0	75.7	16.3	7.0	5185	1539	0.5	77.3	22.2	2916	1742	1.0	99.0			
South A2	7200	1626	0.3	83.5	12.3	4.0	4732	1432	0.2	83.2	16.6	2869	1623	0.3	99.7			
South B2	6471	1454	0.6	78.2	14.9	6.4	4660	1404	1.1	77.6	21.3	2519	1553	1.3	98.7			

Run times are average run times in seconds on a PC (Viglen 4DX33). no. iter. = average total number of iterations. Other columns show the percentage number of moves accepted of the specified section length (* denotes a move where the section moved represents a complete route and the section length > MSL).

The runs were repeated where moves represented by * in Table 2 were not allowed, i.e. sections representing complete routes with a section length > MSL. In almost every case, even when the total number of routes in the final solution was the same as the initial solution, the result was worse. Even though these moves form a small proportion of those accepted, it appears valuable to allow them.

5.2 Comparison with other methods

Table 3 compares the best feasible solutions from LOCSAR with the Time Constrained Two Phases Algorithm (TCTPA), which had already been shown to give better results than the cycles based method (Li 1992). It shows that LOCSAR is able to produce solutions which are an improvement on the best solutions found so far. The table also gives a lower bound for the number of routes and the distance from the depot to the end of the last treated arc (Dist.LTA) summed over the routes which was derived in Li (1992).

Table 3: Comparison of best results

		Lower Bound	TCTPA Automatic	TCTPA Interactive	LOCSAR
	No. Routes	6	7	6	6
East A1	Total Dist.		432089	411020	382390
	Dist.LTA	272776	344396	304564	306834
East B1	No. Routes	6	7	7	7
	Total Dist.		441087	440247	393745
	Dist.LTA	280962	350944	330836	316551
	No. Routes	9	13	12	11
South A1	Total Dist.		965256	911809	738781
	Dist.LTA	478320	691160	649569	559054
	No. Routes	9	12	10	10
South B1	Total Dist.		869414	747224	636637
	Dist.LTA	434025	664509	531922	520970
	No. Routes	14	19	17	16
North A1	Total Dist.		1518959	1399849	1176682
	Dist.LTA	705408	1042651	927176	869311

6. Further Research

Research is continuing with LOCSAR and there are several areas which warrant further consideration.

The tests so far have been conducted for each problem using the same initial solution for each run. The algorithm is designed to allow diversification so that the final results do not depend too heavily on the initial solution used, but it may be worth encouraging a greater degree of diversification by running the algorithm from a variety of initial solutions.

Although a fairly general neighbourhood move has been implemented, it may be worthwhile allowing other forms. For example, a move which swapped two sections from different routes might be considered. There might also be some advantage in changing the value of MSL dynamically as the algorithm proceeds.

In LOCSAR, individual routes are improved by the section moves described. However other methods could be used to improve individual routes. If the required arcs on an individual route are connected then the minimum distance route can be found efficiently. If the number of arcs treated on a single route is not too large, it may still be possible to find the minimum distance route quickly, and this could be used at certain stages during the algorithm and at the end.

The determination of the best parameters to use in the objective function is not easy. Even if it is possible to estimate realistic values representing how management value the trade-off between feasible and infeasible solutions, using these fixed values in the algorithm may not necessarily give the best final results. Gendreau et al. (1993) show how the value of penalty weights can be varied within a tabu search scheme designed to produce good feasible solutions. Maret and Wright (1994) have some suggestions for dealing with multi-objective problems which could be explored in this context.

LOCSAR currently implements a form of tabu search, but it would be possible to use a form of simulated annealing based on the same neighbourhood structure.

7. References

- A.A. Assad and B.L. Golden, Arc Routing Methods and Applications, in: *Handbook on Operations Research and Management Science 8: Network Routing*, ed. M. Ball, T. Magnanti, C. Monma and G. Nemhauser (Elsevier, Amsterdam, 1995).
- R.W. Eglese, Routeing Winter Gritting Vehicles, *Discrete Applied Mathematics*, 48 (1994) 231.
- R.W. Eglese and L.Y.O. Li, Efficient Routeing for Winter Gritting, *J. Opl Res. Soc.*, 43 (1992) 1031.
- H.A. Eiselt, M. Gendreau and G. Laporte, Arc Routing Problems. Part I: The Chinese Postman Problem, Report #960 from Centre de recherche sur les transports, Université de Montréal, C.P. 6128, succursale A, Montréal, Canada (1994a).
- H.A. Eiselt, M. Gendreau and G. Laporte, Arc Routing Problems. Part II: The Rural Postman Problem, Report #961 from Centre de

- recherche sur les transports, Université de Montréal, C.P. 6128, succursale A, Montréal, Canada (1994b).
- M. Gendreau, A. Hertz and G. Laporte, A Tabu Search Heuristic for the Vehicle Routing Problem, Report #777 from Centre de recherche sur les transports, Université de Montréal, C.P. 6128, succursale A, Montréal, Canada (1993).
- F. Glover and M. Laguna, Tabu Search in: *Modern Heuristic Techniques for Combinatorial Problems*, ed. C.R. Reeves (Blackwell, Oxford, 1993).
- P. Greistorfer, Computational experiments with heuristics for a Capacitated Arc Routing Problem, Working paper, Department of Business, University of Graz, Austria (1994).
- L.Y.O. Li, Vehicle Routeing for Winter Gritting, Ph.D. Thesis, Lancaster University (1992).
- L.Y.O. Li and R.W. Eglese, A Lower Bound for the Time Constrained Arc Routeing Problem, Working paper presented at CO92, Oxford (1992).
- J.W. Male and J.C. Liebman, Districting and Routing for Solid Waste Collection, *Journal of the Environmental Engineering Division, ASCE*, 104 (1978) 1.
- R. Marett and M. Wright, (1994) The value of distorting subcosts when using neighbourhood search techniques for multi-objective combinatorial problems, Working paper, Department of Management Science, University of Lancaster, Lancaster, U.K.
- I.H. Osman and S. Salhi, Local search strategies for the vehicle fleet mix problem, Working paper, UKC/IMS/OR93/8, Institute of Mathematics and Statistics, University of Kent, Canterbury, UK (1994) .
- W.L. Pearn, A. Assad and B.L. Golden, Transforming arc routing into node routing problems, *Comput. & Ops Res.*, 14 (1987) 285.
- E. Taillard, Parallel iterative search methods for vehicle routing problems, Working paper, ORWP 92/03, Département de Mathématiques, Ecole Polytechnique Fédéral de Lausanne, Switzerland (1992).

Supervision in the Self-Organizing Feature Map: Application to the Vehicle Routing Problem

Hassan Ghaziri

*Graduate School of Business and Administration
American University of Beirut
Beirut Lebanon
E-Mail: ghaziri@layla.aub.ac.lb*

Abstract:

In this paper, we describe a neural heuristic to find an approximate optimal solution to the vehicle routing problem (VRP) with capacity and time limits constraints. This heuristic is based upon the hierarchical deformable nets (HDN) with stochastic competition introduced by Ghaziri [1991] to solve the VRP with capacity constraints only. For the time limits constraint we have added a constructive procedure to the HDN algorithm. Partial trial solutions are generated one of them is selected using a stochastic competition. In order to improve the results of this approach we have introduced a supervision process to the primary unsupervised algorithm. The performance of this hybrid approach is compared with the unsupervised algorithm and the methods based on other heuristics. We have tested the performance in terms of quality and CPU time consumption. We found that the solutions of the traditional heuristics are still better than the neural heuristics but their CPU time consumption is higher.

Key Words: Vehicle routing problem, neural networks, self-organizing feature maps, supervision.

1. Introduction

Neural networks are an emerging computational technology used in a variety of applications ranging from pattern recognition to non-linear data analysis. In this paper, we will show how neural networks (NN) can be used successfully to handle hard combinatorial optimization problems. We will focus in this study on the application of NN to the VRP with capacity and time limits constraints. The first neural network used in the field of combinatorial optimization problems was introduced by Hopfield and Tank [1985]. In this model an energy function is associated with the neural network. The feasible solutions of combinatorial optimization problems (COP) are encoded in terms of physical states and the cost function in terms of energy. Then using the dynamical rules introduced by Hopfield, the network converges toward a stable equilibrium state corresponding to a local minimum of the cost function. It is quite easy to encode any COP in the Hopfield model framework. For this reason most of the NN used to solve COP were based upon this approach. However, the quality of the solutions obtained by this approach was very poor. In fact, the Hopfield network is always trapped in very bad local optimum. When applied to the TSP this method can solve very small problems (10-30 cities). It is clear that such results were far from convincing the operations research community to explore the possibility of using NN in COP. But the Hopfield network is not the only neural network. Another kind of networks called competitive networks were also used to solve COP. There are mainly two variants of competitive networks applied to the TSP. The elastic net [Durbin and Willshaw, 1987] and the self-organizing feature maps [Fort, 1988]. The results of these heuristics are by far better than those obtained by the Hopfield model. They are able to handle real size problems and not only toy problems. We have extended the heuristic based upon the self-organizing feature map approach to the VRP [Ghaziri, 1991]. In section 2 we will describe briefly this heuristic and we will show how it can be extended to handle another side constraint, namely the time limits constraint. This constraint was never introduced using the neural heuristics.

In order to improve the quality of the results we have added to the primary unsupervised algorithm a supervised process described in section 3. In section 4 we compare the performance of this new

heuristic with the classical methods and the unsupervised algorithm.

2. Description of the hierarchical deformable net (HDN) for the vehicle routing problem

The purpose of this section is to describe an algorithm based upon the self-organizing feature map for the VRP. It is called HDNT and is an extension of the hierarchical deformable net (HDN) presented in [Ghaziri, 1991].

The VRP consists of designing delivery routes, serving a set of customers, at the minimum cost with the following restrictions:

- The vehicle routes start at a depot, visit a subset of customers and end at the same depot,
- each customer is supplied by exactly one vehicle,
- a demand is associated with each customer . The total demand of a vehicle supplying a subset of customers must not exceed the vehicle capacity,
- the fleet of vehicles consists of identical vehicles. All vehicles have the same capacity,
- every customer requires a service time and the total travel time of any route must not exceed a prespecified bound.

We will use the following notations;

n = the number of customers;

N = the set customers, $N = \{1, \dots, n\}$;

m = the number of vehicles;

V = the set of vehicles, $V = \{v_k | k = 1, \dots, m\}$;

q_i = the demand of customer $i \in N$ ($i = 0$ denotes the depot, $q_0 = 0$) ;

δ_i = the service time of the customer $i \in N$ ($\delta_0 = 0$);

Q = the vehicle capacity;

N_k = the set of customers serviced by vehicle v_k ;

Q_k = the current capacity of the vehicle v_k (the sum of the customer s demand assigned to the vehicle v_k) ;

d_k = the distance traveled by the vehicle v_k ;

D_k = the total cost of the vehicle v_k route. This cost includes the travel time d_k and the service times δ_i of customers $i \in N_k$;

$\vec{x}_i = (x_i, y_i)$ = position of the customer i in the euclidean space;

R_k = the ring k. A ring is set of neurons.

u_{ik} = neuron i belonging to the ring R_k ;

$\vec{u}_{ik} = (x_{u_{ik}}, y_{u_{ik}})$ = position of the neuron u_{ik} in the euclidean space;

u_{i^*k} = nearest neuron in the ring R_k to the presented customer;

$d_L(u_{ik}, u_{jk})$ = a neighborhood relation between neurons called lateral distance;

h, h', h'' = control parameters;

The network designed to solve the VRP consists of a set of deformable rings. Each ring represents a vehicle. At the initial state the rings are represented by circles. These rings will interact with the position of the customers. The position of their neurons are adapted according to an adaptation rule. We begin by presenting randomly the customers i . The first step in the adaptation rule is to assign a ring to the presented customer. This assignment is a stochastic one, which means that we define a probability of acceptance $Pr(i, R_k, Q_k, D_k, h, h', h'')$. Once the customer is assigned to a ring, we update the positions of the neurons belonging to this ring and only to this ring according to the adaptation rule presented by Fort [1988] to solve the TSP. At each time a customer is assigned to a ring. Its current capacity is updated by adding the demand of the assigned customer. Without the time limit constraint, the acceptance probability is given by:

$$Pr(R_k, Q_k, h, h') = \frac{Pb(R_k, h, h')}{\sum_k Pb(R_k, Q_k, h, h')} \quad (1)$$

$$Pb(R_k, h) = \frac{\exp(-||\vec{u}_{ik} - \vec{x}_j||^2/h)}{(1 + \exp(q_j - Q_k/h))}. \quad (2)$$

h, h' are control parameters playing the same role as the temperature in the simulated annealing algorithm. h, h' are slowly decreased such that the capacity constraint is softly respected. To handle the time limits constraint, we introduce a constructive procedure. When a new customer is presented, the rings compete to interact with it. The idea is to generate partial trial solutions by inserting the new customer i in the vehicle routes. Suppose that we have already assigned a certain number of customers, to the different rings during an iteration. For each ring R_k , we have the current set of

customers N_k assigned to it. For each vehicle v_k represented by the ring R_k we solve a TSP with the already assigned customers , the customer presented i and the depot. To find this solution we can use the Fort algorithm. The solutions obtained are almost optimal because the number of customers assigned to a vehicle in the problems treated, is very small (less than 30 cities in general). Once we have generated the trial solutions we calculate their total cost D_k . Now we have to decide, which vehicle should visit the customer i without violating the time constraint. We follow the same strategy we have used to respect the capacity constraint. We multiply the probability of acceptance by

$$\frac{1}{1 + \exp(D_k - L/h'')} \quad (3)$$

One ring is adapted. This ring is selected according to the new probability distribution . At the beginning of the process, h'' the control parameter is high allowing violation of the time limits constraint. The value of h'' decreases slowly. At the end of the process only the assignments respecting the time constraint are accepted. The main difference between this algorithm and the algorithm presented in [Ghaziri, 1991] is the construction of a partial solution at each step. However, the solution obtained is valid only for one iteration. At each iteration we construct a new solution. Once we have presented all the customers and completed an iteration we have to repeat again this iteration until we meet a stopping criteria. We consider that the stopping criteria is met when the distance between the customers and their nearest neuron is less than a certain quantity. In the HDN , we do not associate a solution with the state of the network during the process. The solution is obtained only at the end of the process once the stopping criteria is met. The customer is associated with the nearest neuron. This neuron determines the vehicle to which the customer is assigned and its position in the vehicle route.

3. The Supervised Hierarchical Deformable Net (SHDN)

The approach presented in the above sections was an unsupervised approach. The decision of assigning a customer to a vehicle and a position in the tour of this vehicle was based on local informa-

tion. In this section, we will present a hybrid approach, in which the vehicle assignment is a supervised decision . The main difference between supervised and unsupervised decision depends on the available information. In a supervised procedure we need more information than the unsupervised one. Supervised learning needs to make comparison between desired outputs and obtained outputs. We have to compute an error function. This function is a global one depending on the state of the network . Our strategy is to generate from the network state a trial solution. We will take advantage from the partial solution obtained in the HDNT algorithm. We have seen that at each step during an iteration, the customer is added to the different vehicle routes, the partial routes are generated by solving a TSP for each vehicle, and the decision of assigning the customer to a vehicle is made using the acceptance probability. Here we will use this approach, not to test the time limit constraint only but to evaluate the solutions.

Our strategy is the following one. At each iteration we present the customers randomly. We start by applying the unsupervised algorithm HDNT. For the last r customers we introduce the supervision. Suppose that a certain number of customers have been already presented and assigned to the different rings representing the vehicles. For each subset of customers assigned to the same vehicle we compute the best route by solving a TSP. It is clear that the depot is added to the different subsets because the depot is visited by all the vehicles. Now we present a new customer and we generate m TSP solutions where m is the number of vehicles. Each solution is obtained by adding this new customer to the route of one of the vehicles and solving a TSP. To compare the different solutions we have to define a cost function. This cost function $E(T(t))$ is a linear combination of the total length of the different tours and a penalty function depending on the violation of the capacity constraint and the time limits violation. This function is given by

$$E(T(t)) = \sum_{v_k} D_k + \alpha \sum_{v_k} [(\sum_{l \in N_k} q_l) - Q_k]^+ + \beta \sum_{v_k} [(d_k + \sum_{j \in N_k} \delta_j) - L]^+ \quad (4)$$

where $[x]^+ = \max(0, x)$, α and β are parameters controlling the penalties for violating the capacity constraints and the time limits constraints. This function was introduced by Gendreau et al. [1994]. Among all trials we select the one for which the cost function is

minimum. The corresponding ring is adapted according to the same rule used in our previous heuristics. The stopping criteria is also the same.

In this algorithm it is possible choose the new solution in a probabilistic way like a simulated annealing procedure .

4. Empirical Results

We performed simulations on a Sun Sparc Workstation.

Parameter Tuning for the unsupervised HDNT. There parameters to be defined are: the number of neurons in each ring, the control parameters, and the updating factor. The total number of neurons must be at least equal to the number of customers, i.e. each ring must have at least $\lceil n/m \rceil$ neurons. Simulations show that we need $2 * \lceil n/m \rceil$ neurons per ring, which is a good compromise between the probability of convergence and the time required to converge . The second parameter is the control parameter h . This parameter is similar to the temperature parameter in simulated annealing. To ensure convergence we have to decrease h slowly. The initial value of is 1 and the updating factor 0.99. The initial value of h' is 3 and the updating factor 0.98. The initial value of h'' is 5 and the decreasing factor is 0.97. These factors are updated after each iterations. An iteration consists of presenting randomly all the customers. At each iteration the customers are presented in a different order. **Parameters tuning for the SHDN.** We use the same parameters as above for the unsupervised step. After 2/3 of the number of iterations needed for the convergence of HDNT, we begin the supervised process. The parameter α increases from 0.1 to 10. We have used the same values for β . When we start the supervised procedure we apply the supervison only to the last 10% customers in each iteration. We have tested HDNT and SHDN on 12 test problems described in Chtistofides et al. [1979]. The size of these problems range from 50 to 199 cities in addition to the depot. The results are given by table 1.

The CPU time needed for the HDNT is higher than the HDN because we have to generate trial solutions and solve m TSP at each step. However the supervised procedure does not increase considerably the CPU time and improves the quality of the results. We have compared the CPU time consumption with the algorithm of

Problem / customers	Clarke and Wright [1964]	Mole and Jameson [1976]	Gillette et Miller [1974]	Gendreau et al. [1994]/time mn.	HDNT /time mn.	SHDN /time mn.
1/50	585	575	532	524.61/5.6	545.1/0.6	539.2/0.9
3/100	886	882	851	826.14/35.3	911.2/5.8	893.4/6.5
4/150	1204	1259	1079	1034.9/96.1	1133.2/10.4	1084.7/13.2
5/199	1540	1545	1389	1329.29/141.6	1421.5/16.4	1401.4/23.2
6/50	619	599	560	555.43/14.1	567.3/3.2	561.3/4.3
8/100	973	999	888	865.94/25.1	902.4/ 14.3	894.3/18.4
9/150	1426	1289	1230	1189.79/38.4	1304.5/23.6	1264/27.2
10/199	1800	1770	1518	1421.88/83.7	1604.3/45.3	1580.4/52.4
11/120	1079	1100	1266	1043.94/51.9	1113.4/3.4	1102.4/4.2
12/ 100	831	879	937	822.85/13.8	826.6/1.2	825.1/1.7
13/120	1634	1590	1770	1551.63/81.3	1645.8/24.3	1608.5/31.3
14/100	877	883	949	866.37/97.1	881.5/6.7	879.8/8.5

Table 1.

Gendreau et al. because they computed the solutions on the same type of machines a SunSparc 2 workstation.

5. Conclusion

Our first contribution in this paper is to introduce the time limits constraints. The main idea is to generate partial trial solutions at each time we present a new customer by solving a TSP for each subset of customers assigned to the same ring. The new customer is assigned to one of the vehicles according to a probability distribution similarly to the heuristic used for the VRP with the capacity constraint only. This paper shows that self-organizing networks are quite flexible to handle real and complex constraints. The second contribution is to take advantage from the generation of trial solutions and introduce a supervision procedure to the primary unsupervised heuristic. This supervision has improved the quality of the solutions without increasing considerably the CPU time. It has to be noted that the solutions obtained are not competitive with the best traditional heuristics [Osman 1993, Gendreau et al. 1994] though the gap between the neural performance and the traditional one is reduced. Better results could be obtained by refining the supervised procedure. The CPU time consumption of our heuristics are small compared to the traditional heuristics. A hybrid approach combining traditional and neural heuristics could reduce considerably the CPU time consumption and keep the same quality of the solutions.

Acknowledgments This work was supported by the Swiss Foundation for Scientific Research

6. References

Chritofides. N., Mingozzi A., and Toth P., (1979) The vehicle routing problem, in Chritofides. N. et al. (eds), Combinatorial Opti-

- mization, Wiley, 315-338.
- Chritofides. N. and Eilon S. (1969), An Algorithm for Vehicle Dispatching Problem. *Opl. Res. Q.*, vol. 20, pp. 591-601.
- Clarke, G. and Wright J.W. , (1964), Scheduling of Vehicles from a Central Depot to a Number of Delivery Points, *Operations Research* vol. 12, pp. 568-581.
- Durbin, R. and Willshaw, D. (1987), An analogue approach to the TSP using an elastic net method. *Nature* 326, 689-691
- Fort, J.-C., (1988), Solving a combinatorial problem via self-organizing process: an application of the Kohonen algorithm to the traveling salesman problem, *Biol. Cybernetics*, vol. 59, pp. 33-40.
- Gendreau, M., Hertz A. and Laporte G., (1994), "A Tabu Search Heuristic for the Vehicle Routing Problem", *Management Science* 40,1276-1290.
- Ghaziri, H. (1991), " Solving routing problems by a self-organizing map", in *Artificial Neural Networks*, Vol.1, Kohonen T., et al. (eds), North-Holland,829-834.
- Gillett, B. E., and L.R. Miller (1974). A Heuristic Algorithm for the Vehicle Dispatch Problem, *Operations Research*, vol. 22, 340-347.
- Golden, B.L. and Assad A.A. , (1988), *Vehicle Routing: Methods and Studies*, North-Holland, Amsterdam.
- Hopfield, J.J., and D.W. Tank (1985), Neural computation of decisions in optimization problems, *Biological Cybernetics* 52,141-152.
- Kohonen, T. (1982), *Self-Organization and Associative Memory*, Springer-Verlag, Berlin.
- Mole, R. H. and S. R. Jameson , (1976), A Sequential Route Building Algorithm Employing a Generalised Savings Criterion, *Operations Research Quarterly*, vol. 27, pp. 503-512.
- Osman, I. (1993), "Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem", *Annals of Operations Research* 41,421-451.

**A Parallel Tabu Search Algorithm
Using Ejection Chains
for the Vehicle Routing Problem**

César Rego
Universidade Portucalense
Departamento de Informática
Rua Dr. Antonio Bernardino de Almeida 519-614
4200 Porto, Portugal
E-mail: Cesar.Rego@uporto.pt

Catherine Roucairol
Université de Versailles Saint-Quentin-en-Yvelines
Laboratoire PRISM
45, Avenue des Etats-Unis
78035 Versailles, France
E-mail: Catherine.Roucairol@prism.uvsq.fr

Abstract:

In this paper we describe a Parallel Tabu Search algorithm for the vehicle routing problem under capacity and distance restrictions. In the neighborhood search, the algorithm uses compound moves generated by an ejection chain process. Parallel processing is used to explore the solution space more extensively and different parallel techniques are used to accelerate the search process. Tests were carried out on a network of SUNSparc workstations and computational results for a set of benchmark problems prove the efficiency of the algorithm proposed.

Key Words: Ejection chains, parallel algorithms, tabu search, vehicle routing problem.

1. Introduction

The *Vehicle Routing Problem* (VRP) holds a central role in the fields of transportation, distribution and logistics (see, e.g., Bodin *et al.* 1983 and Christofides 1985). It 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. Consider a *depot* to be located at v_0 and let $V' = V \setminus \{v_0\}$ be used as the set of n *cities* or *customers*. A non-negative *cost* or *distance* matrix $C = (c_{ij})$ is associated with every arc. We assume that m identical vehicles each with capacity Q are used. Vehicles make *collections* or *deliveries* but not both. With each vertex v_i in V' is associated a quantity q_i of some goods to be delivered by a vehicle. The VRP thus consists of determining a set of m vehicle routes of minimal total cost, starting and ending at a depot, such that every vertex in V' is visited only once by one vehicle, and the total quantity assigned to each route does not exceed the capacity of the vehicle which services the route.

We will also consider an extension of this problem in which a service time δ_i is required by a vehicle to unload the quantity q_i at v_i . It is required that the total duration of any vehicle route (travel plus service times) may not surpass a given bound D , so in this context the cost c_{ij} is taken to be the travel times between the cities.

The VRP is a hard combinatorial problem and, to date, exact methods have seldom solved problem instances involving more than 50 vertices.

Most researchers have concentrated on the development of heuristic algorithms. Recent progress has been made in the area of local search techniques in particular *metaheuristics* such as *Simulated Annealing*, *Tabu Search*, *Genetic algorithms* and *Neural Networks*, which allow these techniques to overcome local optimality. A comprehensive description of these techniques can be found in Reeves (1993).

The purpose of this paper is to describe a parallel Tabu Search algorithm for the VRP defined above. Tabu search was proposed by Glover (1986) (see Glover and de Werra 1993 for an overview in recent Tabu Search applications). A number of algorithms based on this approach has already been applied to the VRP (see, Pureza and Franca 1991, Osman 1993, Taillard 1993, Gendreau, Hertz and Laporte 1994, and Rego and

Roucairol 1994). These algorithms differ essentially in the type of the neighborhood structures for defining moves leading from one solution to another. The basic moves consist of a simple insertion or exchange of vertices/arcs on the graph of the problem. One improvement on this procedure may be made by combining different types of moves at each step. This framework is based on the assumption that in several cases the best move may only be achieved if a certain number of intermediate steps are performed. This introduces the notion of compound moves.

In Osman (1993) a compound move consists of combining insertion moves and exchange moves, a vertex shift from one route to another and interchanging vertices between routes based on 2-opt procedures. Gendreau, Hertz and Laporte (1994) use an insertion followed by either 3-opt exchange or 4-opt exchange of arcs closest to the inserted vertex. More recently, Rego and Roucairol (1994) have considered compound moves based on the notion of *ejection chains*.

Ejection chains were proposed by Glover (1991, 1992) and appear to be novel techniques to explore the solution space in local search methods.

A generic interpretation consists of successively selecting a set of elements which are assigned to a new state, so that at each step the change of state is created by the element from the step immediately preceding, producing then a chain effect.

In the ejection chain terminology each of these steps is currently named a *level* and the change state of an element produced by another one is called an *ejection*.

The aim of ejection chain approaches is to produce restricted sets of moves which are the best among a vast (often exponential) number of alternatives, requiring an efficient polynomial effort.

These procedures are based on the principle that the choice of the elements being ejected at each level of the chain depends on the cumulative effect of the ejections in previous levels. This passing of information from one level to another leads to compound moves containing a better overview of the solution, thus making it possible to determine the degree of the modification that should be chosen (i.e. the level of the chain to

be selected to perform a move).

In an ejection chain construction, intermediate steps concern a sequence of ejection moves which are themselves not sufficient to transform one solution into another, but make such transformation possible when associated with appropriate trial moves.

A number of algorithms based on these approaches has already been applied with success on other combinatorial problems. Dorndorf and Pesch (1994) and Hubscher and Glover (1992) use these concepts for different forms of clustering problems. Also, Laguna *et al.* (1991) use an ejection chain strategy for multilevel generalized assignment problems. However, all these applications concern chains, based on the ejection of vertices. Recently, Dam and Zachariasen (1994), and Glover and Pesch (1995) have tested arc based ejection chain methods for the traveling salesman problem.

In this paper we show that ejection chain procedures in conjunction with parallel processing give a powerful Tabu Search algorithm for the VRP. In the algorithm, parallel processing is used to explore the solution space more extensively, as well as to accelerate the move evaluation on the ejection chain construction.

The remainder of this paper is organized as follows. In Section 2., the algorithm is described and computation results are presented in Section 3.. Section 4. contains a summary and concluding remarks.

2. Parallel Tabu Search Algorithm

The algorithm considers a node-ejection chain process proposed by Glover (1991) and defined by Rego and Roucairol (1994) for the VRP. In this section we only give definitions which are necessary for the algorithm presentation.

2.1 The Neighborhood Search Procedure

We assume that a partial graph $S = (V, X)$ associated with a solution is given.

For any vertex v_i , in a given orientation of a route, let v_{i-1} be its predecessor and v_{i+1} its successor. Consequently, we denote by

(v_{i-1}, v_i, v_{i+1}) a triplet representing two consecutive arcs (v_{i-1}, v_i) and (v_i, v_{i+1}) .

An ejection results by moving a vertex to a new position occupied by another vertex, disconnecting this vertex from its position.

Let k be a level of the chain, in an ejection chain each vertex v_i^k ejects the vertex v_i^{k+1} ending with the ejection of the v_b^l . As a result, triplets $(v_{i-1}^k, v_i^k, v_{i+1}^k)$ ($k = 0, 1, \dots, l$) are successively replaced by triplets $(v_{i-1}^k, v_i^{k-1}, v_{i+1}^k)$, ($k = 1, 2, \dots, l$). Because v_b^l vertex is not attached to any route, this transformation does not represent a complete transition to a new neighboring solution.

We denote by Γ_l the set of central vertices of every triplet, which are sufficient to identify l levels of the ejection chain. Also we denote by W_l the set of vertices in all triplets of the ejection chain.

To complete the transition from one solution to another two types of *trial moves* are used:

Type I trial move Inserting v_b^k between the last predecessor and successor of v_t^0 , creating arcs (v_{t-1}^0, v_b^k) and (v_b^k, v_{t+1}^0) .

Type II trial move Creating the arc (v_{t-1}^0, v_{t+1}^0) and choosing two vertices $v_p, v_q = v_{p+1} \notin \Gamma_k$. Then, inserting v_b^k between v_p and v_q , adding arcs $(v_p^k, v_b^k), (v_b^k, v_q^k)$ and deleting the arc (v_p, v_q) .

Figure 1 shows two examples of ejection chains transforming a given solution.

The diagram in the right-hand illustrates three levels of an ejection chain ended with Type I trial move and, in that (in this example) only one route is modified. In the left-hand diagram is showed an ejection chain of four levels applied on three routes and the number of routes is reduced by using the Type II trial move. The dotted horizontal lines represent arcs that have been removed from the previous routes and, the arcs that have been created by the move are shown by unbroken non-horizontal arcs.

In order to guarantee that the original costs are not modified during the ejection chain construction, it is still necessary to define a condition to prevent an arc from being inserted more than once. This condition is

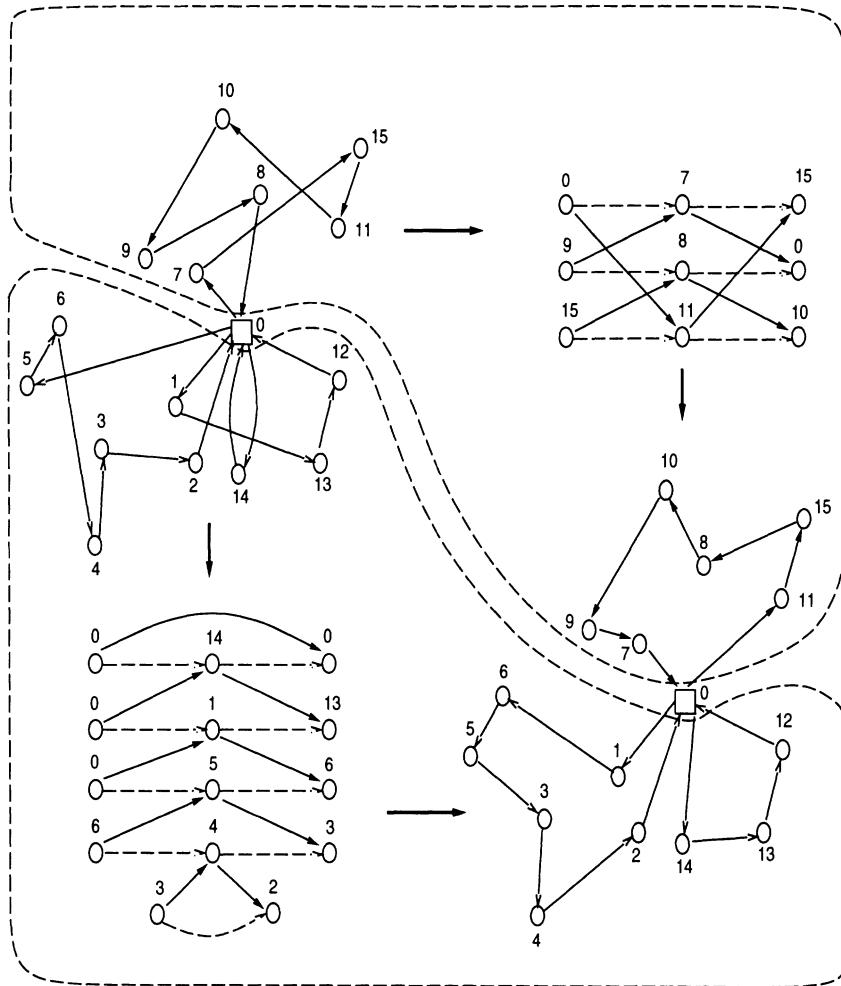


Figure 1: Example of ejection chains ended with Type I and Type II trial moves

imposed by a so-called *legitimacy restriction* stipulating that each vertex in Γ_l occurs only once in W_l . Moreover, any predecessor of a vertex in Γ_l may reappear as a successor of a vertex in Γ_l and vice versa, without violating this restriction. However, an exception is made for arcs linked to the depot which can reappear several times but only connected to vertices in Γ_l , that is, vertex v_0 cannot be ejected.

Now, it is clear that the maximum length of an ejection chain is li-

mited by the legitimacy restriction and depends on the number of times that vertex v_0 appears in the chain. Let r be the number of routes in a given solution, then the maximum number of levels of a legitimate ejection chain is bounded by $O((n+r)/2)$. However, it is not usually necessary to perform all possible levels of the chain to obtain the best move; therefore it may be limited by a parameter which we denote by l_{max} .

For any vertex v in V' , we define its h -neighborhood $N_h(v)$ as the set of the h vertices closest to v , and consequently we define its *legitimate neighborhood* as the subset of these vertices that do not violate the legitimacy restriction. Thus, for a given level k of an ejection chain, we define the *legitimate neighborhood* of a vertex v_i^k as the set:

$$LN_h(v_i^k) = \{v_j \mid v_j \in N_h(v_i^k), v_j \notin W_{k-1}\}.$$

For convenience, we will denote the $c_{ij}s$ values by $c(v_i, v_j)$ and consequently, let us define the cost of a triplet as the cost sum of its arcs.

The neighborhood search procedure may now be described as in Figure 2. In order to avoid cycling, tabu restrictions are taken into account throughout the ejection chain construction. The tabu list is made up of attributes (v_i, v_j) representing ejection moves and only ejections (v_t^0, v_i^1) and $(v_b^{k^*}, v_j^{k^*-1})$ are put into the tabu list. Also, the tabu list size θ is an integer value randomly chosen within an interval $[\underline{\theta}, \bar{\theta}]$.

Nevertheless, if in a given instant an ejection move is declared to be tabu, this status can be disregarded if the corresponding trial move improves the best solution found so far, which defines the classical *aspiration function* used in Tabu search.

In addition to the short term memory specified by the tabu restrictions and the aspiration criterion, the algorithm makes use of intermediate and long term memory respectively to implement the intensification and diversification strategies. Also, oscillation strategies which exploit the moves' characteristics are considered allowing the acceptance of temporarily non-feasible solutions. For more details on such search strategies and other algorithmic considerations we refer readers to the sequential version of the algorithm described in Rego and Roucairol (1994).

PROCEDURE *EjectionChain* (*solution*: S , *number of levels*: l_{max})

- Initialize legitimate neighborhood for all vertices and set $k = 0$.
- Determine the best ejection move between two vertices v_i and v_j by calculating:

$$\begin{aligned} e_{k+1} &= \min_{v_i, v_j \in V'} \{c(v_{j-1}^k, v_i^{k-1}, v_{j+1}^k) - c(v_{i-1}^{k-1}, v_i^{k-1}, v_{i+1}^{k-1}) \\ &\quad - c(v_{j-1}^k, v_j^k, v_{j+1}^k) + \lambda c(v_{i-1}^{k-1}, v_{i+1}^{k-1})\}, \end{aligned}$$

where $\lambda = 1$ if Type II trial move is used and $\lambda = 0$ otherwise.

Set $v_t^k = v_i$, $v_b^{k+1} = v_j$.

- Set $\Gamma_k = v_t^k$.
- While $LN_h(v_b^{k+1}) \neq \emptyset$ and $k < l_{max}$ do
 - Set $k = k + 1$ and set $\Gamma_k = \Gamma_{k-1} \cup \{v_b^k\}$.
 - Inspect the cost changes by calculating the corresponding trial value:

$$\Delta_k = \begin{cases} e_k + c(v_{i-1}^0, v_j^k, v_{i+1}^0), & \text{if Type I,} \\ e_k + \min_{v_p, v_q \in V \setminus \Gamma_k, v_j \in \Gamma_k} \{c(v_p^k, v_j^k, v_q^k) - c(v_p^k, v_q^k)\}, & \text{if Type II.} \end{cases}$$

- Update the best level k^* so far that yielded the minimum trial value.
If using the type II trial move then record the associated v_p^k vertex.
- Determine a new vertex $v_{b^*} \in LN_h(v_b^k)$ by calculating:

$$e_k = e_{k-1} + \min \{c(v_{j-1}^k, v_i^{k-1}, v_{j+1}^k) - c(v_{j-1}^k, v_j^k, v_{j+1}^k)\},$$

and set $v_b^{k+1} = v_{b^*}$.

- Set $l = k^*$ and update the current solution according to Type I or Type II trial moves.
- Update the legitimate neighborhoods for every vertex $v \in W_{k^*}$.

Figure 2: The neighborhood search procedure

2.2 The Parallel Algorithm

Our algorithm was implemented on a network of SUNSparc workstations using the PVM (Parallel Virtual Machine) system, which permits this network to be viewed as a single parallel computer. The unit of pa-

rallelism in PVM is a task, and (as for a Unix process) multiple tasks may execute on a single processor. A standard model of message passing is used to allow communication and synchronization between tasks.

We use a classic *master-slave* model without communication among the slave processes. Each slave executes a complete tabu search algorithm with a different set of parameters, but from the same starting solution which is given by the Clark and Wright (1964) algorithm. Basically, the work of the master is to collect the best local solution from each slave process and transmit the best among them to slaves, to start up a new iteration of the method. Furthermore, the communication and synchronization processes are controlled by the master process. In order to minimize the communication times, only solution values are transmitted to the master and, only slaves that did not find the best value receive the corresponding solution.

In our PVM implementation, the algorithm makes use of the following messages for handling the search and the communication process.

- **MAKE_SEARCH** to a slave process to perform a search.
- **END_SEARCH** to a slave process to stop the search.
- **ELECTED** to indicate that a slave process has found the best global solution.
- **NOT_ELECTED** to indicate that a slave process has not found the best global solution.

The **MASTER** and **SLAVE** algorithms are described respectively in Figures 3 and 4.

The evaluation of a trial move in a given level of the ejection chain depends on the vertices which were ejected up to the level in question. However, the choice of the ejection move is independent of the trial move evaluation throughout the ejection chain. This property allows these two operations to be performed in parallel, which is important when the trial move evaluation demands a considerable effort. This is the case of the Type II trial move in which at each level of the chain the best insertion of the ejected vertex would be chosen after $O(n)$ comparisons, which is also the effort necessary to determine each ejection move. Thus, when the Type II trial move is active in the *EjectionChain* procedure, the evaluation of each trial solution is kept for another process. This type of

MASTER algorithm

- Start up slaves tasks.
- Read problem instance and broadcast it to slave tasks.
- Read starting solution.
- Initialize the set of *not elected slaves* as all slave tasks.
- While *an improvement is found and the number of restarts are not met* do
 - Broadcast MAKE_SEARCH message to all slave tasks.
 - Send the current best solution to *not elected slaves*.
 - Wait for best local solution values from all slave tasks.
 - Identify the new best global solution.
 - If it is a global improvement then
 - * Update the best global solution from one of the slaves that has found it, and send an ELECTED message to it asking for the corresponding solution.
 - * Identify the new set of *not elected slaves* and send a NOT_ELECTED message to them.
 - * Receive the new best global solution.
 - Otherwise broadcast a NOT_ELECTED message to all slave tasks.
 - Update the number of restarts
- Broadcast an END_SEARCH message to all slave processes
- Record the best global solution found by the algorithm and terminate the search.

Figure 3: The master algorithm

parallelization where each process performs a different function is often called *functional parallelism*.

In addition, the algorithm includes a post-optimization procedure, which consists of a local reoptimization of every route by solving the corresponding traveling salesman problem. The parallelization method where all processes are the same, but each one only knows and solves a small part of the data is called *data parallelism*. In the post-optimization procedure, a data parallelism model is used in which each individual route is assigned to a different process and reoptimized separately by us-

SLAVE algorithm

- Receive message from MASTER.
- While message is equal to MAKE_SEARCH do
 - Identify parameters according to its own slave *task identification* and perform a tabu search algorithm using the *EjectionChain* procedure.
 - Send best solution value to MASTER
 - Receive message to know if its solution was selected
 - If message is equal to NOT_ELECTED
 - * Receive message from MASTER for continuing or terminating the search
 - * If message is equal to MAKE_SEARCH
 - Receive best global solution from MASTER
 - If message is equal to ELECTED
 - * Send best local solution to MASTER
 - * Receive message for continuing or terminating the search

Figure 4: The slave algorithm

ing ejection chains with Type I trial moves.

3. Computational Results

The performance of our algorithm was tested on a set of fourteen benchmark problems from Christofides Mingozi and Toth (1979). Each problem has different characteristics and may include capacity and route length constraints. All tests were performed on a network of four SUN-Sparc 4 IPC workstations. The code was written in C and parallelism was supported by using PVM library routines. Our solution values were calculated with real distances and computational results for the sequential (SEQ) and parallel (PAR) versions of our algorithm are reported in Table 1. Bold characters correspond to the best known solutions and asterisks indicate values which had already been proved to be optimal.

We can see that the parallel version never finds solutions worse than the sequential one, and improves seven of these solutions. As a result the PAR algorithm gives solutions which are on average 0.22% better than the SEQ algorithm, which is very significant when we consider that the best known solutions are already optimal or very close to this value. In addition, comparisons were made between our algorithms and the best

Table 1: Characteristics of test problems, solution values and relative percentage deviation (RPD) for both sequential (SEQ) and parallel (PAR) versions of the algorithm.

Prob.	<i>n</i>	<i>Q</i>	<i>D</i>	δ	Best published	Solution values		RPD	
						SEQ	PAR	SEQ	PAR
C1	50	160	∞	0	*524.61^a	524.61	524.61	0.00	0.00
C2	75	140	∞	0	835.26^b	837.50	835.32	0.27	0.01
C3	100	200	∞	0	826.14^b	827.53	827.53	0.17	0.17
C4	150	200	∞	0	1028.42^b	1054.29	1044.35	2.52	1.55
C5	199	200	∞	0	1298.79^b	1338.49	1334.55	3.06	2.75
C6	50	160	200	10	555.43^b	555.43	555.43	0.00	0.00
C7	75	140	160	10	909.68^b	909.68	909.68	0.00	0.00
C8	100	200	230	10	865.94^b	868.29	866.75	0.27	0.09
C9	150	200	200	10	1162.55^b	1178.84	1164.12	1.40	0.14
C10	199	200	200	10	1397.94^b	1420.84	1420.84	1.64	1.66
C11	120	200	∞	0	1042.11^b	1043.54	1042.11	0.14	0.00
C12	100	200	∞	0	*819.56^c	819.56	819.56	0.00	0.00
C13	120	200	720	50	1541.14^b	1550.17	1550.17	0.59	0.59
C14	100	200	1040	90	866.37^b	866.53	866.37	0.02	0.00
Average								0.72	0.50

References in which the solution were obtained:

^a Christofides, Hadjiconstantinou and Mingozi (1993).

^b Taillard (1993).

^c Fisher (1994).

tabu search algorithms for the VRP, whose computation time for finding their best solutions has been published. Computational results are reported in Table 2. The objective is to compare these algorithms in terms of solution quality and computation time.

We can see that for some instances the PAR algorithm takes more time than the SEQ one, but this is compensated for by the quality of solutions.

Although different machines were used it is clear that the SEQ algorithm can compete with the others (which do not use parallel computing either), and that the PAR algorithm may be advantageously compared to all these algorithms.

Table 2: Computation times (in seconds) and relative percentage deviation (RPD).

Algorithm Computer Problem	OTS		GHL		SEQ		PAR	
	VAX 8600		Silicon 4D/35		SUNsparc 4		4 SUNsparc 4	
	CPU	RPD	CPU	RPD	CPU	RPD	CPU	RPD
C1	61	0.00	84	0.00	51	0.00	63	0.00
C2	50	1.05	2352	0.06	1008	0.27	2603	0.01
C3	895	1.07	408	0.40	2034	0.17	1579	0.17
C4	1761	1.55	3270	0.00	1632	2.52	2908	1.55
C5	1704	2.75	5028	0.75	975	3.06	4624	2.75
C6	63	0.00	468	1.84	190	0.00	143	0.00
C7	745	0.15	1908	0.00	1386	0.00	1234	0.00
C8	1965	0.09	354	0.39	516	0.27	1136	0.09
C9	2475	1.85	1278	0.00	933	1.40	1791	0.14
C10	4025	1.42	2646	1.31	3121	1.64	2563	1.66
C11	780	0.00	714	1.47	378	0.14	674	0.00
C12	340	0.00	102	3.01	73	0.00	94	0.00
C13	1576	0.38	2088	0.00	120	0.59	117	0.59
C14	582	0.00	1782	2.12	565	0.02	1479	0.00
Average	1216	0.74	1606	0.81	512	0.72	1501	0.50

OTS: Osman's (1993) tabu search algorithm.

GHL: the Gendreau, Hertz and Laporte (1994) tabu search algorithm.

4. Conclusions

We have described and tested a parallel tabu search algorithm for the VRP. In the neighborhood search the algorithm uses a new concept of creating compound moves based on ejection chain processes, which have already shown to be efficient for the sequential version of the algorithm.

The parallel implementation was based on a synchronous model and different levels of parallelization were used. Experiments showed that search strategies based on different parameter settings make it possible to explore the solution space more extensively and to find better solutions. We have noticed that the process which finds the best global solution usually changes at each point of synchronization, hence it reflects a dynamic adjustment of parameters which are the most adequate to improve the best global solution at each step.

Indeed, two usual parallelization techniques were used in order to accelerate the search process. The time per iteration was reduced by

half, using a *functional parallelism*, which separated the evaluations of an ejection move and the associated trial move into two independent operations. Furthermore, a *data parallelism* method, which assigns each route to a distinct process, accelerated the post-optimization phase to the maximum time required for the reoptimization of a route.

5. References

- L. Bodin, B. Golden, A. Assad, and M. Ball, Routing and scheduling of vehicles and crews: the state of the art, *Computers and Operations Research*, 10 (1983) 63.
- N. Christofides, The Traveling Salesman Problem in: *A Guide Tour of Combinatorial Optimisation* ed. E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan and D. B. Shmoys (Wiley, Chichester, 1985) p. 431.
- N. Christofides, E. Hadjiconstantinou, and A. Mingozzi, A new exact algorithm for the vehicle routing problem based on q-path and k-shortest path relaxations, Working paper, The Management School, Imperial College, London, (1993).
- N. Christofides, A. Mingozzi, and P. Toth, *The Vehicle Routing Problem in: Combinatorial Optimisation*, ed. N. Christofides, A. Mingozzi, P. Toth and C. Sandi,(Wiley, Chichester, 1979) p. 315.
- G. Clark and J. W. Wright, Scheduling of vehicles from a central depot to a number of delivery points, *Operations Research*, 12 (1964) 568.
- M. Dam and M. Zachariasen, *Tabu Search on the Geometric Traveling Salesman Problem*, M. S. Thesis, Department of Computer Science, University of Copenhagen, (1994).
- U. Dorndorf and E. Pesch, Fast clustering algorithms, ORSA Journal on Computing, 6 (1994) 141.
- M. L. Fisher, Optimal solution of vehicle routing problems using minimum k-trees, *Operations Research*, 42 (1994) 626.
- M. Gendreau, A. Hertz, and G. Laporte, Tabu search heuristic for the vehicle routing problem, *Management Science*, 40 (1994) 1276.
- F. Glover, Future paths for integer programming and links to artificial intelligence, *Computers and Operations Research*, 5 (1986) 533.
- F. Glover, Multilevel tabu search and embedded search neighborhoods for the traveling salesman problem, Working paper, University of Colorado at Boulder,Graduate School of Business and Administration (1991).

- F. Glover, New ejection chain and alternating path methods for traveling salesman problems, *Computer Science and Operations Research*, (1992) 449.
- F. Glover and D. de Werra, *Tabu Search, Annals of Operations Research*, (Baltzer Science Publishers, 1993) 41.
- F. Glover and E. Pesch, TSP Ejection chains. Working paper, University of Colorado at Boulder, Graduate School of Business and Administration (1995).
- R. Hubscher and F. Glover, Ejection chain methods and tabu search for clustering, Working paper, Graduate School of Business and Administration, University of Colorado at Boulder (1992).
- M. Laguna, J. P. Kelly, J. L. Gonzalez-Valarde, and F. Glover, Tabu search for multilevel generalized assignment problems, Working paper, Graduate School of Business and Administration, University of Colorado at Boulder (1991).
- I. H. Osman, Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem in: *Annals of Operations Research*, (Baltzer Science Publishers, 1993) 41 p. 421.
- V. M. Pureza and P. M. França, Vehicle routing problems via tabu search metaheuristic, Working paper CRT-747, Centre de Recherche sur les Transports, Université de Montréal (1991).
- C. R. Reeves, *Modern Heuristic Techniques for Combinatorial Optimisation*, (Blackwell, Oxford, 1993).
- C. Rego and C. Roucairol, An efficient implementation of ejection chain procedures for the vehicle routing problem, Working paper, 44, Laboratoire PRISM, Université de Versailles, (1994).
- E. Taillard, Parallel iterative search methods for vehicle routing problems, *Networks*, 23 (1993) 661.

Acknowledgements

The research of the first author was partially supported by the Junta Nacional de Investigação Científica e Tecnologica (JNICT), Portugal.

Fast Local Search Algorithms for the Handicapped Persons Transportation Problem

Paolo Toth, Daniele Vigo

*Dipartimento di Elettronica, Informatica e Sistemistica
Università di Bologna*

Viale Risorgimento, 2 – 40136 Bologna – Italy

E-mail: ptoth@deis.unibo.it – dvigo@deis.unibo.it

Abstract:

We examine the problem of determining an optimal schedule for a fleet of vehicles used to cover the trips required to transport handicapped persons in an urban area. The problem is a generalization of the well-known Pickup and Delivery Problem with Time Windows since, due to the high level of service required by this kind of transport, many additional operational constraints must be considered. We describe fast and effective local search refining procedures, based on trip movements and exchanges, which can be used to improve within acceptable computing time the solutions of large-size instances obtained by using a parallel insertion heuristic algorithm. The procedures have been successfully applied to a set of large real-life problems for the town of Bologna, involving about 300 trips each, showing the effectiveness of the proposed approach.

Key Words: Heuristic algorithms, local search, pickup and delivery, scheduling, transportation.

1. Introduction

We consider the Handicapped persons Transportation Problem (HTP), i.e., the problem of determining an optimal schedule for

a fleet of vehicles used to cover the regular transport requests of handicapped persons, hereafter also called *users*, in an urban area. This problem is a generalization of the Pickup and Delivery Problem with Time Windows (PDPTW), which calls for the determination of a min-cost set of routes for vehicles with given capacity, satisfying a set of n transportation requests, each requiring pickup at a given origin and delivery at a given destination, within given time windows.

Each transport request for a given user, also called *trip*, is characterized by an origin location, where the user must be picked up, and a destination location, where the user must be delivered. Origin and destination locations of a given user are also called *stops*. Pickup and delivery instants must occur within given time windows depending on the user. If a vehicle arrives at a stop too early, it must wait until the beginning of the relative time window before starting the service at the stop. The total time spent on the vehicle by each user must not exceed a prescribed limit which, in general, is proportional to the time needed to travel directly from origin to destination locations. The service requirements of each user depend on his physical characteristics (for example, he may either be able to walk, or require a wheelchair). In particular, non-walking users who require to remain seated in the wheelchair can be transported only by vehicles equipped with a wheelchair elevator. Analogously, users who require the assistance of specialized personnel can be assigned only to vehicles with specialized crew.

A mixed fleet of vehicles is available for the service, composed of a limited number of minibuses and special cars, and a (practically) unlimited number of taxis. The vehicles belong to different agencies (municipality, hospitals, private associations and private companies) and are located at different depots. A vehicle can be *depot-based*, hence it starts its route from an *origin depot* and it must return to a, possibly different, *destination depot* at the end of the route. There exist also *non-depot-based* vehicles (like, for example, taxis) for which the route begins at the pickup location of the first served trip and ends at the last delivery location, i.e., initial and final empty vehicle travels are not considered from the cost point of view. Each vehicle is characterized by its load capacity in persons and wheelchairs, by the load/unload times, and by

other optional features such as the presence on board of specialized personnel or of a wheelchair elevator. Routing costs, depending on distance traveled and time, are associated with each vehicle used for service.

In general, taxis can be more flexible and less expensive than minibuses for the service of isolated and non "difficult" requests (i.e., users who do not need to remain seated in the wheelchair or do not require specialized personnel), but it is highly desirable to use them only if strictly necessary. This is justified both by economical considerations, and by the fact that, being non-dedicated vehicles with non-specialized personnel on board, taxis are less suited for the transport of handicapped persons. Thus, with each vehicle is associated a penalty (high for taxis) which is incurred if the vehicle is used.

The desired level of service from the user point of view can be obtained by imposing appropriate time window and maximum travel time constraints. Moreover, high routing and fixed costs can be imposed in order to reduce the number of used taxis. Hence, the main problem objective is to minimize the total cost of the service.

The problem is NP-hard in the strong sense since it is a generalization of PDPTW. For complete surveys on PDPTW and other related problems see, e.g., Bodin *et al.* (1983), Solomon and Desrosiers (1988), and Dumas, Desrosiers and Soumis (1991).

Exact approaches for the solution of real-life handicapped transport problems (typically with hundreds of trips) are, in general, not practicable. A heuristic algorithm for the solution of a disabled persons transportation problem having characteristics similar to those mentioned above, has been proposed by Roy *et al.* (1984). Sutcliffe and Board (1990) addressed a special case of HTP where all users have the same destination and only the vehicle capacity and total tour time constraints are considered, thus reducing HTP to a Vehicle Routing Problem (VRP). Using an algorithm for the solution of VRP they have been able to solve a real-life problem with 38 trips. Toth and Vigo (1993) proposed a parallel insertion heuristic algorithm that iteratively assigns unrouted trips to the routes, solving at each iteration an Assignment Problem on a cost matrix obtained

by using a modified cheapest insertion criterion based on locally optimal choices. This constructive algorithm proved to be able to quickly determine on a Personal Computer (PC) good solutions with respect to the hand-made ones, both in terms of number of used taxis and overall routing cost. Moreover, the heuristic solutions satisfy all the time window and maximum travel time constraints, while the hand-made ones violate many of them, as illustrated in Section 3. However, when facing large and strongly constrained instances (i.e., instances with tight time windows and small average maximum travel time) as those examined in Section 3, this heuristic tends to produce several short routes (typically served by taxis). In this paper we describe fast and effective local search refining procedures, which are generally able to reduce considerably both the number of used taxis and the overall cost.

In practical applications (as that described in Section 3) the heuristic algorithms for the solution of HTP are often incorporated within an interactive Decision Support System, and several alternative solutions must be obtained and compared in order to determine a satisfactory one. Then it is highly desirable that the computing time spent for each single run of the heuristic is limited to a few minutes on a PC, even for instances with hundreds of trips. In this case the use of standard local search procedures or of metaheuristic techniques could be very expensive from the computational point of view (see, e.g., Solomon, Baker and Schaffer (1988), Thompson and Psaraftis (1989) and Osman (1993) for an application to related problems). Thus, we have used restricted neighborhoods and a simple descent procedure which, in smaller computing time, allowed for a considerable improvement of the solutions obtained by using the insertion algorithm proposed by Toth and Vigo (1993). The proposed procedures have been tested on a set of five real-life problems for the town of Bologna involving about 300 trips each.

2. Fast local search refining procedures

In this section we describe refining procedures that can be applied to improve the solution of large-size HTP instances obtained through a constructive algorithm. The procedures use different neighborhoods obtained by considering trip movements and exchanges, both within the same route (*intra-route*) and between different routes (*inter-route*). We next give a description of the neighborhoods used and

of the overall refining procedure we have implemented.

2.1 Intra-route movements

A first simple local search procedure attempts to improve a given route by modifying the sequence of the pickup and delivery stops along the route. The modification is obtained by moving a single stop to a different position of the route, while preserving route feasibility. All feasible movements of the stops to a different position are evaluated, and the best movement among all those producing a positive cost reduction (also called *active movements*), if any, is performed.

The number of movements to be evaluated can be considerably reduced. Indeed, it is evident that a pickup stop cannot be moved to a position which is after the corresponding delivery and, analogously, a delivery stop cannot be moved before the corresponding pickup. Moreover, movements which cause the violation of some time window, maximum travel time or capacity constraints, i.e., which produce infeasible routes starting from feasible ones, are not considered.

Let n be the total number of trips of an instance and let \bar{n} denote the maximum number of trips of a route. For each route, the procedure requires $O(\bar{n}^3)$ time. In fact each stop can be inserted in $O(\bar{n})$ different positions, while the feasibility and the cost of a given sequence of stops can be computed in $O(\bar{n})$ time. Note that in practical situations the value of \bar{n} is generally not greater than 10, thus the cost of the intra-route movements can be computed very quickly. A single iteration of the above procedure considers all the routes. The corresponding overall time complexity is clearly $O(n\bar{n}^2)$. The procedure is iterated until no active movement is found.

A more effective, but also more time consuming, intra-route procedure can be obtained by removing each trip from its current route, and inserting its two stops in the best feasible positions within the same route. This is a special case of the inter-route movement of type (a) described in the next section.

2.2 Inter-route movements

The inter-route movements considered require the removal of a trip from its current route and the insertion of the pickup and delivery stops of the trip in the best feasible positions of a possibly different route. In particular, given a trip i we consider the following movements:

- a) *Trip insertion*: removal of the given trip i from its route and insertion of the two stops of i in the best positions of a, possibly different, route.
- b) *Trip exchange*: swap between the given trip i and a trip j belonging to a different route. Both trips are removed from their current route, and their associated stops are inserted in the best positions along the new routes.
- c) *Trip double insertion*: removal of the given trip i from its route and insertion of i in the best positions of a different route. Then a trip j is removed from a third route and its stops are inserted in the best feasible positions along the route initially containing trip i .

In each iteration of our procedure we consider, in turn, each trip i and perform the best, in term of global cost saving, movement among all those belonging to neighborhoods (a), (b) and (c) above involving trip i . In this way, within an iteration at most n movements (one for each trip) are performed. Note that the best movement of type (a) could leave trip i in its current position, with no change in the current solution. Note also that in the evaluation of movements of type (a) and (c), if the removal of a trip leaves its current route empty, the corresponding global cost saving takes into account also the reduction of the number of performed routes.

The time complexity of an iteration of the above inter-route procedure is $O(n^2\bar{n}^3)$. Indeed, for each trip all movements of type (a) can be evaluated in $O(n\bar{n}^2)$ time, while the evaluation of all movements of types (b) and (c) require $O(n\bar{n}^3)$ time.

2.3 Overall refining procedure

Intra-route and inter-route procedures are combined together to obtain an overall iterative refining procedure. At each iteration we

first apply to each route the intra-route procedure until no active exchange is found. Then we apply the inter-route procedure performing for each trip the best movement found (possibly leaving it in the same position). The overall procedure is stopped after a given number of iterations or when no improvement is obtained during a complete iteration.

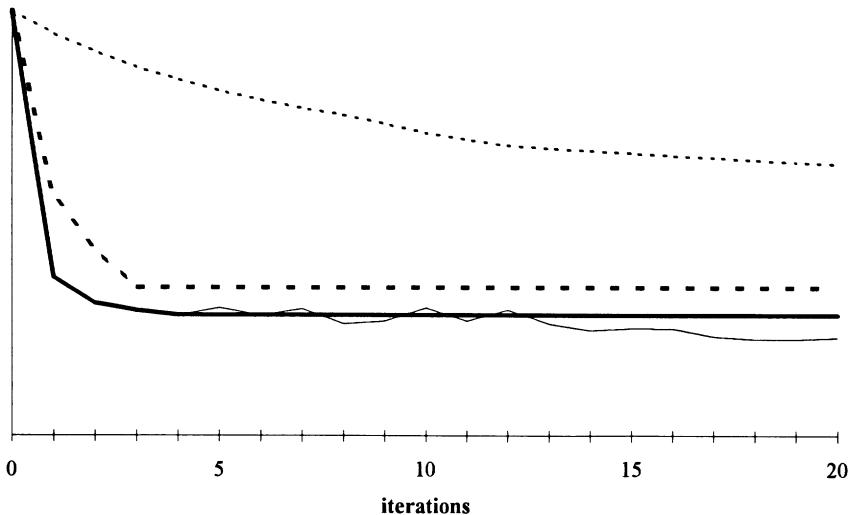
Movements of type (a) and (b) are widely used in local search procedures, while the use of the trip double insertion (c) is less usual. Nevertheless, this type of movement proved to be very effective within our descent procedure, being, on average, the selected movement in 40% of the cases, while movements (a) and (b) were selected in 17% and 43% of the cases, respectively. Figure 1 further illustrates this, showing, for the Monday instance of Section 3, the percentage cost reductions obtained at the end of each iteration by applying: i) a “traditional” local search procedure (where at each iteration the best movement of type (a) and (b), over all trips, is performed); ii) the above procedure using only movements of type (a) and (b); iii) the above procedure using all movement types. Obviously, if metaheuristic search procedures are used to escape from local minima, the importance of movements of type (c) is reduced.

Since, as we noted in the introduction, we want to limit to a few minutes the time needed to obtain a solution, we perform a small number of iterations, say 2 or 3, of the overall procedure. In this case normally the procedure does not remain trapped into a local minimum, even if this obviously happens with a greater number of iterations. When an higher time can be spent in the refining step, metaheuristic techniques could be used in order to reduce this inconvenience. We examined this possibility in a simple way by adding to the cost saving of each movement a small random perturbation, whenever the cost reduction obtained in the last iteration is smaller than a given threshold, say r . As shown in Figure 1, this allowed for the determination of better solutions, but required an higher computing time.

3. Computational experiments

In this section we discuss the results obtained by using the local search procedure of Section 2 to improve the solutions obtained by the parallel insertion algorithm (called TV hereafter) when solving

Figure 1: Comparison of different local search procedures applied to the Monday instance. The behaviour of the traditional procedure is indicated by thin dashed line; our procedure using only movements of type (a) and (b) and all three types of movements is indicated by thick dashed and thick solid line, respectively; our procedure with a perturbation added when $r = 0.1\%$ is indicated by a thin solid line.



a set of five large real-life instances of the problem. These instances consist of the service covered by the Municipality of Bologna in 1994 during a week, from Monday to Friday, for which complete numerical input data as well as the hand-made solutions are available. The Municipality of Bologna is currently using algorithm TV for the solution of HTP.

Origin and destination locations of the users are dispersed over 290 physical zones covering the whole territory of the Municipality of Bologna. Asymmetric matrices containing the distance and the

travel time between all the physical locations are available. The problems consist of a number of trips between 276 and 312, whose service requirements are summarized in Table 1. The table reports, for each instance, the number and the percentage of users who require the wheelchair, who must remain seated in the wheelchair, and who need accompanying personnel. The last two columns report the totals of the data over all the week. By observing Table 1 it can be noted that more than 60% of the trips to be served are associated with “difficult” users. Indeed, the percentage of the non-walking users varies between 13.5% and 15.6%, with an average of 14.4%; the percentage of users who must remain seated in the wheelchair is between 3.8% and 4.3%, with an average of 4%; and the percentage of trips requiring the presence on the vehicle of specialized accompanying personnel is between 45.1% and 49.3%, with an average of 46.3%.

Table 1: Trip service requirements for all instances

non walk. seated	accomp. person.	Monday		Tuesday		Wednesday		Thursday		Friday		Total		
		# tr.	%	# tr.	%	# tr.	%	# tr.	%	# tr.	%	# tr.	%	
NO	-	NO	142	46.9	117	42.4	144	46.2	141	45.7	136	45.9	680	45.4
NO	-	YES	120	39.6	116	42.1	123	39.4	122	39.5	120	40.5	601	40.2
YES	NO	NO	21	6.9	21	7.6	25	8.0	25	8.1	21	7.1	113	7.6
YES	NO	YES	8	2.6	10	3.6	8	2.6	9	2.9	7	2.4	42	2.8
YES	YES	NO	2	0.7	2	0.7	2	0.6	2	0.6	2	0.7	10	0.7
YES	YES	YES	10	3.3	10	3.6	10	3.2	10	3.2	10	3.4	50	3.3
Total		303		276		312		309		296		1496		

For the great majority of the trips only one of the two time windows is tight. For example, in a morning *outward trip* from home to school, only the destination time window is tight, while the origin time window is tight for the *return trip* from school to home. Operational constraints are considerably tight. Indeed, in all instances the width of the tight time windows is 10 minutes, and the average maximum travel time over all trips is equal to 59 minutes.

The characteristics of the available vehicles are summarized in Table 2. In total two minibuses are owned by the Municipality, while the others cars and minibuses are owned by user associations and hospitals.

The available hand-made solutions, whose characteristics are

Table 2: Characteristics of the available vehicles.

type	owner	q.ty	capacity					costs in ITL	
			op.	wh.	norm.	elev.	pers.	Km.	hour
minibus	public	2	2	9	YES	YES		650	20000
minibus	priv. ass.	1	1	8	YES	YES		650	20000
minibus	priv. ass.	1	2	7	YES	YES		650	20000
minibus	priv. ass.	1	1	9	YES	YES		650	20000
minibus	priv. ass.	2	0	7	NO	YES		650	20000
minibus	priv. ass.	1	0	8	NO	YES		650	20000
minibus	priv. ass.	10	0	9	NO	YES		650	20000
minibus	priv. ass.	1	0	12	NO	YES		650	20000
car	priv. ass.	2	0	2	NO	NO		650	0
taxi	private	∞	0	2	NO	NO		1350	0

reported in Tables 3 and 4, do not satisfy all the constraints. For example, the analysis of the hand-made vehicle routes for the Monday instance shows that 186 out of 303 time windows are violated (154 of which by more than 5 minutes), 87 maximum travel times are exceeded (46 of which by more than 20 minutes) and 4 used taxis are overloaded, carrying 3 passengers at a time. Similar violations can be detected in all the other instances.

Algorithm TV and the refining procedures of Section 2 have been coded in FORTRAN and tested on an IBM Personal Computer 486/66. Tables 3 and 4 reports the comparison between the hand-made solution, and the results obtained by algorithm TV, with and without the use of the refining procedure of Section 2. The refining procedure has been applied to the solution obtained by algorithm TV, executing a maximum of 5 iterations. For each solution the tables report:

```

# vehic :    number of used vehicles;
# routes :   number of routes performed by the vehicles;
serv trips:  number of served trips;
Avg pass :  average number of passengers for each route;
Tot trav t : total travel time of all the routes (in hours);
Avg trav t : average travel time for each trip (in minutes);

```

Km total : total distance traveled by the vehicles (in Km);
 Km empty : total distance traveled by empty vehicles (in Km);
 Cost : global cost of the solution (in 10^3 ITL)
 CPU time : computing time (in IBM 486/66 seconds)

For the solutions obtained by algorithm TV and by the refining procedure (indicated as "TV + refining") the tables report the percentage difference of each of the above parameters with respect to the hand-made solution. The data relative to the taxi routes are also reported, since a significant part of the total cost of the hand-made solution is due to the large number of taxis used.

The solutions obtained in less than one minute by algorithm TV satisfy all the imposed operational constraints and are also much better than the hand-made ones. Indeed, the overall cost, the total travel time, the total distance traveled and the distance traveled with empty vehicles have been reduced, on average, by 20%, 11%, 29%, and 57%, respectively. The number of used taxis has been drastically limited, the number of performed routes has been decreased by 31%, and vehicle productivity has been substantially increased, almost doubling the average number of trips served by each route. Moreover, the average travel time for each user has been decreased by 22%. These solutions are substantially improved, in almost 3 minutes each, by the refining procedure, which greatly increased the reduction of the overall cost (32%), and of the total traveled time (23%). Consistent improvements have also been obtained with respect to the number of used vehicles, the total traveled distance, and the average number of trips served by a route. Finally, the good level of service of the solutions of algorithm TV has been preserved, since the average travel time for each trip is unchanged.

4. Conclusions

In this paper we have presented fast and effective refining procedures that can be used to improve the schedules for a mixed fleet of vehicles used to transport handicapped persons in an urban area. The problem is NP-hard in the strong sense and is characterized by several peculiar operational constraints due to the high level of service required by this kind of transport. Because of the large size of the instances arising in real situations (characterized by hundreds of

Table 3: Solutions obtained for the Monday instance

	Hand-made taxis		Algorithm TV taxis			TV + refining taxis		
	Tot.		Total	TV	diff.	Total	TV	diff.
# vehic	49	70	36	57	-19%	29	50	-29%
# routes	49	105	36	78	-26%	29	71	-32%
serv trips	67	303	79	303	—	58	303	—
Avg pass	1.4	2.9	2.2	3.9	+35%	2.0	4.3	+48%
Tot trav t	71.1	181.5	46.5	168.0	-7%	33.3	151.4	-17%
Avg trav t	32	52	36	39	-25%	36	40	-23%
Km total	1404	3026	771	2277	-25%	529	2060	-32%
Km empty	1012	1843	255	766	-58%	167	700	-62%
Cost	1825	4652	1002	3782	-19%	687	3331	-28%
CPU time				44.1			194.6	

Table 4: Summary of the solutions obtained for the five instances

	Hand-made taxis		Algorithm TV taxis			TV + refining taxis		
	Tot.		Total	TV	diff.	Total	TV	diff.
# vehic	245	349	155	260	-26%	120	225	-36%
# routes	245	531	155	365	-31%	120	330	-38%
serv trips	331	1496	300	1496	—	237	1496	—
Avg pass	1.4	2.8	2.0	4.1	+46%	2.0	4.6	+64%
Tot trav t	365.7	935.2	208.9	831.6	-11%	135.7	719.9	-23%
Avg trav t	31	51	38	40	-22%	36	40	-22%
Km total	7363	15919	3898	11281	-29%	2424	9956	-37%
Km empty	5234	9619	1456	4142	-57%	809	3227	-66%
Cost	9584	24357	5066	19471	-20%	3053	16552	-32%
CPU time				234.7			934.5	

trips) the use of standard inter-route insertion and exchange neighborhoods or of metaheuristic techniques could be very expensive from the computational point of view. Hence, we have considered restricted neighborhoods which can be examined in an acceptable computing time but allowed for consistent improvement of the initial solutions.

The proposed procedures have been applied to the solutions obtained by a parallel insertion heuristic algorithm proposed by Toth and Vigo when applied to a set of large real problems with about 300 trips each. The computational test shows that the heuristic solutions can be substantially improved, both in terms of overall

cost and of number of vehicles used, in a few minutes on a PC. Future research can be in the direction of including metaheuristic techniques, and of the development of dominance and reduction rules that can speed-up the examination of different neighborhoods in order to obtain better solutions from the refining phase.

5. References

- L. Bodin, B. Golden, A. Assad and M. Ball, Routing and Scheduling of Vehicles and Crews, The State of the Art, *Computers and Operations Research*, 10(2) (1983) p. 63.
- G. Carpaneto, S. Martello and P. Toth, Algorithms and Codes for the Assignment Problem in: *FORTRAN Codes for Network Optimization*, eds. B. Simeone, P. Toth, G. Gallo, F. Maffioli, S. Pallottino, *Annals of Operations Research* 13 (1988).
- Y. Dumas, J. Desrosiers and F. Soumis, The Pickup and Delivery Problem with Time Windows, *European Journal of Operational Research* 54 (1991) p. 7.
- I. H. Osman, Metastrategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem, *Annals of Operations Research* 41 (1993) p. 421.
- S. Roy, J.-M. Rousseau, G. Lapalme and J. Ferland, Routing and Scheduling for the Transportation of Disabled Persons – The Algorithm, Working Paper, TP5596E, Transport Development Center, Montréal, Canada (1984).
- M. Solomon, E. Baker and J. Schaffer, Vehicle Routing and Scheduling Problems with Time Windows Constraints: Efficient Implementations of Solution Improvement Procedures, in: *Vehicle Routing: Methods and Studies*, eds. B. Golden and A. Assad (1988) p. 85.
- M. Solomon and J. Desrosiers, Time Window Constrained Routing and Scheduling Problems, *Transportation Science* 22(1) (1988) p. 1.
- C. Sutcliffe and J. Board, Optimal Solution of a Vehicle-routeing Problem: Transporting Mentally Handicapped Adults to an

Adult Training Centre, *Journal of the Operational Research Society* 41(1) (1990) p. 61.

- P. Thompson and H. Psaraftis, Cyclic Transfer Algorithms for Multivehicle Routing and Scheduling Problems, *Operations Research* 41(5) (1993) p. 935.
- P. Toth and D. Vigo, A Heuristic Vehicle Scheduling Algorithm for the Transportation of Handicapped Persons, in: *Proceedings of the IIIrd Meeting of the EURO Working Group on Urban Traffic and Transportation*, Paris (1993).

Acknowledgments

This work has been partially supported by Ministero dell'Università e della Ricerca Scientifica e Tecnologica. We thank the Municipality of Bologna and Sisplan srl for providing the numerical input data used for the algorithm testing of Section 3.